2100030320

M. Bhavana

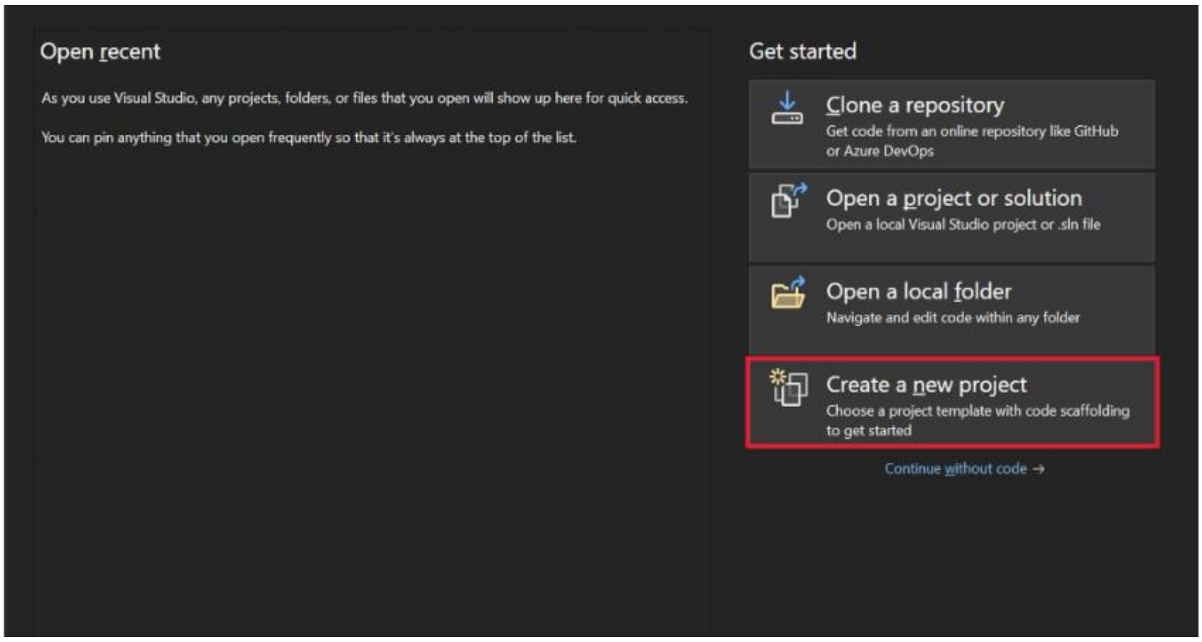
EPAM-LAB_10

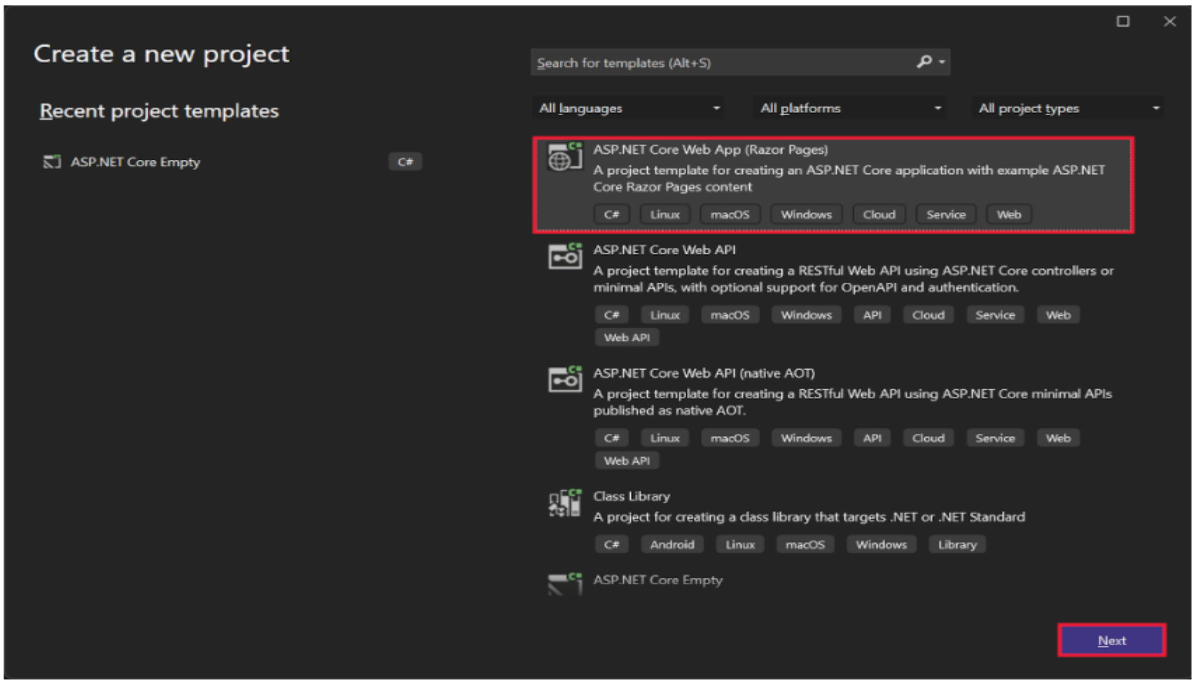
(Asp.Net using State management Technique)

Implement a website for Chatting using Asp.net web forms. By taking the following requirements.

i) Create one login form with 1 label, 1 TextBox and 1 button
ii) Create one ChatPage.html which will be open when user clicks on login button
iii) Create Message.aspx for sending user given messages to the Application Object
iv) Create Display.aspx for displaying all the messages in Chat Area from the Application object

Hint: Before sending messages to the Application, you must store Some messages in the Application object within Global.asax file. And also you must store UserName in the Session also.


Open recent
As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.
You can pin anything that you open frequently so that it's always at the top of the list.
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →


Create a new project

Recent project templates

ASP.NET Core Empty C#

Search for templates (Alt+S)

All languages · All platforms · All project types

ASP.NET Core Web App (Razor Pages)
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# Linux macOS Windows API Cloud Service Web Web API

ASP.NET Core Web API (native AOT)
A project template for creating a RESTful Web API using ASP.NET Core minimal APIs published as native AOT.
C# Linux macOS Windows API Cloud Service Web Web API

Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library

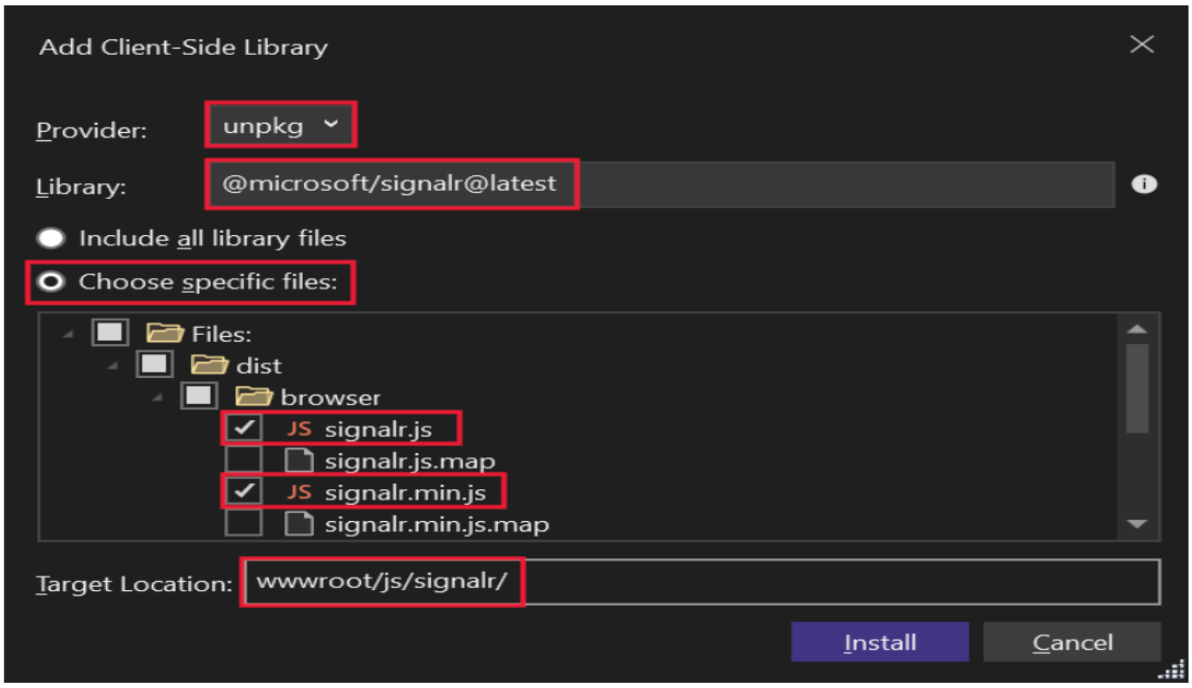
ASP.NET Core Empty

Next

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

In the **Add Client-Side Library** dialog:

- Select **unpkg** for **Provider**
- Enter `@microsoft/signalr@latest` for **Library**.
- Select **Choose specific files**, expand the *dist/browser* folder, and select `signalr.js` and `signalr.min.js`.
- Set **Target Location** to `wwwroot/js/signalr/`.
- Select **Install**.


Add Client-Side Library

Provider: unpkg

Library: @microsoft/signalr@latest

Include all library files

Choose specific files:

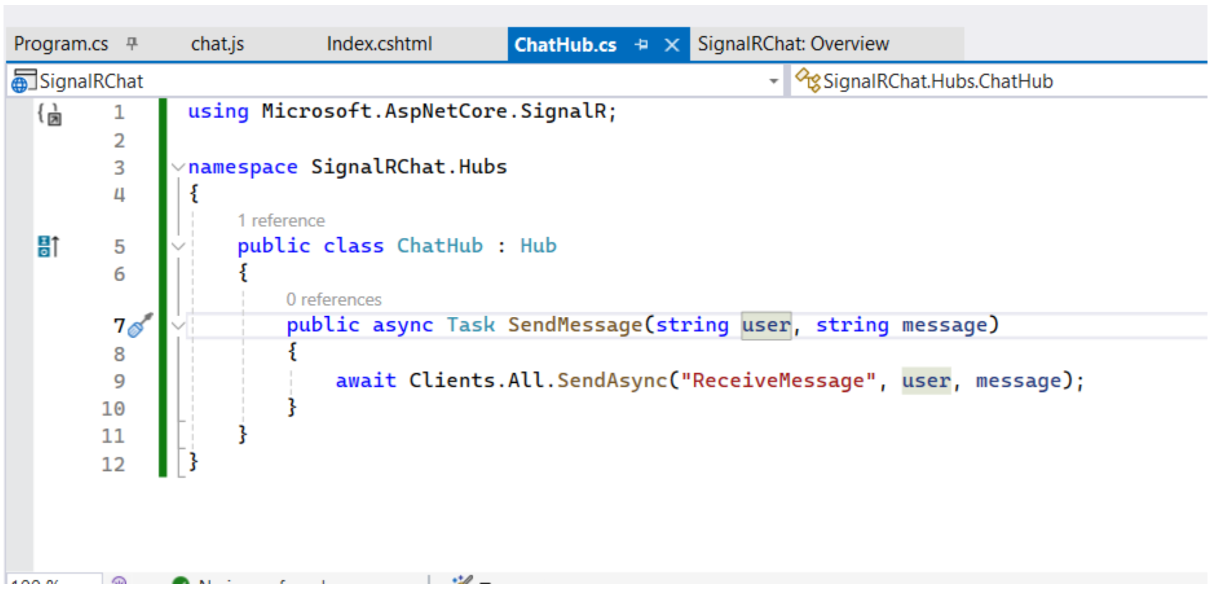
Files:
- dist
  - browser
    - [✓] signalr.js
    - [ ] signalr.js.map
    - [✓] signalr.min.js
    - [ ] signalr.min.js.map

Target Location: wwwroot/js/signalr/

Install · Cancel

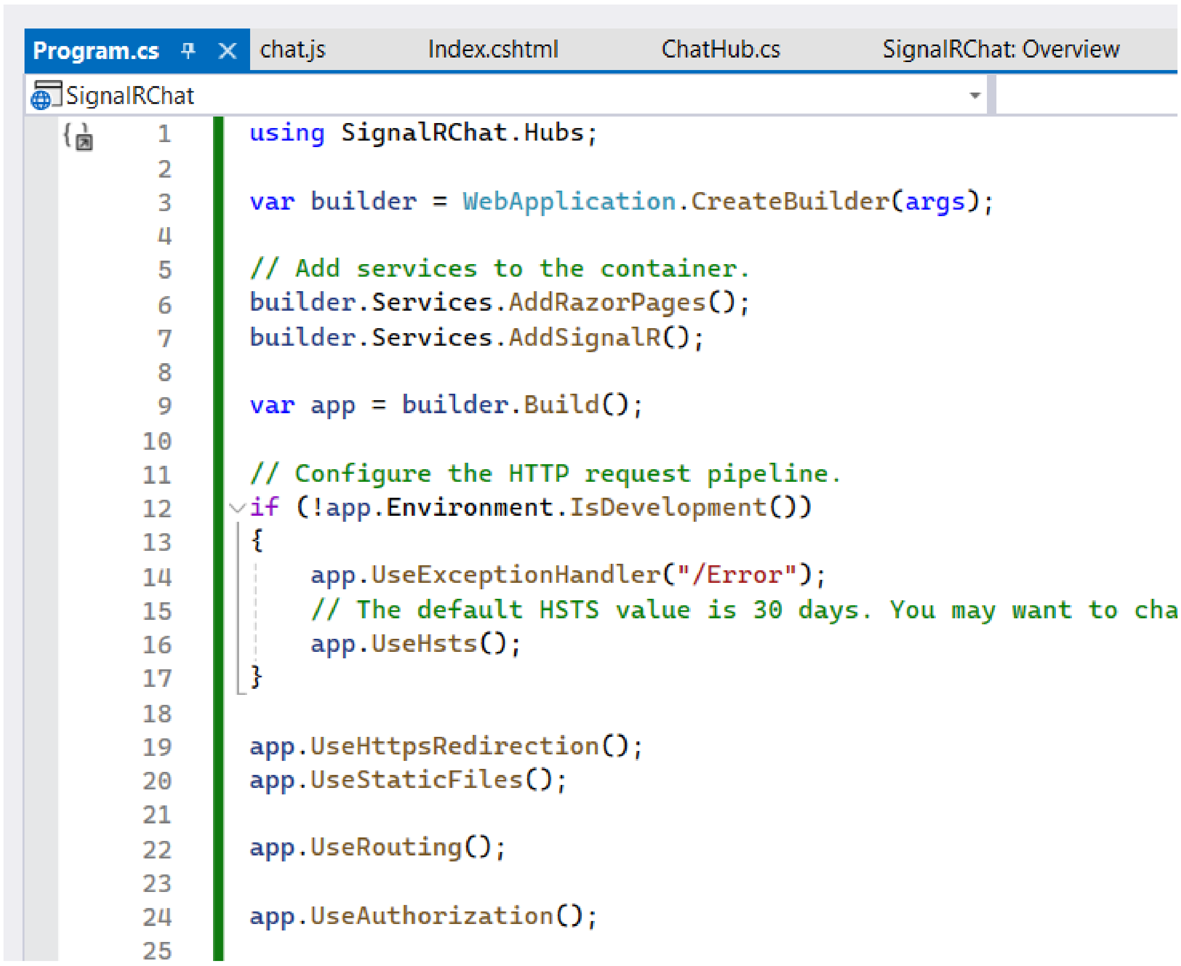
In the SignalRChat project folder, create a `Hubs` folder.

In the `Hubs` folder, create the `ChatHub` class with the following code:

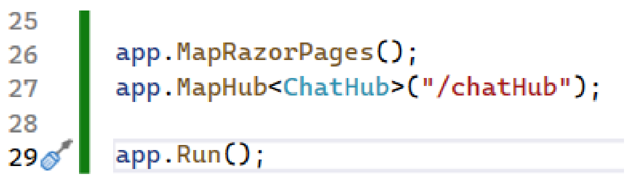


```
using Microsoft.AspNetCore.SignalR;

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

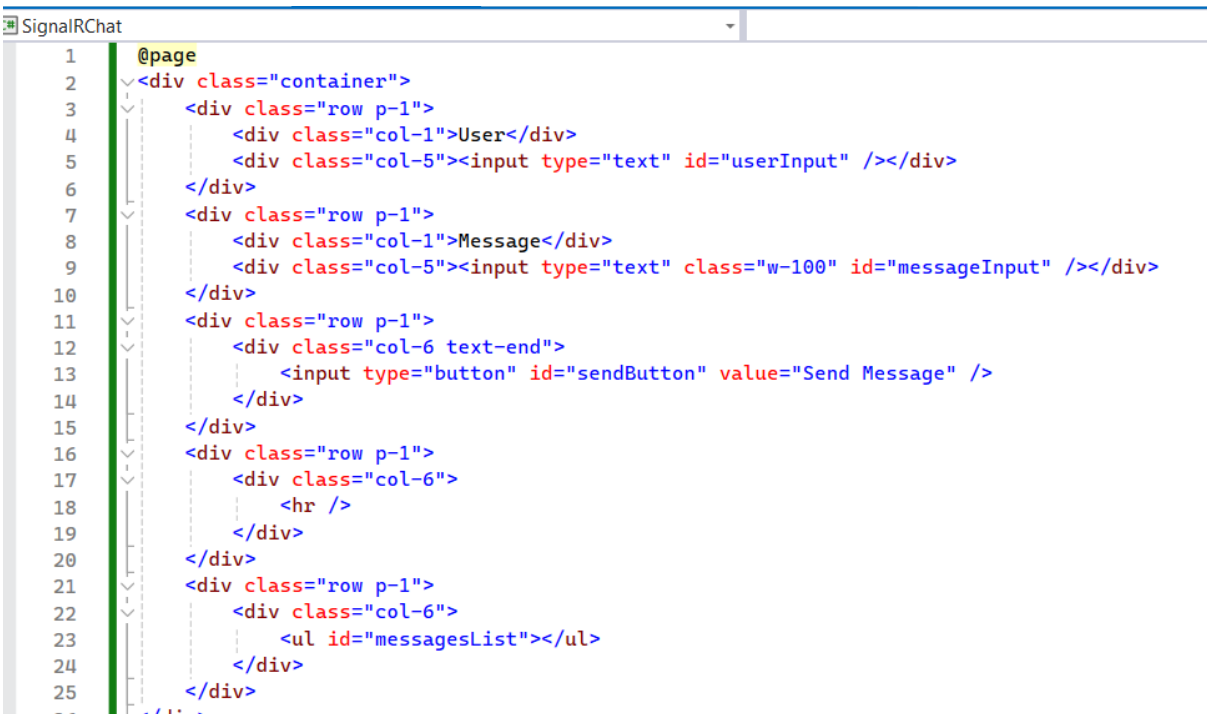
The SignalR server must be configured to pass SignalR requests to SignalR. Add the following code to the `Program.cs` file.



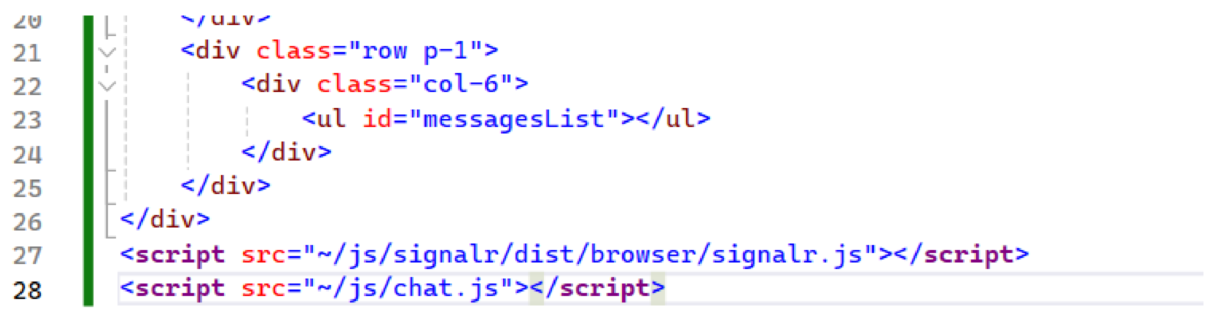
```
using SignalRChat.Hubs;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddSignalR();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to cha
    app.UseHsts();
}

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

app.UseRouting();

app.UseAuthorization();

```


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

app.Run();
```

Replace the content in `Pages/Index.cshtml` with the following code:



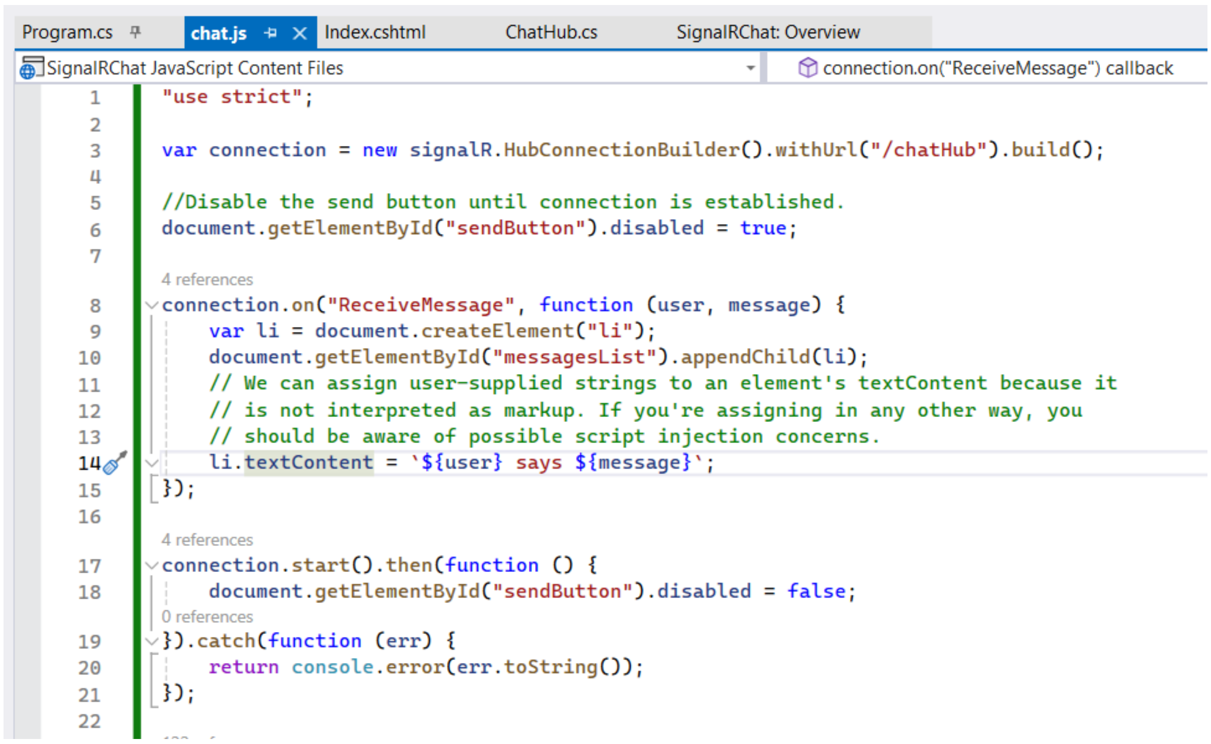
```
@page
<div class="container">
    <div class="row p-1">
        <div class="col-1">User</div>
        <div class="col-5"><input type="text" id="userInput" /></div>
    </div>
    <div class="row p-1">
        <div class="col-1">Message</div>
        <div class="col-5"><input type="text" class="w-100" id="messageInput" /></div>
    </div>
    <div class="row p-1">
        <div class="col-6 text-end">
            <input type="button" id="sendButton" value="Send Message" />
        </div>
    </div>
    <div class="row p-1">
        <div class="col-6">
            <hr />
        </div>
    </div>
    <div class="row p-1">
        <div class="col-6">
            <ul id="messagesList"></ul>
        </div>
    </div>
```



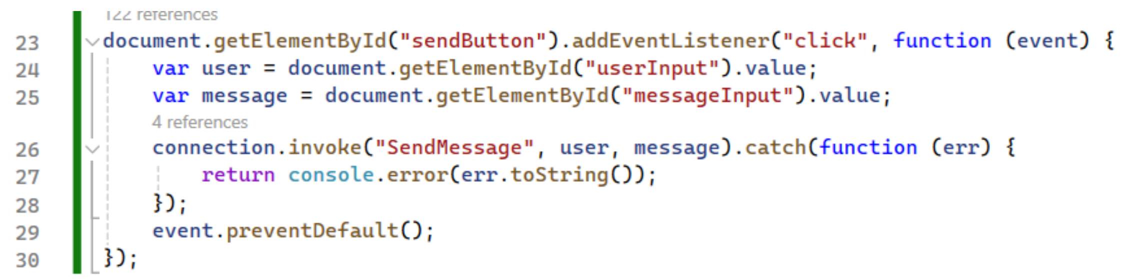
```
    <div class="row p-1">
        <div class="col-6">
            <ul id="messagesList"></ul>
        </div>
    </div>
```

```
</div>
<script src="~/js/signalr/dist/browser/signalr.js"></script>
<script src="~/js/chat.js"></script>
```

In the `wwwroot/js` folder, create a `chat.js` file with the following code:



```
"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

//Disable the send button until connection is established.
document.getElementById("sendButton").disabled = true;

connection.on("ReceiveMessage", function (user, message) {
    var li = document.createElement("li");
    document.getElementById("messagesList").appendChild(li);
    // We can assign user-supplied strings to an element's textContent because it
    // is not interpreted as markup. If you're assigning in any other way, you
    // should be aware of possible script injection concerns.
    li.textContent = `${user} says ${message}`;
});

connection.start().then(function () {
    document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
    return console.error(err.toString());
});

```



```
document.getElementById("sendButton").addEventListener("click", function (event) {
    var user = document.getElementById("userInput").value;
    var message = document.getElementById("messageInput").value;
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});
```

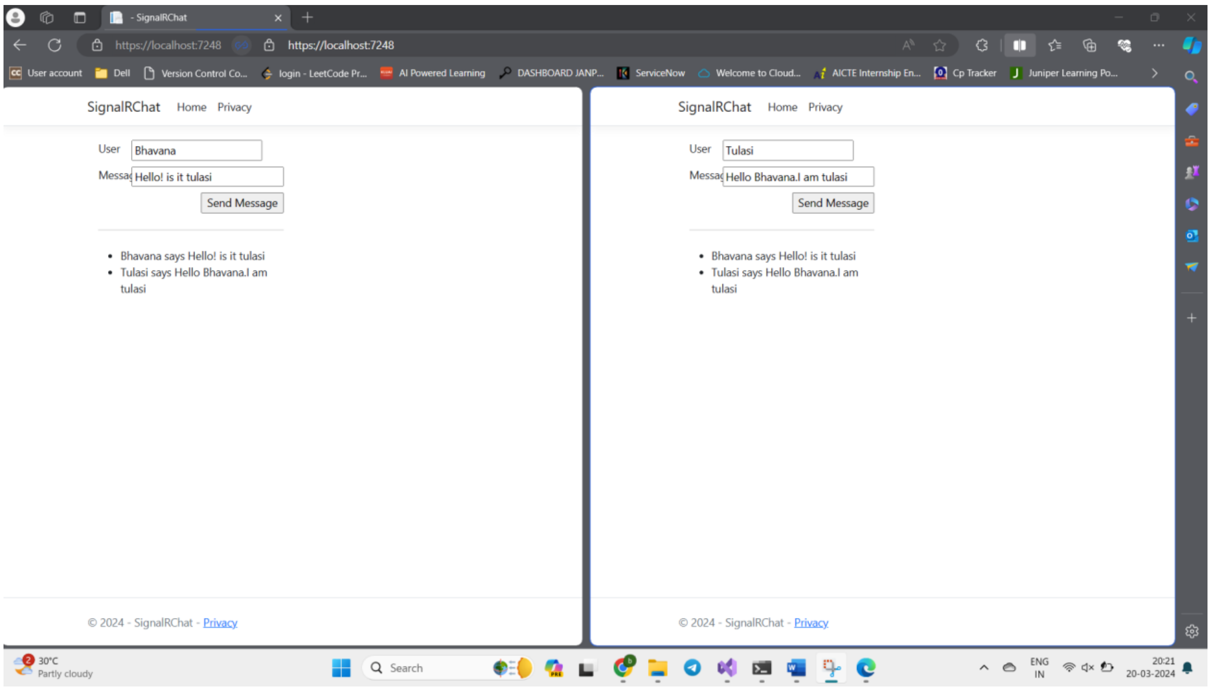
Run the app

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 **Send Message** button.

The name and message are displayed on both pages instantly.

Output:



SignalRChat Home Privacy

User Bhavana

Message Hello! is it tulasi

Send Message

- Bhavana says Hello! is it tulasi
- Tulasi says Hello Bhavana.I am tulasi

© 2024 - SignalRChat - Privacy

SignalRChat Home Privacy

User Tulasi

Message Hello Bhavana.I am tulasi

Send Message

- Bhavana says Hello! is it tulasi
- Tulasi says Hello Bhavana.I am tulasi

© 2024 - SignalRChat - Privacy