Table of Contents

[Basics 2](#_Toc118039273)

[Current Resources with Changes 2](#_Toc118039274)

[New Resources 2](#_Toc118039275)

[/Core/SignalR 2](#_Toc118039276)

[PresenseTracker.cs 2](#_Toc118039277)

[PresenceHub.cs 3](#_Toc118039278)

[MessageHub.cs 4](#_Toc118039279)

[/Core/Extensions 5](#_Toc118039280)

[ServiceIdentityExtensions.cs 5](#_Toc118039281)

[ServiceExtensions.cs 6](#_Toc118039282)

[Programs.cs 7](#_Toc118039283)

# Basics

|  |  |
| --- | --- |
| WorkingFolder | Copy the content of “Site-05-Identity Role Management” in “Site-06-SignalR” and issue   * dotnet build |
| Cloudinary Setting | appsetting.json is not checked in so make sure to bring in this from “Site-01-Basics” |

# Current Resources with Changes

1. Programs.cs
2. /Core/Extensions/ServiceIdentityExtensions
3. /Core/Extensions/ServiceExtensions

# New Resources

1. /Core/SignalR/PresenceTracker
2. /Core/SignalR/PresenceHub
3. /Core/SignalR/MessageHub

# /Core/SignalR

Create a new folder under core as SignalR

## PresenseTracker.cs

A class to track the users

//1. starting with local presence tracker. The elaborate one would be with redis or in database

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MSC.Api.Core.SignalR;

/// <summary>

/// Add the service as singleton in

/// </summary>

public class PresenceTracker

{

    //dictionary to hold the userKey and the list of connections

    private static readonly Dictionary<string, List<string>> \_onlineUsers = new Dictionary<string, List<string>>();

    /// <summary>

    /// Add to the user and connectionId on login

    /// </summary>

    /// <param name="userName"></param>

    /// <param name="connectionId"></param>

    /// <returns></returns>

    public Task UserConnected(string userName, string connectionId)

    {

        lock (\_onlineUsers)

        {

            if (\_onlineUsers.ContainsKey(userName))

            {

                //add the connectionId to the second part of the dictionary

                \_onlineUsers[userName].Add(connectionId);

            }

            else

            {

                //add the the user name with the connection id

                \_onlineUsers.Add(userName, new List<string> { connectionId });

            }

        }

        return Task.CompletedTask;

    }

    /// <summary>

    /// Remove the user and connection on logout

    /// </summary>

    /// <param name="userName"></param>

    /// <param name="connectionId"></param>

    /// <returns></returns>

    public Task UserDisconnected(string userName, string connectionId)

    {

        lock (\_onlineUsers)

        {

            if (!\_onlineUsers.ContainsKey(userName))

            {

                return Task.CompletedTask;

            }

            //remove the connection

            \_onlineUsers[userName].Remove(connectionId);

            if (\_onlineUsers[userName].Count == 0)

            {

                //remove the user

                \_onlineUsers.Remove(userName);

            }

        }

        return Task.CompletedTask;

    }

    /// <summary>

    /// Get the list of online users

    /// </summary>

    /// <returns></returns>

    public Task<string[]> GetOnlineUsers()

    {

        string[] onlineUsers;

        lock (\_onlineUsers)

        {

            onlineUsers = \_onlineUsers.OrderBy(k => k.Key).Select(k => k.Key).ToArray();

        }

        return Task.FromResult(onlineUsers);

    }

}

## PresenceHub.cs

using System;

using System;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.SignalR;

using MSC.Api.Core.Extensions;

//no package to install for SignalR

namespace MSC.Api.Core.SignalR;

[Authorize]

/// <summary>

/// PresenceHub, it derives from Hub and then override the virtual methods

/// </summary>

public class PresenceHub : Hub

{

    private readonly PresenceTracker \_tracker;

    public PresenceHub(PresenceTracker tracker)

    {

        \_tracker = tracker;

    }

    /// <summary>

    /// Implement OnConnectedAsync to tell other users when the current user goes online

    /// </summary>

    /// <returns></returns>

    public override async Task OnConnectedAsync()

    {

        //pick the current user from Token ==> claims

        //add the user to presenceTracker

        //other than the current user tell all others that the user is online

        var userName = Context.User.GetUserName();

        var connectionId = Context.ConnectionId;

        await \_tracker.UserConnected(userName, connectionId);

        await Clients.Others.SendAsync("UserIsOnline", userName);

        //get the users online and send to every one who is connected

        var currentUsers = await \_tracker.GetOnlineUsers();

        await Clients.All.SendAsync("GetOnlineUsers", currentUsers);

    }

    /// <summary>

    /// Implmenent OnDisconnectedAsync to tell other users when the current user goes offline

    /// </summary>

    /// <param name="exception"></param>

    /// <returns></returns>

    public override async Task OnDisconnectedAsync(Exception exception)

    {

        //pick the current user from Token ==> claims

        //remove the user from presenceTracker

        //tell all other users when the use goes offline

        var userName = Context.User.GetUserName();

        var connectionId = Context.ConnectionId;

        await \_tracker.UserDisconnected(userName, connectionId);

        await Clients.Others.SendAsync("UserIsOffline", Context.User.GetUserName());

        //get the users online and send to every one who is connected

        var currentUsers = await \_tracker.GetOnlineUsers();

        await Clients.All.SendAsync("GetOnlineUsers", currentUsers);

        await base.OnDisconnectedAsync(exception);

    }

}

# MessageHub.cs

using System;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.SignalR;

using MSC.Api.Core.BusinessLogic;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Extensions;

//no package to install for SignalR

namespace MSC.Api.Core.SignalR;

[Authorize]

/// <summary>

/// MessageHub, it derives from Hub and then override the virtual methods

/// </summary>

public class MessageHub : Hub

{

    private readonly IMessageBusinessLogic \_msgBl;

    public MessageHub(IMessageBusinessLogic msgBl)

    {

        \_msgBl = msgBl;

    }

    /// <summary>

    /// Implement OnConnectedAsync

    /// </summary>

    /// <returns></returns>

    public override async Task OnConnectedAsync()

    {

        //create a group of two users. will be passing in the other users name

        var httpContext = Context.GetHttpContext();

        var otherUserName = httpContext.Request.Query["otherUserName"].ToString();

        var otherUserId = int.Parse(httpContext.Request.Query["otherUserId"].ToString());

        //build the group name

        var groupName = GetGroupName(Context.User.GetUserName(), otherUserName);

        //add to group

        await Groups.AddToGroupAsync(Context.ConnectionId, groupName);

        //when the user joins the group get the messages

        var messages = await \_msgBl.GetMessageThread(Context.User.GetUserId(), otherUserId);

        //send the message

        await Clients.Group(groupName).SendAsync("ReceiveMessageThread", messages);

    }

    /// <summary>

    /// Implmenent OnDisconnectedAsync

    /// </summary>

    /// <param name="exception"></param>

    /// <returns></returns>

    public override async Task OnDisconnectedAsync(Exception exception)

    {

        //users will be automatically removed from the group

        await base.OnDisconnectedAsync(exception);

    }

    //copied the code from the MessageController CreateMessage action

    public async Task SendMessage(MessageCreateDto msg)

    {

        //get the claims

        var userClaims = Context.User.GetUserClaims();

        if (userClaims == null || (!userClaims.HasGuid || !userClaims.HasUserName))

            throw new HubException("User issue");

        var result = await \_msgBl.AddMessage(msg, userClaims.UserId);

        if (result == null)

            throw new HubException("Unable to send message");

        if (result.HttpStatusCode != HttpStatusCode.OK)

            throw new HubException(result.Message ?? "Unable to send message");

        var message = result.ConvertDataToType<MessageDto>();

        var groupName = GetGroupName(message.SenderUsername, message.ReceipientUsername);

        await Clients.Group(groupName).SendAsync("NewMessage", message);

    }

    private string GetGroupName(string caller, string other)

    {

        //Less than zero –strA is less than strB.

        //Zero –strA and strB are equal.

        //Greater than zero –strA is greater than strB

        var stringCompare = string.CompareOrdinal(caller, other) < 0;

        return stringCompare ? $"{caller}-{other}" : $"{other}-{caller}";

    }

}

# /Core/Extensions

## ServiceIdentityExtensions.cs

SignalR or websockets cannot send authentication header. Have to use query string with SignalR. Add JwtBearerEvents to pick token from query string

        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

                .AddJwtBearer(options =>

                {

                    options.TokenValidationParameters = new TokenValidationParameters

                    {

                        ValidateIssuerSigningKey = true,

                        IssuerSigningKey = new SymmetricSecurityKey(tokenKey),

                        ValidateIssuer = false,

                        ValidateAudience = false

                    };

                    //signalR or websockets cannot send authentication header. Have to use query string with SignalR

                    //signalR, this allows the client to send the token as query string

                    options.Events = new JwtBearerEvents

                    {

                        OnMessageReceived = context =>

                        {

                            //get access\_token from the query

                            var accessToken = context.Request.Query["access\_token"];

                            //check the path of the request and only do for signalr. Our paths starts with hubs/ check programs.cs for details

                            var path = context.HttpContext.Request.Path;

                            if (!string.IsNullOrWhiteSpace(accessToken) && path.StartsWithSegments("/hubs"))

                            {

                                context.Token = accessToken;

                            }

                            return Task.CompletedTask;

                        }

                    };

                });

## ServiceExtensions.cs

Add AllowCredentials to cors policy

    public static string RegisterCors(this IServiceCollection services, IConfiguration config)

    {

        var myAllowSpecificOrigins = "\_myAllowSpecificOrigins";

        //https://stackoverflow.com/questions/42858335/how-to-hardcode-and-read-a-string-array-in-appsettings-json

        var allowedSpecificOrigins = config.GetAllowSpecificOrigins();

        if (allowedSpecificOrigins != null && allowedSpecificOrigins.Any())

        {

            services.AddCors(options =>

            {

                options.AddPolicy(name: myAllowSpecificOrigins,

                                policy =>

                                {

                                    policy.WithOrigins(allowedSpecificOrigins.ToArray())

                                    .AllowAnyHeader()

                                    .AllowAnyMethod()

                                    //signalR

                                    .AllowCredentials()

                                    ;

                                });

            });

        }

        return myAllowSpecificOrigins;

    }

Add PresenceTracker as Singleton in RegisterRepos method

    public static void RegisterRepos(this IServiceCollection services, IConfiguration config)

    {

        /\*

        check

        https://stackoverflow.com/questions/69722872/asp-net-core-6-how-to-access-configuration-during-startup

        https://stackoverflow.com/questions/70865207/net-6-stable-iconfiguration-setup-in-program-cs

        https://stackoverflow.com/questions/69722872/asp-net-core-6-how-to-access-configuration-during-startup/70161492?noredirect=1#comment128539331\_70161492

        \*/

        //AddScoped: is for the life time of the request. Use this for the http requests

        //AddTransient: a new instance is provided to every request

        //AddSingleton: objects are the same for every object and every request

        //add singleton items

        //SignalR add presenceTracker as a singleton

        services.AddSingleton<PresenceTracker>();

# Programs.cs

Add SignalR service in programs

//CUSTOM:Start

/\*

Check:

https://stackoverflow.com/questions/69722872/asp-net-core-6-how-to-access-configuration-during-startup

https://stackoverflow.com/questions/70865207/net-6-stable-iconfiguration-setup-in-program-cs

https://stackoverflow.com/questions/69722872/asp-net-core-6-how-to-access-configuration-during-startup/70161492?noredirect=1#comment128539331\_70161492

\*/

IConfiguration configuration = builder.Configuration; // allows both to access and to set up the config

builder.Services.RegisterRepos(configuration);

builder.Services.RegisterDBContext(configuration);

var myAllowSpecificOrigins = builder.Services.RegisterCors(configuration);

builder.Services.AddIdentityServices(configuration);

builder.Services.AddSignalR();

//CUSTOM:End

And then also specify the hub end end points

//CUSTOM: Start

//ordering is important here. UseCors before UseAuthentication and UseAuthentication before UseAuthorization

app.UseCors(myAllowSpecificOrigins);

app.UseAuthentication();

//tell routing about a hub end point and provide a route for accessing PresenceHub

app.MapHub<PresenceHub>("hubs/presence");

app.MapHub<MessageHub>("hubs/message");

//CUSTOM: End