SignalR Core

ASP.NET Core SignalR

- **ASP.NET Core SignalR** is an open-source library that simplifies adding real-time web functionality to apps. Real-time web functionality enables server-side code to push content to clients instantly.

- SignalR provides an API for creating server-to-client ***remote procedure calls (RPC).*** The RPCs call JavaScript functions on clients from server-side .NET Core code.

Transports

SignalR supports the following techniques for handling real-time communication (in order of graceful fallback):

- WebSockets
- Server-Sent Events
- Long Polling

SignalR automatically chooses the best transport method that is within the capabilities of the server and client.

Configure SignalR(startup class)

- **ConfigureServices** :

  services.AddSignalR();

- **Configure:**

  app.MapHub<ChatHub>("/chatHub");

Create a SignalR hub

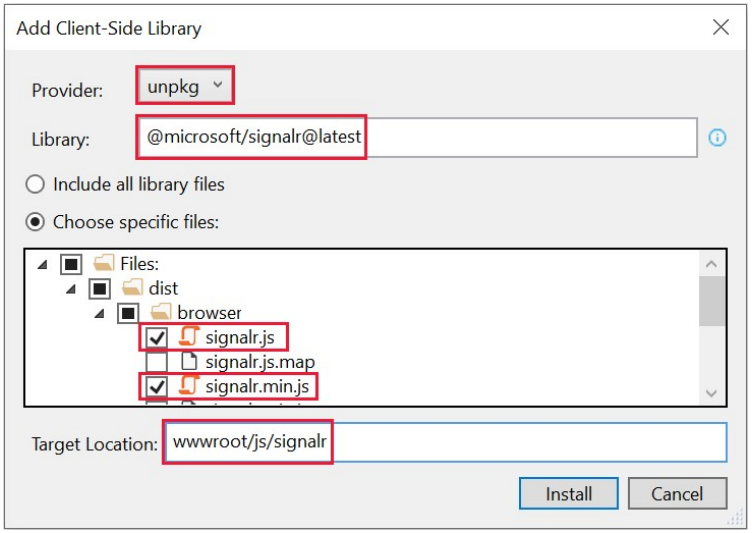
- In the SignalRChat project folder, create a *Hubs* folder.
- In the *Hubs* folder, create a *ChatHub.cs*

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

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

Add the SignalR client library

- In **Solution Explorer**, right-click the project, and select **Add** > **Client-Side Library**.



- **Use a Content Delivery Network (CDN)**

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/microsoft-signalr/6.0.1/signalr.js"></script>
```

Add SignalR client code

```
<script src="~/js/signalr/dist/browser/signalr.js"></script>
<script>
    //define connection
    var connection = new signalR.HubConnectionBuilder()
        .withUrl("/chatHub").build();

    //start connection
    connection.start()

    //define callback fun
    connection.on("ReceiveMessage", function (user, message) {
        //anycode
    });

    //call server method
    connection.invoke("SendMessage", user, message)

</script>
```

The Clients object

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
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 the specified users

Groups in SignalR

- A group is a collection of connections associated with a name.
- Messages can be sent to all connections in a group.
- A connection can be a member of multiple groups.
- Connections are added to or removed from groups via the AddToGroupAsync and RemoveFromGroupAsync methods.

ASP.NET Core SignalR .NET Client

- The *Microsoft.AspNetCore.SignalR.Client* package is required for .NET clients to connect to SignalR hubs.

```
HubConnection con;
1 reference
public Form1()
{
    InitializeComponent();
     con = new HubConnectionBuilder()
        .WithUrl("https://localhost:7269/chat").Build();

    con.StartAsync();
    con.On<string, string>("newmess", (n, m) => listBox1.Items.Add(n + m));
}

1 reference
private void button1_Click(object sender, EventArgs e)
{
    con.InvokeAsync("sendmessage", "ali", textBox1.Text);
}
```

Handle lost connection (Automatically reconnect)

- Without any parameters, WithAutomaticReconnect() configures the client to wait 0, 2, 10, and 30 seconds respectively before trying each reconnect attempt, stopping after four failed attempts.

```
HubConnection connection= new HubConnectionBuilder()
    .WithUrl(new Uri("http://127.0.0.1:5000/chathub"))
    .WithAutomaticReconnect()
    .Build();
```

Handle events for a connection

- The SignalR Hubs API provides the OnConnectedAsync and OnDisconnectedAsync virtual methods to manage and track connections

```
public override async Task OnConnectedAsync()
{
    await Groups.AddToGroupAsync(Context.ConnectionId, "SignalR Users");
    await base.OnConnectedAsync();
}
```

Send messages from outside a hub

- The SignalR hub is the core abstraction for sending messages to clients connected to the SignalR server.
- It's also possible to send messages from other places in your app using the *IHubContext* service.
- in ASP.NET Core SignalR, you can access an instance of IHubContext via dependency injection. You can inject an instance of IHubContext into a controller, middleware, or other DI service

Send messages from outside a hub

```
public class HomeController : Controller
{
    private readonly IHubContext<NotificationHub> _hubContext;

    public HomeController(IHubContext<NotificationHub> hubContext)
    {
        _hubContext = hubContext;
    }

    public async Task<IActionResult> Index()
    {
        await _hubContext.Clients.All.SendAsync("Notify", $"Home page loaded at: {DateTime.Now}");
        return View();
    }
```

Cross-origin connections (CORS)

```
builder.Services.AddCors(options => {
            options.AddDefaultPolicy( builder => {
            builder.WithOrigins("https://example.c
            om") .AllowAnyHeader() .
            WithMethods("GET", "POST")
            .AllowCredentials();
    });
});
```

```
// UseCors must be called before MapHub.
            app.UseCors();
```

Change the name of a hub method

- By default, a server hub method name is the name of the .NET method. To change this default behavior for a specific method, use the *HubMethodName* attribute.
- The client should use this name instead of the .NET method name when invoking the method

```
[HubMethodName("SendMessageToUser")]
public async Task DirectMessage(string user, string message)
    => await Clients.User(user).SendAsync("ReceiveMessage", user, message);
```

Strongly typed hubs

- A drawback of using SendAsync is that it relies on a string to specify the client method to be called. This leaves code open to runtime errors if the method name is misspelled or missing from the client

```
public interface IChatClient
{
    Task ReceiveMessage(string user, string message);
}
```

```
public class StronglyTypedChatHub : Hub<IChatClient>
{
    public async Task SendMessage(string user, string message)
        => await Clients.All.ReceiveMessage(user, message);

    public async Task SendMessageToCaller(string user, string message)
        => await Clients.Caller.ReceiveMessage(user, message);

    public async Task SendMessageToGroup(string user, string message)
        => await Clients.Group("SignalR Users").ReceiveMessage(user, message);
}
```