# Project

* Site-16-Api-Ng-sinalR
  + Copied from: Site-15-Api-Ng-identity-role-management
* For the “MySocialConnect-SPA” issue npm install command to install all packages
  + ng serve : to run the spa

# New & Updates

|  |  |
| --- | --- |
| New | Updates |
| /core/services/signalr/presence-hub.service.ts | environments |
| /core/services/signalr/message-hub.service.ts | /core/services/helper.service.ts |
| /core/Strategy/customRouteReuseStrategy.ts | /core/serices/account.service.ts |
| /core/models-interfaces/signalr/signalr-connection.model.ts | /site/members/member-card |
| /core/models-interfaces/signalr/signalr-group.model.ts | /site/members/member-detail |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Clear Database

If you want to clear database then drop it and recreate it

dotnet ef database drop

dotnet ef database update

# Package Install

> npm install @microsoft/signalr

# environments

Add the base url to the hub as specified on the webapi/programs.cs

## environment.development.ts

  webApiBaseHubsUrlHttps: 'https://localhost:5001/hubs/',

  webApiBaseHubsUrlHttp: 'http://localhost:5000/hubs/',

## environment.ts

  webApiBaseHubsUrlHttps: 'hubs/',

  webApiBaseHubsUrlHttp: 'hubs/',

# /core/services

## helper.service.ts

At the top, after the baseUrlServer as the baseUrl for the hub

private baseUrlHub: string = environment.usebaseUrlHttps ? environment.webApiBaseHubsUrlHttps : environment.webApiBaseHubsUrlHttp;

And then add the presence /message end points

  //signalr end points

  public urlSignalrPresence: string = `${this.baseUrlHub}presence`;

  public urlSignalrMessage: string = `${this.baseUrlHub}message`;

# Setting up Presence

## /core/services

### /signalr/presence-hub.service.ts

Used for login, logout and active users notification

> ng g s core/services/signalr/presenceHub --skip-tests

import { Injectable } from '@angular/core';

import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr';

import { ToastrService } from 'ngx-toastr';

import { BehaviorSubject, take } from 'rxjs';

import { Router } from '@angular/router';

import { HelperService } from '../helper.service';

import { LoggedInUserDto } from '../../models-interfaces/logged-in-user-dto.model';

import { UserDto } from '../../models-interfaces/user-dto.model';

@Injectable({

  providedIn: 'root'

})

export class PresenceHubService {

  private hubConnecton!: HubConnection;

  private onlineUsersSource = new BehaviorSubject<string[]>([]);

  onlineUsers$ = this.onlineUsersSource.asObservable();

  //events from presence hub web api

  private \_keyUserIsOnline = 'UserIsOnline';

  private \_keyUserIsOffline = 'UserIsOffline';

  private \_keyUsersOnline = 'GetOnlineUsers';

  constructor(private helperService: HelperService,

              private toastr: ToastrService,

              private router: Router) { }

  //create Hub Connection

  createHubConnection(user: LoggedInUserDto){

    const url = this.helperService.urlSignalrPresence;

    this.helperService.logIfFrom(url, "PresenceHubService Conn url")

    //build

    this.hubConnecton = new HubConnectionBuilder()

                        .withUrl(url, {

                          accessTokenFactory: () => user.token

                        })

                        .withAutomaticReconnect()

                        .build();

    //start the connection

    this.hubConnecton.start()

                    .catch(e => {

                      this.helperService.logIfFrom(e, "PresenceHubService start Error");

                    });

    //listen for event UserIsOnline, returns userName

    this.hubConnecton.on(this.\_keyUserIsOnline, userName => {

      this.toastr.info(`${userName} has connected!`);

      this.onlineUsers$.pipe(take(1)).subscribe({

        next: (userNames) => {

          const newUserNames = [...userNames, userName];

          this.fireOnlineUsers(newUserNames);

        }

      });

    });

    //listen for event UserIsOffline, returns userName

    this.hubConnecton.on(this.\_keyUserIsOffline, userName => {

      this.toastr.warning(`${userName} has disconnected!`);

      this.onlineUsers$.pipe(take(1)).subscribe({

        next: (userNames) => {

          //filter method cretaed a new array so no need to use the spread operator

          const newUserNames = userNames.filter(x => x !== userName);

          this.fireOnlineUsers(newUserNames);

        }

      });

    });

    //listen for event GetOnlineUsers, returns string[]

    this.hubConnecton.on(this.\_keyUsersOnline, (userNames: string[]) => {

      this.fireOnlineUsers(userNames);

    });

  }

  //stop hub connection

  stopHubConnection(){

    try{

      if(this.hubConnecton){

        this.hubConnecton.stop()

                      .catch(e => {

                        this.helperService.logIfFrom(e, "PresenceHubService stop Error");

                      });

      }

    }

    catch(e){

      this.helperService.logIfFrom(e, "PresenceHubService stop Exception");

    }

  }

  private fireOnlineUsers(userNames: string[]){

    this.onlineUsersSource.next(userNames);

  }

}

### account.service.ts

**constructor**

Inject PresenceHubService

  constructor(private httpClientService: HttpClientService,

              private helperService: HelperService,

              private localStorageService: LocalStorageService,

              private presenseHubService: PresenceHubService) { }

**New methods createHubConnection and stopHubConnection**

  private createHubConnection(user: LoggedInUserDto){

    this.presenseHubService.createHubConnection(user);

  }

  private stopHubConnection(){

    this.presenseHubService.stopHubConnection();

  }

**logout method**

call the stopHubConnection

  logout(){

    //remove the user

    this.localStorageService.removeUser();

    this.fireCurrentUser(null);

    //presence hub

    this.stopHubConnection();

  }

**setAndFireCurrentUser Method**

This method is called by both login and register so call createHubConnection here

  setAndFireCurrentUser(user: LoggedInUserDto){

    //pick roles from the token, for this need to decode the token.

    //keep in mind users may only have single role as string or a string array of roles

    user.roles = [];

    const roles = this.getDecodedToken(user.token)?.role;

    if(roles){

      if(Array.isArray(roles))

        user.roles = roles;

      else

        user.roles.push(roles);

    }

    this.localStorageService.setUser(user);

    this.fireCurrentUser(user);

    //presence hub

    this.createHubConnection(user);

  }

**getAndFireCurrentUser Method**

This is called from the app.componenet.ts so call createHubConnection here as well.

  //this being called from the app.component.ts

  getAndFireCurrentUser(){

    const user: LoggedInUserDto = this.localStorageService.getUser();

    if(!user) {

      this.fireCurrentUser(null);

      return;

    }

    this.fireCurrentUser(user);

    //presence hub

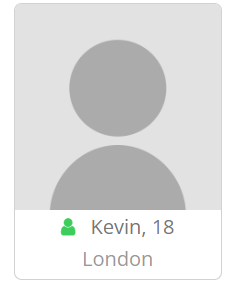
    this.createHubConnection(user);

  }

## Display User Online - Presence

### /site/members/member-card

On the member card blink the icon green when the user is online



**member-card.component.css**

/\* users is online after implementing presenceHub

making the icon blink infinite\*/

@keyframes fa-blink{

    0% {opacity: 1;}

    100% {opacity: 0.4;}

}

.is-online{

    animation: fa-blink 1.5s linear infinite;

    color: rgb(1, 189, 42);

}

**member-card.component.ts**

inject the presenceHubService as public

  constructor(private memberService: MemberService,

              private toastrService: ToastrService,

              public presenceHubService: PresenceHubService){}

**member-card.component.html**

Using async pipe on subject presenceHubService.onlineUsers$, apply the is-online class

    <div class="card-body p-1">

        <h6 class="card-title text-center mb-1">

            <span [class.is-online]="(presenceHubService.onlineUsers$ | async)?.includes(memberIn.userName)">

                <i class="fa fa-user me-2"></i>

            </span>

            {{memberIn.displayName | titlecase}}, {{memberIn.age}}

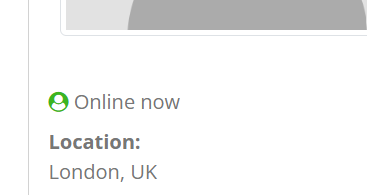
        </h6>

        <p class="card-text text-muted text-center">{{memberIn.city}}</p>

    </div>

### /site/members/member-detail

Will display the online user icon here as well



**member-detail.component.ts**

Inject the presenceHubService as public

  constructor(private memberService: MemberService,

              private activatedRoute: ActivatedRoute,

              private router: Router,

              private toastr: ToastrService,

              private messageService: MessageService,

              public presenceHubService: PresenceHubService){}

**member-detail.component.html**

            <div class="card-body">

                <div class="mb-2" \*ngIf="(presenceHubService.onlineUsers$ | async)?.includes(member.userName)">

                    <i class="fa fa-user-circle text-success"></i> Online now

                </div>

                <div>

                    <strong>Location:</strong>

                    <p>{{member.city}}, {{member.country}}</p>

                </div>

# Setting up Messages

## /core/models-interfaces/signalr

Models for tracking group

### SignalRConnection

> ng g interface core/models-interfaces/signalr/SignalrConnection --type=model

export interface SignalRConnection {

  connectionId: string;

  userName: string;

  userId: number;

}

### SignalRGroup

> ng g interface core/models-interfaces/signalr/SignalrGroup --type=model

import { SignalRConnection } from "./signalr-connection.model";

export interface SignalRGroup {

  groupName: string;

  connections: SignalRConnection[];

}

## /core/services/signalr/messge-hub.service.ts

Used for two way message communication in real time

> ng g s core/services/signalr/messageHub --skip-tests

using System;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

using AutoMapper;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.SignalR;

using MSC.Core.BusinessLogic;

using MSC.Core.DB.Entities.SignalR;

using MSC.Core.Dtos;

using MSC.Core.Extensions;

using MSC.Core.SignalR;

namespace MSC.WebApi.SignalR;

[Authorize]

/// <summary>

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

/// No package to install for SignalR

/// Messages will be sent using the MessageHub and not MessageController.CreateMessage

/// </summary>

public class MessageHub : Hub

{

    private const string \_keyReceiveMessageThread = "ReceiveMessageThread";

    private const string \_keyNewMessage = "NewMessage";

    private const string \_keyUpdatedGroup = "UpdatedGroup";

    //when the recipient is not on the message page, is online then send the new message received event

    private const string \_keyNewMessageReceived = "NewMessageReceived";

    private readonly IMessageBusinessLogic \_msgBl;

    private readonly IUserBusinessLogic \_userBl;

    private readonly ISignalRBusinessLogic \_srBl;

    private readonly PresenceTrackerMemory \_presenceTracker;

    private readonly IHubContext<PresenceHub> \_presenceHub;

    private readonly IMapper \_mapper;

    public MessageHub(IMessageBusinessLogic msgBl,

                    IUserBusinessLogic userBl,

                    ISignalRBusinessLogic srBl,

                    PresenceTrackerMemory presenceTracker,

                    IHubContext<PresenceHub> presenceHub,

                    IMapper mapper)

    {

        \_msgBl = msgBl;

        \_userBl = userBl;

        \_srBl = srBl;

        \_presenceTracker = presenceTracker;

        \_presenceHub = presenceHub;

        \_mapper = mapper;

    }

    /// <summary>

    /// Implement OnConnectedAsync

    /// </summary>

    /// <returns></returns>

    public override async Task OnConnectedAsync()

    {

        var httpContext = Context.GetHttpContext();

        var connectionId = Context.ConnectionId;

        //current user

        var callerUserName = Context.User.GetUserName();

        var callerUserId = Context.User.GetId();

        //will be passing in the other users info via query string

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

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

        //build the group name, create a group of two users.

        var groupName = GetGroupName(callerUserName, otherUserName);

        //add to SignalR groups

        await Groups.AddToGroupAsync(connectionId, groupName);

        //add to database

        var group = await AddToGroup(groupName, connectionId, callerUserName, callerUserId);

        //send the message to the group

        await Clients.Group(groupName).SendAsync(\_keyUpdatedGroup, group);

        //get the message thread from the message business logic just like MessgeController

        var messages = await \_msgBl.GetMessageThread(callerUserId, otherUserId);

        //send the messages to the caller

        //await Clients.Group(groupName).SendAsync(\_keyReceiveMessageThread, messages);

        await Clients.Caller.SendAsync(\_keyReceiveMessageThread, messages);

    }

    /// <summary>

    /// Implmenent OnDisconnectedAsync

    /// </summary>

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

    /// <returns></returns>

    public override async Task OnDisconnectedAsync(Exception exception)

    {

        var group = await RemoveFromMessageGroup(Context.ConnectionId);

        await Clients.Group(group.GroupName).SendAsync(\_keyUpdatedGroup, group);

        //users will be automatically removed from the group

        await base.OnDisconnectedAsync(exception);

    }

    /// <summary>

    /// Moved here from MessageController

    /// </summary>

    /// <param name="msg"></param>

    /// <returns></returns>

    public async Task CreateMessage(MessageCreateDto msg)

    {

        //get the claims

        var claims = Context.User.GetUserClaims();

        if(claims == null || !claims.HasUserName || !claims.HasId || !claims.HasGuid)

            throw new HubException("User issue");

        //check message

        if(msg == null || msg.RecipientId <= 0 || string.IsNullOrWhiteSpace(msg.MessageContent))

            throw new HubException("Message info invalid");

        //check that we have a connection for the recipient. If we have then the recipient is on message page

        bool isRecipientOnMessagePage = false;

        bool markMessageAsRead = false;

        var recipient = await \_userBl.GetUserRawAsync(msg.RecipientId, includePhotos: true);

        if(recipient == null)

            throw new HubException("Recipient not found");

        var groupName = GetGroupName(claims.UserName, recipient.UserName);

        SignalRGroup group = await \_srBl.GetMessageGroup(groupName);

        if(group != null && group.Connections.Any(x => x.UserName == recipient.UserName)){

            isRecipientOnMessagePage = true;

            markMessageAsRead = true;

        }

        //add message , mark the message read if the recipient is in message group with sender

        var result = await \_msgBl.AddMessageWithReadRecipt(msg, claims.Id, markMessageAsRead);

        if(result == null)

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

        if(result.HttpStatusCode != HttpStatusCode.OK)

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

        //the message that got added

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

        //when the recipient is not on the same message page and have connection then notify the recipient

        if(!isRecipientOnMessagePage){

            var connections = await \_presenceTracker.GetConnectionsForUser(recipient.UserName);

            if(connections != null){

                var sender = await \_userBl.GetUserAsync(Context.User.GetId());

                if(sender != null){

                    //to display to the logged in user the sender info

                    //since presenceHub us bing used to send the message, implement this on the presenceHub client

                    await \_presenceHub.Clients.Clients(connections).SendAsync(\_keyNewMessageReceived, sender);

                }

            }

        }

        //send the event for new message created

        await Clients.Group(groupName).SendAsync(\_keyNewMessage, messagedAdded);

    }

    //sort in alphabatical order and build group name

    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}";

    }

    private async Task<SignalRGroup> AddToGroup(string groupName, string connectionId, string callerUserName, int callerUSerId)

    {

        //get the group from the DB and save it

        SignalRGroup group = await \_srBl.GetMessageGroup(groupName);

        if(group == null){

            group = new SignalRGroup(groupName);

            //only saving the group when not found.

            \_srBl.AddGroup(group);

        }

        //create connection

        SignalRConnection connection = new SignalRConnection(connectionId, callerUserName, callerUSerId);

        //add connection to group and call save method

        group.Connections.Add(connection);

        //save

        if(await \_srBl.SaveAllSync())

            return group;

        throw new HubException("Failed to join group");

    }

    private async Task<SignalRGroup> RemoveFromMessageGroup(string connectionId)

    {

        SignalRGroup group = await \_srBl.GetGroupByConnection(connectionId);

        if(group == null)

            throw new HubException("Failed to get group for connection");

        SignalRConnection connection = group.Connections.FirstOrDefault(x => x.ConnectionId == connectionId);

        if (connection == null)

            throw new HubException("Failed to get connection");

        \_srBl.RemoveConnection(connection);

        if(await \_srBl.SaveAllSync()){

            //group.Connections.Remove(connection);

            return group;

        }

        throw new HubException("Failed to remove from group");

    }

}

## /core/services/signalr/presence-hub.service.ts

On the WebApi side the “message create” inside MessageHub is firing “NewMessageReceived” event using the presenceHub. So add that to presenceHubService.

Add the new key at the top

  //this is fired by the messageHub using presenceHub on CreateMessage

  private \_keyNewMessageReceived = 'NewMessageReceived';

Listen to the event

    //fired from messageHub when new message is sent and the user is not on message page, same group as sender

    //this is using the core/strategy/CustomRouteReuseStrategy

    this.hubConnecton.on(this.\_keyNewMessageReceived, (sender: UserDto) => {

      this.toastr.info(`${sender.displayName} has sent you a new message! Click me to see it`)

      .onTap

      .pipe(take(1))

      .subscribe({

        next: () => {

          this.router.navigateByUrl(`members/detail/${sender.guId}/${sender.displayName}?tab=messages`);

        }

      });

    });

## /core/Strategy/customRouteReuseStrategy.ts

    //fired from messageHub when new message is sent and the user is not on message page, same group as sender

    //this is using the core/strategy/CustomRouteReuseStrategy

    this.hubConnecton.on(this.\_keyNewMessageReceived, (sender: UserDto) => {

      this.toastr.info(`${sender.displayName} has sent you a new message! Click me to see it`)

      .onTap

      .pipe(take(1))

      .subscribe({

        next: () => {

          this.router.navigateByUrl(`members/detail/${sender.guId}/${sender.displayName}?tab=messages`);

        }

      });

    });

### app.module.ts

Add CustomeRoouteReuseStrategy

  providers: [

    { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },

    { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true },

    { provide: HTTP\_INTERCEPTORS, useClass: LoadingInterceptor, multi: true },

    { provide: RouteReuseStrategy, useClass: CustomRouteReuseStrategy},

  ],

## /site/members/member-detail

We’ll use the messageHub now to get the message thread. So, we are not loading the messages nor passing the messages to message component.

However, we will start the stop the messageHub connection here.

**member-detail.component.ts**

* Add the param for logged in user
* Inject the account service and messageHubService
* Get the logged I user inside the constructor

  user: LoggedInUserDto = <LoggedInUserDto>{};

  constructor(private memberService: MemberService,

              private activatedRoute: ActivatedRoute,

              private router: Router,

              private toastr: ToastrService,

              private messageService: MessageService,

              public presenceHubService: PresenceHubService,

              private messageHubService: MessageHubService,

              private accountService: AccountService){

    //get the logged in user

    this.accountService.currentLoggedInUser$.pipe(take(1)).subscribe({

      next: (user: LoggedInUserDto | null) => {

        if(user)

          this.user = user;

      }

    });

  }

On ngOnDestroy stop the hub connection

  ngOnDestroy(): void {

    if(this.memberSubscription) this.memberSubscription.unsubscribe();

    if(this.messageThreadSubscription) this.messageThreadSubscription.unsubscribe();

    if(this.memberDataFromRouteSubscription) this.memberDataFromRouteSubscription.unsubscribe();

    if(this.queryParamSubscripton) this.queryParamSubscripton.unsubscribe();

    //also stop the message hub connection. This happening below on activated tab as well

    this.messageHubService.stopHubConnection();

  }

onTabActivated comment the current loadMessages and instead start and stop the messageHub connection

  onTabActivated(data: TabDirective){

    this.activeTab = data;

    //user check and else put in place after message hub implementation

    if(this.activeTab.heading === "Messages" && this.user){

      //using presence hub to get the messages now rather than from the message service

      //note that we are not passing the messages from detail to messages any more

      //only create the hub connection here

      //this.loadMessages();

      this.messageHubService.createHubConnection(this.user, this.member.userName, this.member.id);

    }

    else{

      //also happening in ngOnDestroy

      this.messageHubService.stopHubConnection();

    }

  }

**member-detail.component.html**

Not passing the message to messages component

            <tab heading="Messages" (selectTab)="onTabActivated($event)">

                <!--not passing the messages any more after message hub implementation

                <app-member-messages [memberIn]="member" [messagesIn]="messages"></app-member-messages>

                -->

                <app-member-messages [memberIn]="member"></app-member-messages>

            </tab>

## /site/members/member-messages

**member-messages.component.ts**

This has a lot of changes so full component file placed here

import { Component, Input, OnDestroy, OnInit, ViewChild } from '@angular/core';

import { Subscription } from 'rxjs';

import { TimeagoModule } from 'ngx-timeago';

import { ToastrService } from 'ngx-toastr';

import { CommonModule } from '@angular/common';

import { MessageService } from '../../../core/services/message.service';

import { MessageHubService } from '../../../core/services/signalr/message-hub.service';

import { MessageDto } from '../../../core/models-interfaces/message-dto.model';

import { UserDto } from '../../../core/models-interfaces/user-dto.model';

import { FormsModule, NgForm } from '@angular/forms';

@Component({

  selector: 'app-member-messages',

  standalone: true,

  templateUrl: './member-messages.component.html',

  styleUrls: ['./member-messages.component.css'],

  imports: [CommonModule, TimeagoModule, FormsModule]

})

export class MemberMessagesComponent implements OnInit, OnDestroy {

  //messge template driven form

  @ViewChild('messageForm') msgForm!: NgForm;

  @Input() memberIn!:UserDto;

  //not passing the messages any more after message hub iplementation

  @Input() messagesIn: MessageDto[] = [];

  messageContent: string = '';

  messageSubscription!: Subscription;

  messageHubSubscription!: Subscription;

  constructor(private messageService: MessageService,

              private toastr: ToastrService,

              private messageHubService: MessageHubService){}

  ngOnDestroy(): void {

    if(this.messageSubscription) this.messageSubscription.unsubscribe();

    if(this.messageHubSubscription) this.messageHubSubscription.unsubscribe();

  }

  ngOnInit(): void {

    this.loadMessagesFromMessageHub();

  }

  loadMessagesFromMessageHub(){

    this.messageHubSubscription = this.messageHubService.messageThread$.subscribe({

      next: (messages: MessageDto[]) => {

        //just use the input messages here

        this.messagesIn = messages;

      }

    })

  }

  //not using this any more

  //using the message hub create message methos

  onSubmitMessage(){

    if(!this.memberIn || !this.memberIn.id){

      this.toastr.error('User error', 'Error');

      return;

    }

    this.messageSubscription = this.messageService.createMessage(this.memberIn.id, this.messageContent).subscribe({

      next: (message: MessageDto) => {

        if(!message){

          this.toastr.error("Unable to get back the created message. Refresh page...", "Error");

          return;

        }

        //not using it after the message hub implementation

        this.messagesIn.push(message);

        this.msgForm.reset();

        //this.messageContent = "";

      },

      error: e => {},

      complete: () => {}

    })

  }

  OnSubmitMessageUsingHub(){

    if(!this.memberIn || !this.memberIn.id){

      this.toastr.error('User error', 'Error');

      return;

    }

    //this is returning a promise

    //The newly added message will show via loadMessagesFromMessageHub

    this.messageHubService.createMessage(this.memberIn.id, this.messageContent)

                          .then(() => {

                            this.msgForm.reset();

                          });

  }

}

**member-messages.componenet.html**

on ngsubmit use the new method to add the messages via message hub

    <div class="card-footer">

        <form #messageForm="ngForm" (ngSubmit)="OnSubmitMessageUsingHub()" autocomplete="off">