# Project

* Site-14-Api-Ng-messaging-queryparams-routeresolvers
  + Copied from: Site-13-Api-Ng-like
* For the “MySocialConnect-SPA” issue npm install command to install all packages
  + ng serve : to run the spa

# New & Updates

|  |  |
| --- | --- |
| New | Updates |
| /core/enums/z-message-type.ts | /core/services/helper.service.ts |
| /core/models-interfaces/pagination/page-params.model.ts | /core/services/member.service.ts |
| /core/models-interfaces/message-dto.model.ts | /site/messages |
| /core/models/interfcaes/message-params-dto.model.ts | /site/modules/sahed.module.ts |
| /core/models-interfaces/message-create-dto.model.ts | /site/members/memberDetail |
| /core/services/message.service.ts | /site/members/memberCard |
| /core/services/page.service.ts | app-routing.module.ts |
| /site/members/memberMessages |  |
| /core/resolvers.member-detail.resolver.ts |  |
|  |  |

# Clear Database

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

dotnet ef database drop

dotnet ef database update

New migrations added so check the [web api doc](013%20Project14%20-WebApi-messaging-queryparams-routeresolvers.docx)

# /core/enums

## z-message-type.ts

> ng g enum core/enums/zMessageType

export enum ZMessageType {

    inbox = 0,

    inboxUnread = 1,

    outbox = 2,

}

# /core/models-interfaces

## pagination

### page-params.model.ts

> ng g class core/models-interfaces/pagination/pageParams --type=model --skip-tests

import { HttpParams } from "@angular/common/http";

import { AppConstants } from "../../constants/app-constants";

export class PageParams {

    pageNumber: number = AppConstants.PageNumber;

    pageSize: number = AppConstants.PageSize;

    constructor(){}

    //helper method to build search prams

    getPaginationSearchParams(): HttpParams{

        let params = new HttpParams();

        params = params.append('pageNumber', this.pageNumber.toString());

        params = params.append('pagesize', this.pageSize.toString());

        return params;

    }

}

## message-dto.model.ts

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

export interface MessageDto {

    id: number;

    guid: string;

    senderId: number;

    senderGuid: string,

    senderUsername: string;

    senderPhotoUrl: string;

    receipientId: number;

    receipientGuid: string;

    receipientUsername: string;

    receipientPhotoUrl: string;

    messageContent: string;

    dateMessageRead?: Date;

    dateMessageSent: Date;

}

## message-params-dto.model.ts

> ng g class core/models-interfaces/messageParamsDto --type=model --skip-tests

import { HttpParams } from "@angular/common/http";

import { ZMessageType } from "../enums/z-message-type";

import { PageParams } from "./pagination/page-params.model";

export class MessageParamsDto extends PageParams {

    messageType: ZMessageType = ZMessageType.inboxUnread;

    constructor(){

        super();

    }

    //helper metod to build search params

    getSearchParams() : HttpParams {

        let params = super.getPaginationSearchParams();

        params = params.append('messageType', ZMessageType[this.messageType]);

        return params;

    }

}

## message-create-dto.model.ts

> ng g class core/models-interfaces/messageCreateDto --type=model --skip-tests

export class MessageCreateDto {

    constructor(public recipientId: number, public messageContent: string){}

}

# /core/services

## helper.service.ts

Add the section for messages urls

  //messages

  private urlMessages: string = `${this.baseUrlServer}message`;

  public urlMessageCreate: string = `${this.urlMessages}/create`;

  public urlMessageGetForUser: string = `${this.urlMessages}/user/get/messages`;

  public urlMessageGetThread: string = `${this.urlMessages}/thread/${this.keyId}`;  //recipientId

  public urlMessageDelete: string = `${this.urlMessages}/delete/${this.keyGuid}`; //message guid

## page.service.ts

Move the getPaginatedResult method from member.service.ts to this service so that we can reuse it.

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

import { map } from 'rxjs';

import { HttpParams } from '@angular/common/http';

import { HttpClientService } from './http-client.service';

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

import { PaginatedResult } from '../models-interfaces/pagination/paginated-result.model';

import { AppConstants } from '../constants/app-constants';

@Injectable({

  providedIn: 'root'

})

export class PageService {

  constructor(private httpClientService: HttpClientService, private helperService: HelperService) { }

  getPaginatedResult<T>(url: string, params: HttpParams){

    this.helperService.logIfFrom(url, "PageService getPaginatedResult Request");

    this.helperService.logIf(params);

    const paginatedResult: PaginatedResult<T> = new PaginatedResult<T>();

    return this.httpClientService.getWithFullResponse<T>(url, params).pipe(

      map(response => {

        //result - members list

        if(response.body)

          paginatedResult.result = response.body;

        //pagination info from the header

        const header = response.headers.get(AppConstants.PaginationHeader);

        if(header != null)

          paginatedResult.pagination = JSON.parse(header)

        this.helperService.logIfFrom(paginatedResult, "PageService getPaginatedResult");

        return paginatedResult;

      })

    );

  }

}

## member.service.ts

Remove getPaginatedResult from member.service.ts and use it from page.service.ts

Inject the service in constructor

  constructor(private localStorageService: LocalStorageService,

            private helperService: HelperService,

            private httpClientService: HttpClientService,

            private accountService: AccountService,

            private pageService: PageService)

Update getMembers method to use page service

  getMembers(userParams: UserParamsDto){

    //maake sure that the user is in cache

    const key = this.getCacheKey(userParams);

    this.helperService.logIfFrom(key, "getMembers cachekey");

    const members = this.memberCache.get(key);

    if(members)

      return of(members);

    //add the pagination and search params. Following method gives both pagination and user search params

    //return HttpParams

    const params = userParams.getDefaultMemberSearchParams();

    //url

    const url = this.helperService.urlUsersAll;

    //make the call

    return this.pageService.getPaginatedResult<UserDto[]>(url, params).pipe(

      map(response=> {

        //add the members to the cache

        this.memberCache.set(key, response)

        return response;

      })

    );

  }

Update getLike method to use page service

  getLike(likeParams: LikeParamsDto){

    //add the pagination and filtering params

    const params = likeParams.getLikeSearchParams();

    const url = this.helperService.urlLikeGetUsers;

    this.helperService.logIfFrom(url, "memberService getLike")

    this.helperService.logIf(params);

    //make the call, the result is not full userDto, has some properties

    return this.pageService.getPaginatedResult<Partial<UserDto[]>>(url, params);

  }

## message.service.ts

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

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

import { HttpClientService } from './http-client.service';

import { PageService } from './page.service';

import { MessageParamsDto } from '../models-interfaces/message-params-dto.model';

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

import { Observable } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class MessageService {

  constructor(private httpClientService: HttpClientService,

              private helperService: HelperService,

              private pageService: PageService) { }

  getMessages(search: MessageParamsDto){

    const params = search.getSearchParams();

    const url = this.helperService.urlMessageGetForUser

    return this.pageService.getPaginatedResult<MessageDto[]>(url, params);

  }

  getMessageThread(recipientId: number) : Observable<MessageDto[]>{

    const url = this.helperService.replaceKeyValue(this.helperService.urlMessageGetThread, this.helperService.keyId, recipientId);

    this.helperService.logIfFrom(url, "MessageService getMessageThread");

    return this.httpClientService.get<MessageDto[]>(url);

  }

  createMessage(recipientId: number, messageContent: string){

    const url = this.helperService.urlMessageCreate;

    const data = {recipientId: recipientId, messageContent: messageContent};

    this.helperService.logIfFrom(url, "MessageService createMessage");

    return this.httpClientService.post<MessageDto>(url, data);

  }

  deleteMessage(guid: string){

    const url = this.helperService.replaceKeyValue(this.helperService.urlMessageDelete, this.helperService.keyGuid, guid);

    this.helperService.logIfFrom(url, "MessageService deleteMessage");

    return this.httpClientService.delete(url);

  }

}

# /core/resolvers

## member-detail.resolvers.ts

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

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

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

import { MemberService } from '../services/member.service';

//add the resolver to the app-routing.module, member-detail route

export const memberDetailResolver: ResolveFn<UserDto> = (route, state) => {

  const memberService = inject(MemberService);

  //get the route params

  //this.guidParam = this.activatedRoute.snapshot.paramMap.get('guid');

  //this.nameParam = this.activatedRoute.snapshot.paramMap.get("name");

  /\*

  this.paramSubscription = this.activatedRoute.params.subscribe(params => {

      this.guid = params['guid'];

      this.name = params['name'];

    });

  this.queryParamSubscription = this.route.queryParams.subscribe({

      next: params => {

        const tab: number = +params['tab'];

        const abc: stirng = params['abc']

      },

      error: e => { },

      complete: () => {}

    });

  \*/

  const guid = route.paramMap.get('guid');

  const name = route.paramMap.get('name');

  return memberService.getMemberByGuid(guid!);

};

## app-routing.module.ts

update member-detail route to use the resolver

      //resolver used, property member will get the data so use 'member' in member-detail page

      { path: 'members/detail/:guid/:name', component: MemberDetailComponent, resolve: {member: memberDetailResolver} },

# /site/messages

## messages.component.ts

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

import { ToastrService } from 'ngx-toastr';

import { Subscription } from 'rxjs';

import { ZMessageType } from 'src/app/core/enums/z-message-type';

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

import { MessageParamsDto } from 'src/app/core/models-interfaces/message-params-dto.model';

import { PaginatedResult } from 'src/app/core/models-interfaces/pagination/paginated-result.model';

import { Pagination } from 'src/app/core/models-interfaces/pagination/pagination.model';

import { HelperService } from 'src/app/core/services/helper.service';

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

@Component({

  selector: 'app-messages',

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

  styleUrls: ['./messages.component.css']

})

export class MessagesComponent implements OnInit, OnDestroy {

  isGettingMessage: boolean = false;

  messages!: MessageDto[];

  pagination!: Pagination;

  msgParams: MessageParamsDto; //this has the default for messageType which is inboxUnread

  msgTypeString: string;

  //for use in the componenet

  zMessageType = ZMessageType;

  messageSubscription!: Subscription;

  deleteSubscription!: Subscription;

  fromTo:string = "";

  sentReceived: string = "";

  constructor(private messageService: MessageService,

            private helperService: HelperService,

            private toastr: ToastrService){

    this.msgParams = this.messageService.messageSearchParams;

    this.msgTypeString = ZMessageType[this.msgParams.messageType];

  }

  ngOnDestroy(): void {

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

    if(this.deleteSubscription) this.deleteSubscription.unsubscribe();

  }

  ngOnInit(): void {

    this.setFromToSentReceived();

    this.loadMessages();

  }

  loadMessages(){

    this.pagination = <Pagination>{};

    this.isGettingMessage = true;

    this.messages = [];

    const msgTypeString = ZMessageType[this.msgParams.messageType];

    //if the user has asked for a different type then reset the number

    if(this.msgTypeString !== msgTypeString){

      this.msgTypeString = msgTypeString;

      this.msgParams.pageNumber = 1;

    }

    this.setFromToSentReceived();

    this.messageSubscription = this.messageService.getMessages(this.msgParams).subscribe({

      next: (response: PaginatedResult<MessageDto[]>) => {

        if(response && response.result && response.pagination){

          this.messages = response.result;

          this.pagination = response.pagination;

        }

      },

      error: e => {},

      complete: () => this.isGettingMessage = false

    })

  }

  onPageChange(event: any){

    if(this.msgParams && this.msgParams.pageNumber != event.page){

      this.msgParams.pageNumber = event.page;

      this.loadMessages;

    }

  }

  onDeleteMessage(guid: string){

    this.deleteSubscription = this.messageService.deleteMessage(guid).subscribe({

      next: () => {

        //remove the message from the messages by passing the index and total number of messages to delete which is one in this case

        if(this.messages && this.messages.length > 0){

          this.messages.splice(this.messages.findIndex(x => x.guid === guid), 1);

          this.toastr.success("Message deleted", "Delete");

        }

      },

      error: e => {},

      complete: () => {}

    })

  }

  private setFromToSentReceived(){

    this.fromTo = "From";

    this.sentReceived = "Received";

    if(this.isOutBox()){

      this.fromTo = "To";

      this.sentReceived = "Sent";

    }

  }

  private isOutBox(): boolean{

    return this.msgParams.messageType == ZMessageType.outbox;

  }

  private getMemberPiece(piece: string, message: MessageDto): string{

    let userName = message.senderUsername;

    let photoUrl = message.senderPhotoUrl;

    let guid = message.senderGuid;

    const detailLink = "/members/detail/";

    let url = `${detailLink}/${message.senderGuid}/${message.senderUsername}`;

    if(this.isOutBox()){

      userName = message.recipientUsername;

      photoUrl = message.recipientPhotoUrl;

      guid = message.recipientGuid;

      url = `${detailLink}/${message.recipientGuid}/${message.recipientUsername}`;

    }

    let result = "";

    switch(piece){

      case "username":

        result = userName;

        break;

      case "photourl":

        result = photoUrl;

        break;

      case "guid":

        result = guid;

        break;

      case "detaillink":

        result = url;

        break;

      default:

        this.toastr.error("Unaable to get piece", "Error");

        break;

    }

    return result;

  }

  getMemberUserName(messsage: MessageDto){

    return this.getMemberPiece("username", messsage);

  }

  getMemberPhotoUrl(message: MessageDto){

    return this.getMemberPiece("photourl", message);

  }

  getMemberGuid(message: MessageDto){

    return this.getMemberPiece("guid", message);

  }

  getMemberDetailLink(message: MessageDto){

    return this.getMemberPiece("detaillink", message);

  }

}

## messages.componenet.css

.img-circle{

    max-height: 30px;

}

## messages.component.html

* router link is used on the tr tag.
* This is passing query params member-detail to show messages tab.

            <tr [hidden]="isGettingMessage" \*ngFor="let message of messages"

                routerLink='{{ getMemberDetailLink(message) }}'

                [queryParams]="{tab: 'messages'}"

                title="{{ message.dateMessageRead ? 'View Details' : 'Unread - View Details' }}">

Here is the full html

<div class="mb-4 d-flex">

    <!--redio buttons-->

    <div class="btn-group" name="messageType">

        <button class="btn btn-primary btn-small me-1"

            [ngClass]="{'btn-success': msgTypeString == zMessageType[zMessageType.inboxUnread]}"

            [(ngModel)]="msgParams.messageType"

            btnRadio="{{zMessageType.inboxUnread}}"

            (click)="loadMessages()"

            >

            <i class="fa fa-envelope"></i> Unread

        </button>

        <button class="btn btn-primary btn-small me-1"

            [ngClass]="{'btn-success': msgTypeString == zMessageType[zMessageType.inbox]}"

            [(ngModel)]="msgParams.messageType"

            btnRadio="{{zMessageType.inbox}}"

            (click)="loadMessages()"

            >

            <i class="fa fa-envelope-open"></i> Inbox

        </button>

        <button class="btn btn-primary btn-small me-1"

            [ngClass]="{'btn-success': msgTypeString == zMessageType[zMessageType.outbox]}"

            [(ngModel)]="msgParams.messageType"

            btnRadio="{{zMessageType.outbox}}"

            (click)="loadMessages()"

            >

            <i class="fa fa-regular fa-paper-plane"></i> Outbox

        </button>

    </div>

</div>

<!--when no messages are found then displaying conditional messages-->

<div class="row" \*ngIf="!isGettingMessage && messages?.length === 0" [ngSwitch]="msgTypeString">

    <h3 \*ngSwitchCase="zMessageType[zMessageType.inbox]">Inbox no messages</h3>

    <h3 \*ngSwitchCase="zMessageType[zMessageType.inboxUnread]">No unread messages</h3>

    <h3 \*ngSwitchDefault>Nothing in outbox</h3>

</div>

<!--displaying messages-->

<div class="row" \*ngIf="messages!.length > 0">

    <table class="table table-hover table-striped table-bordered" style="cursor: pointer">

        <thead class="thead-dark">

            <tr>

                <th style="width: 60%">Message</th>

                <th style="width: 15%">{{ fromTo }}</th>

                <th style="width: 15%">{{ sentReceived }}</th>

                <th style="width: 10%"></th>

            </tr>

        </thead>

        <tbody class="align-middle">

            <tr [hidden]="isGettingMessage" \*ngFor="let message of messages"

                routerLink='{{ getMemberDetailLink(message) }}'

                [queryParams]="{tab: 'messages'}"

                title="{{ message.dateMessageRead ? 'View Details' : 'Unread - View Details' }}">

                <td class="message-content">

                    <span class="" \*ngIf="!message.dateMessageRead"><i class="fa fa-envelope-o" aria-hidden="true"></i>&nbsp;</span>

                    <span class="" \*ngIf="message.dateMessageRead"><i class="fa fa-envelope-open-o" aria-hidden="true"></i>&nbsp;</span>

                    {{message.messageContent}}

                </td>

                <td>

                    <div>

                        <img class="img-circle rounded-circle me-2" src="{{ getMemberPhotoUrl(message)  || './assets/user.png' }}" alt="{{ getMemberUserName(message) }}">

                        <strong>{{ getMemberUserName(message) | titlecase }}</strong>

                    </div>

                </td>

                <td>{{ message.dateMessageSent  + 'Z' | timeago }}</td>

                <td class="text-center">

                    <!--there is link on the row. add second link to stop propagation -->

                    <button type="button" class="btn btn-danger btn-sm" (click)="$event.stopPropagation()" (click)="onDeleteMessage(message.guid)" title="Delete">Delete</button>

                </td>

            </tr>

        </tbody>

    </table>

</div>

<div class="d-flex justify-content-center" \*ngIf="pagination && messages!.length > 0">

    <pagination

        [boundaryLinks]="true"

        [totalItems]="pagination.totalItems"

        [itemsPerPage]="pagination.itemsPerPage"

        [(ngModel)]="pagination.currentPage"

        (pageChanged)="onPageChange($event)"

        previousText="&lsaquo;"

        nextText="&rsaquo;"

        firstText="&laquo;"

        lastText="&raquo;"

        >

    </pagination>

</div>

# /site/members/memberCard

## member-card.component.html

put a link on envelope button to take the users to member-detail and pass tab=Members as query string.

* router link is used on the button tag.
* This is passing query params member-detail to show messages tab.

            <li class="list-inline-item">

                <button class="btn btn-primary"

                    routerLink='/members/detail/{{memberIn.guId}}/{{memberIn.displayName}}'

                    [queryParams]="{tab: 'messages'}"><i class="fa fa-envelope"></i></button>

            </li>

# /site/members/memberDetail

Messages tab will be using the component [member-messages](#_/site/members/memberMessages) created below.

## member-detail.component.ts

member is being passed in via the [route resolver](#_/core/resolvers), created above.

Full component class is here now

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

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

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

import { TabDirective, TabsModule, TabsetComponent } from 'ngx-bootstrap/tabs';

import { GalleryItem, GalleryModule, ImageItem } from 'ng-gallery';

import { ToastrService } from 'ngx-toastr';

import { Subscription } from 'rxjs';

import { TimeagoModule } from 'ngx-timeago';

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

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

import { MemberService } from '../../../core/services/member.service';

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

import { MemberMessagesComponent } from '../member-messages/member-messages.component';

@Component({

  selector: 'app-member-detail',

  standalone: true,

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

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

  imports: [CommonModule, TabsModule, GalleryModule, TimeagoModule, MemberMessagesComponent]

})

export class MemberDetailComponent implements OnInit, OnDestroy {

  //adding static since now we are using route resolver to get memebr data

  //also removed the check for members in the html

  @ViewChild('memberTabs', {static: true}) memberTabs?: TabsetComponent;

  activeTab?: TabDirective;

  messages: MessageDto[] = [];

  //after resolver so that we don't check ? in component. Initializing with empty object

  //can also do member:USerDto = {} as UserDto

  member: UserDto = <UserDto>{};

  guidParam: string | null = "";

  nameParam: string | null = "";

  images: GalleryItem[] = [];

  memberSubscription!: Subscription;

  messageThreadSubscription!: Subscription;

  //after resolver

  memberDataFromRouteSubscription!: Subscription;

  queryParamSubscripton!: Subscription;

  constructor(private memberService: MemberService,

              private activatedRoute: ActivatedRoute,

              private router: Router,

              private toastr: ToastrService,

              private messageService: MessageService){}

  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();

  }

  ngOnInit(): void {

    //old way is getting the params from the route for guid and name and then call the loadMembers

    //this.oldWay();

    //new way is using the route resolver

    this.newRouteResolver();

  }

  private oldWay(){

    this.guidParam = this.activatedRoute.snapshot.paramMap.get('guid');

    this.nameParam = this.activatedRoute.snapshot.paramMap.get("name");

    if(!this.guidParam)

      this.naviagteToMember("GUID not received on the page", "User Error");

    this.loadMember();

  }

  private newRouteResolver(){

    //we need to have the member before our component is constructed

    //for this will use the route resolver

    //check core/resolvers/member-detail and also the member-detail route in app-routing.module

    //pick member data

    this.memberDataFromRouteSubscription = this.activatedRoute.data.subscribe({

      next: data => {

        this.member = data['member'];

        if(!this.member)

          this.naviagteToMember("No member found", "Member Error");

      }

    });

    //switch tab if tab is available

    this.queryParamSubscripton = this.activatedRoute.queryParams.subscribe({

      next: params => {

        //const tab:number = +params['tab'];

        const tab: string = params['tab'];

        if(tab){

          switch (tab.toLowerCase()){

            case 'messages':

              this.onSelectTab('Messages');

              break;

          }

        }

      },

      error: e => {},

      complete: () => {}

    });

    //images

    this.getImages();

  }

  loadMember(){

    this.memberSubscription = this.memberService.getMemberByGuid(this.guidParam!).subscribe({

      next: (member: UserDto) => {

        if(this.nameParam?.toLowerCase() !== member.displayName.toLowerCase())

          this.naviagteToMember("Mismatched user", "User Error");

        this.member = member;

        this.getImages();

      },

      error: e => {},

      complete: () => {}

    })

  }

  getImages(){

    if(!this.member || !this.member.photos || this.member.photos.length <= 0) return;

    for(const photo of this.member.photos){

      this.images.push(new ImageItem({src: photo.url, thumb: photo.url} ))

    }

  }

  naviagteToMember(error: string, errorTitle: string){

    if(error)

      this.toastr.error(error, errorTitle);

    this.router.navigate(["/members"]);

  }

  onSelectTab(tabHeading: string){

    if(!this.memberTabs) return;

    this.memberTabs.tabs.find(x => x.heading === tabHeading)!.active = true;

  }

  onTabActivated(data: TabDirective){

    this.activeTab = data;

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

      this.loadMessages();

    }

  }

  loadMessages(){

    if(!this.member){

      this.toastr.error("No user", "Error");

      return;

    }

    this.messageThreadSubscription = this.messageService.getMessageThread(this.member.id).subscribe({

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

        this.messages = messages;

      },

      error: e => {},

      complete: () => {}

    });

  }

}

## member-detail.component.html

### Initial

* on the tab set add template reference variable #memberTabs
* inside header messages, added the new app-member-massages
* for each tab added (selectTab)="onTabActivated($event)"

        <tabset class="member-tabset" #memberTabs>

            <tab heading="About {{member.displayName | titlecase }}" (selectTab)="onTabActivated($event)">

                <h4>Description</h4>

                <p>{{member.introduction}}</p>

                <h4>Looking for</h4>

                <p>{{member.lookingFor}}</p>

            </tab>

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

                <h4>Interests</h4>

                <p>{{member.interests}}</p>

            </tab>

            <tab heading="Photos" #photoTab="tab" (selectTab)="onTabActivated($event)"> <!--a template reference variable-->

                <gallery \*ngIf="photoTab.active" [items]="images" class="gallery"></gallery>

            </tab>

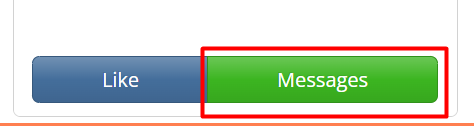
            <tab heading="Messages" (selectTab)="onTabActivated($event)">

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

            </tab>

        </tabset>

On the left hand side we have a Messages button. Call the method to switch the tab with button click.



                <div class="btn-group d-flex">

                    <button class="btn btn-primary">Like</button>

                    <button class="btn btn-success" (click)="onSelectTab('Messages')">Messages</button>

                </div>

### Final

* We are using route resolver and to get data
* We may be receiving the tab so need to switch tab
* So conditional check do not work with switching tabs as the tabs are not available. Remove conditional

<div class="row"> <!--removed \*ngIf="member" after resolver-->

    <div class="col-4">

        <!--left start-->

        <div class="card">

            <img src="{{member.photoUrl || './assets/user.png'}}" alt="{{member.displayName | titlecase}}"

                class="card-img-top img-thumbnail">

            <div class="card-body">

                <div>

                    <strong>Location:</strong>

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

                </div>

                <div>

                    <strong>Age:</strong>

                    <p>{{member.age}}</p>

                </div>

                <div>

                    <strong>Last active:</strong>

                    <!--add the 'Z' to the time, will change later-->

                    <p>{{member.lastActive + 'Z' | timeago}}</p>

                </div>

                <div>

                    <strong>Member since:</strong>

                    <p>{{member.createdOn | date: 'dd MMM yyyy'}}</p>

                </div>

            </div>

            <div class="card-footer">

                <div class="btn-group d-flex">

                    <button class="btn btn-primary">Like</button>

                    <button class="btn btn-success" (click)="onSelectTab('Messages')">Messages</button>

                </div>

            </div>

        </div>

         <!--left end-->

    </div>

    <div class="col-8">

        <!--right start-->

        <tabset class="member-tabset" #memberTabs>

            <tab heading="About {{member.displayName | titlecase }}" (selectTab)="onTabActivated($event)">

                <h4>Description</h4>

                <p>{{member.introduction}}</p>

                <h4>Looking for</h4>

                <p>{{member.lookingFor}}</p>

            </tab>

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

                <h4>Interests</h4>

                <p>{{member.interests}}</p>

            </tab>

            <tab heading="Photos" #photoTab="tab" (selectTab)="onTabActivated($event)"> <!--a template reference variable-->

                <gallery \*ngIf="photoTab.active" [items]="images" class="gallery"></gallery>

            </tab>

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

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

            </tab>

        </tabset>

        <!--right end-->

    </div>

</div>

# /site/members/memberMessages

## Creating

> ng g c site/members/memberMessages --skip-tests

Make it

* make it stand alone component since it will be used inside member-detail which is stand alone
* also remove the declaration from shared module or app module

@Component({

  selector: 'app-member-messages',

  standalone: true,

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

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

})

### /site/modules/shared.module.ts

It will also addd the component to shared module

## member-messages.component.css

.card{

    border: none;

}

.chat{

    list-style: none;

    margin: 0;

    padding: 0;

}

.chat li{

    margin-bottom: 10px;

    padding-bottom: 10px;

}

.chat li:not(:last-child){

    border-bottom: 1px dotted #B3A9A9;

}

.rounded-circle{

    max-height: 50px;

}

## member-messages.component.ts

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 { 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;

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

  messageContent: string = '';

  messageSubscription!: Subscription;

  constructor(private messageService: MessageService, private toastr: ToastrService){}

  ngOnDestroy(): void {

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

  }

  ngOnInit(): void {

  }

  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;

        }

        this.messagesIn.push(message);

        this.msgForm.reset();

        //this.messageContent = "";

      },

      error: e => {},

      complete: () => {}

    })

  }

}

## member-messages.component.html

<div class="card">

    <div class="card-body">

        <div \*ngIf="messagesIn.length === 0">No messages yet... say hi by using the message box below</div>

        <ul \*ngIf="messagesIn.length > 0" class="chat">

            <li \*ngFor="let message of messagesIn">

                <div>

                    <span class="chat-img float-end">

                        <img class="rounded-circle" src="{{message.senderPhotoUrl || './assets/user.png'}}" alt="image of user">

                    </span>

                    <div class="chat-body">

                        <div class="header">

                            <small class="text-muted">

                                <span class="fa fa-clock-o">&nbsp;{{ message.dateMessageSent  + 'Z' | timeago }}</span>

                                <!--message not read where user is not the sender-->

                                <span class="text-danger" \*ngIf="!message.dateMessageRead && message.senderId !== memberIn.id">

                                (unread)

                                </span>

                                <!--message read where user is not the sender-->

                                <span class="text-success" \*ngIf="message.dateMessageRead && message.senderId !== memberIn.id">

                                    (read {{ message.dateMessageRead  + 'Z' | timeago }})

                                </span>

                            </small>

                        </div>

                        <p>{{ message.messageContent }}</p>

                    </div>

                </div>

            </li>

        </ul>

    </div>

    <div class="card-footer">

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

            <div class="input-group">

                <input type="text" class="form-control input-sm" placeholder="Send a private message"

                    name="messageContent" [(ngModel)]="messageContent" required>

                <div class="input-group-append">

                    <button class="btn btn-primary ms-2 me-1" type="submit" [disabled]="!messageForm.valid">Send</button>

                    <button class="btn btn-info" type="reset" [disabled]="!messageContent || messageContent === ''">Clear</button>

                </div>

            </div>

        </form>

    </div>

</div>