ST. CLAIR

COLLEGE

ST. CLAIR COLLEGE

WEB315

Introduction to ASP.NET

Week 10

RISE ABOVE THE ORDINARY

Blazor & Real-time Apps

- Blazor is a framework for building interactive client-side web UI with .NET
- This can also extend to building real-time Apps with SignalR.
- What is SignalR?
  - ASP.NET Core SignalR is an open-source "middleware" library that simplifies adding real-time web functionality to apps. Real-time web functionality enables server-side code to push content to clients instantly.

Source:

Blazor & Real-time Apps (2)

- Good candidates for SignalR:
  - Apps that require high frequency updates from the server. Examples are gaming, social networks, voting, auction, maps, and GPS apps.
  - Dashboards and monitoring apps. Examples include company dashboards, instant sales updates, or travel alerts.
  - Collaborative apps. Whiteboard apps and team meeting software are examples of collaborative apps.
  - Apps that require notifications. Social networks, email, chat, games, travel alerts, and many other apps use notifications.

Source:

SignalR

- SignalR provides an API (Application Programming Interface) for creating server-to-client remote procedure calls (RPC).
- The RPCs call JavaScript functions on clients from server-side .NET Core code.
- Features of SignalR for ASP.NET Core:
  - Handles connection management automatically.
  - Sends messages to all connected clients simultaneously. For example, a chat room.
  - Sends messages to specific clients or groups of clients.
  - Scales to handle increasing traffic.

Source:

SignalR (2)

- In order to handle real-time communication between clients and servers, SignalR supports the following techniques:
- Transport:
  - WebSockets
  - Server-Sent Events
  - Long Polling
- Hubs:
  - A high-level pipeline for clients and servers to call methods on each other
  - Has two built-in hub protocols: text based (JSON) and binary (MessagePack)

Source:

SignalR (3)

- The SignalR Hubs API enables methods to be called on connected clients from the server.
- In the server code, methods are defined that are called by client.
- In the client code, methods are defined that are called from the server.
- SignalR takes care of everything behind the scenes that makes real-time client-to-server and server-to-client communications possible.

Source:

SignalR (4)

- The SignalR middleware requires some services, which are configured by calling **`services.AddSignalR`**.
- When adding SignalR functionality to an ASP.NET Core app, setup SignalR routes by calling **`endpoint.MapHub`** in the **`Startup.Configure`** method's **`app.UseEndpoints`** callback:

```
app.UseRouting();
app.UseEndpoints(endpoints =>
{
endpoints.MapHub<ChatHub>("/chathub");
});
```

Source:

SignalR (5)

- Create a hub by declaring a class that inherits from **`Hub`** and add public methods to it. Clients can call methods that are defined as **`public`**:

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

*Hubs are transient:
Don't store state in a property on the hub class. Every hub method call is executed on a new hub instance.
Use **`await`** when calling asynchronous methods that depend on the hub staying alive.
For example, a method such as **`Clients.All.SendAsync(...)`** can fail if it's called without **`await`** and the hub method completes before **`SendAsync`** finishes.

Source:

SignalR (6)

- The **`Hub`** class has a **`Context`** property that contains the following properties with information about the connection:

Property	**Description**
`ConnectionId`	Gets the unique ID for the connection, assigned by SignalR. There is one connection ID for each connection.
`UserIdentifier`	Gets the user identifier. By default, SignalR uses the `ClaimTypes.NameIdentifier` from the `ClaimsPrincipal` associated with the connection as the user identifier.
`User`	Gets the `ClaimsPrincipal` associated with the current user.
`Items`	Gets a key/value collection that can be used to share data within the scope of this connection. Data can be stored in this collection and it will persist for the connection across different hub method invocations.
`Features`	Gets the collection of features available on the connection.
`ConnectionAborted`	Gets a `CancellationToken` that notifies when the connection is aborted.

Source:

SignalR (7)

- `Hub.Context` also contains the following methods:

Method	**Description**
`GetHttpContext`	Returns the `HttpContext` for the connection, or `null` if the connection is not associated with an HTTP request. For HTTP connections, you can use this method to get information such as HTTP headers and query strings.
`Abort`	Aborts the connection.

Source:

SignalR (8)

- The `Hub` class has a `Clients` property that contains the following properties for communication between server and client:

Property	Description
`All`	Calls a method on all connected clients
`Caller`	Calls a method on the client that invoked the hub method
`Others`	Calls a method on all connected clients except the client that invoked the method

Source:

SignalR (9)

*Note: Each property or method in the tables returns an object with a `SendAsync` method. The `SendAsync` method allows you to supply the name and parameters of the client method to call.

- `Hub.Client` also contains the following methods:

Method	**Description**
`AllExcept`	Calls a method on all connected clients except for the specified connections
`Client`	Calls a method on a specific connected client
`Clients`	Calls a method on specific connected clients
`Group`	Calls a method on all connections in the specified group
`GroupExcept`	Calls a method on all connections in the specified group, except the specified connections
`Groups`	Calls a method on multiple groups of connections
`OthersInGroup`	Calls a method on a group of connections, excluding the client that invoked the hub method
`User`	Calls a method on all connections associated with a specific user
`Users`	Calls a method on all connections associated with a specific users

Source:

SignalR (10)

- To make calls to specific clients, use the properties of the `Clients` object. In the following example, there are three Hub methods:
  - `SendMessage` sends a message to all connected clients, using `Clients.All`
  - `SendMessageToCaller` sends a message back to the caller, using `Clients.Caller`
  - `SendMessageToGroup` sends a message to all clients in the `SignalR` Users group

* See next slide for code examples

Source:

SignalR (11)

```
public Task SendMessage(string user, string message)
{
    return Clients.All.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToCaller(string user, string message)
{
    return Clients.Caller.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToGroup(string user, string message)
{
    return Clients.Group("SignalR Users").SendAsync("ReceiveMessage", user, message);
}
```

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

1. Create a Blazor project
2. Add the SignalR client library
3. Add a SignalR hub
4. Add SignalR services and an endpoint for the SignalR hub
5. Add Razor component code for chat

*Note: This is the 'Blazor WebAssembly' version

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

1. Create a hosted Blazor WebAssembly app:

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

  - The `-ho|--hosted` option creates a hosted Blazor WebAssembly solution.
  - 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.

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

2. Add the SignalR client library:

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

3. Add a SignalR hub:
   - In the BlazorWebAssemblySignalRApp.Server project, create a Hubs (plural) folder and add the following ChatHub class (Hubs/ChatHub.cs):

```
using System.Threading.Tasks;
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);
        }
    }
}
```

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

4. Add services and an endpoint for the SignalR hub:
   - In the `BlazorWebAssemblySignalRApp.Server` project, open the `Startup.cs` file.
   - Add the namespace for the ChatHub class to the top of the file:
     `using BlazorWebAssemblySignalRApp.Server.Hubs;`
   - Add SignalR and Response Compression Middleware services to `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSignalR();
    services.AddControllersWithViews();
    services.AddRazorPages();
    services.AddResponseCompression(opts =>
    {
        opts.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(
            new[] { "application/octet-stream" });
    });
}
```

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:
  4. Add services and an endpoint for the SignalR hub:
     - In `Startup.Configure`:
       - Use Response Compression Middleware at the top of the processing pipeline's configuration.
       - Between the endpoints for controllers and the client-side fallback, add an endpoint for the hub.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
{
    app.UseResponseCompression();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseWebAssemblyDebugging();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseBlazorFrameworkFiles();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
        endpoints.MapControllers();
        endpoints.MapHub<ChatHub>("/chathub");
        endpoints.MapFallbackToFile("index.html");
    });
}
```

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

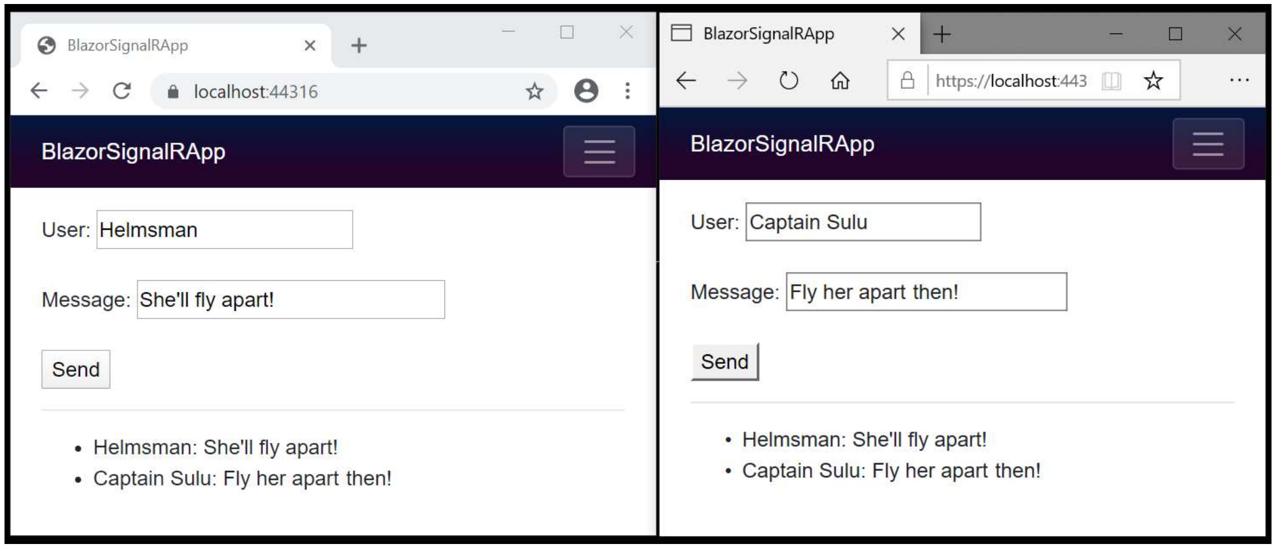
  5. Add Razor component code for chat
     - In the `BlazorWebAssemblySignalRApp.Client` project, open the `Pages/Index.razor` file.
     - Modify the markup with the code provided in the tutorial source link below.

Source:

SignalR Tutorial

- Use ASP.NET Core SignalR with Blazor:

- Run the app:
  - Press F5 to run the app with debugging or Ctrl+F5 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:



BlazorSignalRApp
localhost:44316
BlazorSignalRApp
User: Helmsman
Message: She'll fly apart!
Send
• Helmsman: She'll fly apart!
• Captain Sulu: Fly her apart then!
BlazorSignalRApp
https://localhost:443
BlazorSignalRApp
User: Captain Sulu
Message: Fly her apart then!
Send
• Helmsman: She'll fly apart!
• Captain Sulu: Fly her apart then!

Source:

ST. CLAIR

COLLEGE

RISE ABOVE THE ORDINARY

stclaircollege.ca |

ST. CLAIR COLLEGE
ZEKELMAN
SCHOOL OF
BUSINESS
& INFORMATION TECHNOLOGY