# Project

* Site-12-Api-Ng-paging-sorting-filtering
  + Copied from: Site-11-Api-Ng-reactiveforms-reusable-input-validators
* For the “MySocialConnect-SPA” issue npm install command to install all packages
  + ng serve : to run the spa

# New & Updates

|  |  |
| --- | --- |
| New | Updates |
| /core/models-interfaces/pagination/pagination.model.ts | /core/constants/app-constants.ts |
| /core/models-interfaces/pagination/pagination-result.model.ts | /core/services/member.service.ts |
| /core/models-interfaces/user-params-dto.model.ts | /core/services/http-client.service.ts |
| /core/enums/z-member-get-by.ts | /core/modules/shared.module.ts |
|  | /site/members/member-card |
|  | /site/members/member-detail |
|  | /site/members/member-edit |
|  |  |
|  |  |
|  |  |

# Clear Database

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

dotnet ef database drop

dotnet ef database update

# ngx-timeago

<https://www.npmjs.com/package/ngx-timeago>

> npm install ngx-timeago

And add it to the shared module

Impot it

import { TimeagoModule } from 'ngx-timeago';

and add it to the imports and exports array as well

  imports: [

    CommonModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    BsDropdownModule.forRoot(),

    TabsModule.forRoot(),

    ToastrModule.forRoot({ positionClass: 'toast-bottom-right'}),

    NgxSpinnerModule.forRoot({type: 'line-scale-party'}),

    FileUploadModule,

    BsDatepickerModule.forRoot(),

    PaginationModule.forRoot(),

    ButtonsModule.forRoot(),

    TimeagoModule.forRoot(),

  ],

  exports: [

    AppRoutingModule,

    BrowserAnimationsModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    BsDropdownModule,

    TabsModule,

    ToastrModule,

    NgxSpinnerModule,

    FileUploadModule,

    BsDatepickerModule,

    PaginationModule,

    ButtonsModule,

    TimeagoModule,

  ]

# /core/modules

## shared.module.ts

Also, import it

import { PaginationModule } from 'ngx-bootstrap/pagination';

import { ButtonsModule } from 'ngx-bootstrap/buttons';

Add pagination module to the imports and exports array.

  imports: [

    CommonModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    BsDropdownModule.forRoot(),

    TabsModule.forRoot(),

    ToastrModule.forRoot({ positionClass: 'toast-bottom-right'}),

    NgxSpinnerModule.forRoot({type: 'line-scale-party'}),

    FileUploadModule,

    BsDatepickerModule.forRoot(),

    PaginationModule.forRoot(),

    ButtonsModule.forRoot(),

  ],

  exports: [

    AppRoutingModule,

    BrowserAnimationsModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    BsDropdownModule,

    TabsModule,

    ToastrModule,

    NgxSpinnerModule,

    FileUploadModule,

    BsDatepickerModule,

    PaginationModule,

    ButtonsModule,

  ]

# /core/constants

## app-constants.ts

Add the new constants

export class AppConstants {

    public static readonly Bearer = 'Bearer ';

    public static readonly Female = "female";

    public static readonly Male = "male";

    public static readonly membersOrderByLastActive = 'lastActive';

    public static readonly membersOrderByCreated = 'created';

    public static readonly MinAge: number = 18;

    public static readonly MaxAge: number = 99;

    public static readonly PageNumber: number = 1;

    public static readonly PageSize: number = 5;

    //headers

    public static readonly PaginationHeader = 'pagination';

}

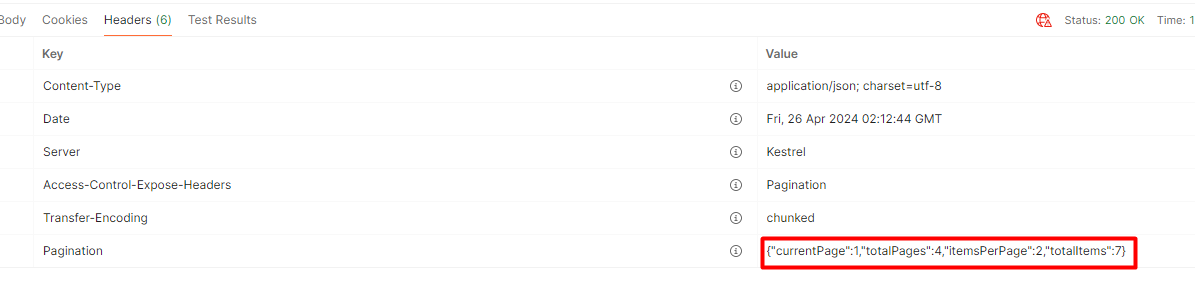
# /core/models-interfaces

## pagination

### pagination.model.ts

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

The properties should be exactly as we have seen in the postman



export class Pagination {

    constructor(public currentPage: number,

                public totalPages: number,

                public itemsPerPage: number,

                public totalItems: number){}

}

### pagination-result.model.ts

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

import { Pagination } from "./pagination.model";

export class PaginatedResult<T> {

    result?: T; //it can be optional. could be undefined as well with !

    pagination?: Pagination //it can be optional. could be undefined as well with !

}

## user-params-dto.model.ts

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

import { LoggedInUserDto } from "./logged-in-user-dto.model";

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

export class UserParamsDto {

    gender: string;

    minAge: number = AppConstants.MinAge;

    maxAge: number = AppConstants.MaxAge;

    pageNumber: number = AppConstants.PageNumber;

    pageSize: number = AppConstants.PageSize;

    orderBy: string = AppConstants.membersOrderByLastActive;

    displayName: string;

    constructor(user: LoggedInUserDto) {

        //when logged in user is male then get the female members otherwise male members

        this.gender = user.gender === AppConstants.Female ? AppConstants.Male : AppConstants.Female;

        this.displayName = user.displayName;

    }

    //helper metod to build search params

    getPaginationSearchParams() : HttpParams {

        let params = new HttpParams();

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

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

        return params;

    }

    //helper metod to build search params

    getMemberSearchParams() : HttpParams {

        let params = this.getPaginationSearchParams();

        params = params.append('gender', this.gender);

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

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

        params = params.append('orderBy', this.orderBy);

        return params;

    }

}

# /core/enums

## z-member-get-by.ts

> ng g enum core/enums/zMemberGetBy

export enum ZMemberGetBy {

    id = 1,

    userName = 2,

    guid = 3,

}

# /core/services

## http-client.service.ts

Add a new method to get the full response for the get. This is needed for looking at the headers like pagination

### Imports

import { HttpClient, HttpResponse } from '@angular/common/http';

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

import { Observable, retry } from 'rxjs';

### getwithFullResposne

   /\*\*

   \* A GET method

   \* @param url api url

   \* @param params pass empty meaning do not pass when not used, will cover stuff like ?x=1&y=2,

   \*        instead use HttpParams

   \*        let params = new HttpParams();

   \*        params = params.append('pageNumber', page.toString());

   \*        params = params.append('pageSize', itemsPerPage.toString());

   \* @returns rreturns the full HttpResponse

   \*/

  getWithFullResponse<TReturn>(url: string, params = {}) : Observable<HttpResponse<TReturn>> {

    return this.httpClient

      .get<TReturn>(url, { observe: 'response', params })

      .pipe(retry(this.retries));

  }

## member.service.ts

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

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

import { Observable, map, of, take } from 'rxjs';

import { LocalStorageService } from './local-storage.service';

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

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

import { AccountService } from './account.service';

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

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

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

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

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

import { ZMemberGetBy } from '../enums/z-member-get-by';

@Injectable({

  providedIn: 'root'

})

export class MemberService {

  //to pass the auth token to the api, later will use interceptor

  private httpOptions;

  //state mangement. We dont want to pull the user info every time

  private members: UserDto[] = []; //not using it any more due to pagination, keeping it but made it private

  //properties after pagination

  //state management- use javascript object map

  private memberCache = new Map();

  private userParams!: UserParamsDto;

  private logedInUser!: LoggedInUserDto;

  constructor(private localStorageService: LocalStorageService,

            private helperService: HelperService,

            private httpClientService: HttpClientService,

            private accountService: AccountService) {

    //to pass the auth token to the api, later will use interceptor

    this.httpOptions = {

      headers: new HttpHeaders({

        Authorization: AppConstants.Bearer + this.localStorageService.getUserToken

      })

    };

    //set the logged in user and default params

    this.setLoggedInUserAndDefaultParams()

  }

  private setLoggedInUserAndDefaultParams(){

    //get the logged in use. no need to unsubscribe since using take(1)

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

      next: user => {

        if(user){

          this.logedInUser = user;

          this.userParamsSet();

        }

      },

      error: e => { },

      complete: () => { }

    })

  }

  memberCacheClear(){

    if(!this.memberCache) return;

    this.memberCache.clear();

  }

  userParamsSet(){

    if(!this.logedInUser) return;

    this.userParams = new UserParamsDto(this.logedInUser);

  }

  userParamsUpdate(userParams: UserParamsDto) {

    this.userParams = userParams;

  }

  userParamsGet(): UserParamsDto{

    return this.userParams;

  }

  private getCacheKey(userParams: UserParamsDto) : string{

    const key = Object.values(userParams).join('-');

    return key;

  }

  /\*\*

   \* A GET users method

   \* @returns Observable userDto[]

   \*/

  /\*

  //not using this any more due to pagination

  getMembers(): Observable<UserDto[]>{

    //state

    if(this.members.length > 0){

      return of(this.members);

    }

    const url = this.helperService.urlUsersAll;

    this.helperService.logIfFrom(url, "memberService getMembers");

    //return this.httpClientService.get<UserDto[]>(url, this.httpOptions)

    return this.httpClientService.get<UserDto[]>(url, this.httpOptions) //jwt interceptor

    .pipe(

      map((members: UserDto[]) => {

        this.members = members;

        return members;

      })

    );

  }

  \*/

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

      map(response=> {

        //add the members to the cache

        this.memberCache.set(key, response)

        return response;

      })

    );

  }

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

    this.helperService.logIfFrom(url, "MemberService 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 infor from the header

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

        if(header != null)

          paginatedResult.pagination = JSON.parse(header)

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

        return paginatedResult;

      })

    );

  }

  /\*\*

   \* A GET user by guid method

   \* @returns Observable userDto

   \*/

  /\*

  //using the new getMemberByGuid method

  getMemberByGuid(guid: string): Observable<UserDto>{

    //state

    if(this.members.length > 0){

      const member = this.members.find(x => x.guId === guid);

      if(member){

        return of(member);

      }

    }

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

    this.helperService.logIfFrom(url, "memberService getMemberByGuid");

    //return this.httpClientService.get<UserDto>(url, this.httpOptions)

    return this.httpClientService.get<UserDto>(url, this.httpOptions) //jwt interceptor

      .pipe(

        map((member: UserDto) => {

          return member;

        })

      );

  }

  \*/

  getMemberByGuid(guid: string): Observable<UserDto>{

    //first try getting the use from the cache

    const member = this.getMemberCache(guid, ZMemberGetBy.guid);

    if(member)

      return of(member);

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

    this.helperService.logIfFrom(url, "memberService getMemberByGuid");

    return this.httpClientService.get<UserDto>(url);

  }

  private getMemberFromArray(key: any, by: ZMemberGetBy): UserDto | undefined{

    let member: UserDto | undefined;

    if(!by || !key || this.members.length <= 0) return member;

    switch(by){

      case ZMemberGetBy.id:

        member = this.members.find(x => x.id === +key);

        break;

      case ZMemberGetBy.userName:

        member = this.members.find(x => x.userName === key);

        break;

      case ZMemberGetBy.guid:

        member = this.members.find(x => x.guId === key);

        break;

    }

    return member;

  }

  private getMemberCache(key: any, by: ZMemberGetBy): UserDto | undefined{

    let member: UserDto | undefined;

    if (!key || !by) return member;

    //using map and reduce to create a single array out of key value array pairs

    const members = [...this.memberCache.values()].reduce((arr, elem) => arr.concat(elem.result), []);

    this.helperService.logIfFrom(members, "MemberService getMemberCache")

    if (members.length <= 0) return member;

    switch (by) {

      case ZMemberGetBy.id:

        member =  members.find((x: UserDto) => x.id === +key);

        break;

      case ZMemberGetBy.userName:

        member =  members.find((x: UserDto) => x.userName === key);

        break;

      case ZMemberGetBy.guid:

        member =  members.find((x: UserDto) => x.guId === key);

        break;

    }

    return member;

  }

  updateMember(user: UserDto){

    const url = this.helperService.urlUserUpdate;

    this.helperService.logIfFrom(url, "memberService updateMember");

    return this.httpClientService.put(url, user).pipe(

      map(() => {

        const index = this.members.indexOf(user);

        //this.members[index] = user;

        this.members[index] = {...this.members[index], ...user};

      })

    );

  }

  setMainPhoto(photoId: number){

    const url = this.helperService.replaceKeyValue(this.helperService.urlUserPhotoSetMain, this.helperService.keyPhotoId, photoId, "");

    this.helperService.logIfFrom(url, "memberService setMainPhoto");

    return this.httpClientService.put(url, {});

  }

  deletePhoto(photoId: number){

    const url = this.helperService.replaceKeyValue(this.helperService.urlUserPhotoDelete, this.helperService.keyPhotoId, photoId, "");

    this.helperService.logIfFrom(url, "memberService deletePhoto");

    return this.httpClientService.delete(url);

  }

}

# /site/member-list

## member-list.component.ts

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

import { Observable, Subscription, take } from 'rxjs';

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

import { AccountService } from 'src/app/core/services/account.service';

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

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

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

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

import { AppConstants } from 'src/app/core/constants/app-constants';

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

@Component({

  selector: 'app-member-list',

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

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

})

export class MemberListComponent implements OnInit, OnDestroy {

  members: UserDto[] = [];

  membersAsync$: Observable<UserDto[]> | undefined;

  //pagination and filtering

  pagination!: Pagination;

  userParams!: UserParamsDto;

  user!: LoggedInUserDto;

  membersSubscription!: Subscription;

  //filtering

  genderList = [

    { value: AppConstants.Male, display: 'Males' },

    { value: AppConstants.Female, display: 'Females' }

  ];

  constructor(private memberService: MemberService, private accountService: AccountService){

    this.userParams = this.memberService.userParamsGet();

  }

  ngOnDestroy(): void {

    if(this.membersSubscription) this.membersSubscription.unsubscribe();

  }

  ngOnInit(): void {

    //not using load members any more but using membersAsync$

    this.loadMembers();

    //this.membersAsync$ = this.memberService.getMembers();

  }

  loadMembers(){

    if(!this.userParams) return;

    //reset user params

    this.memberService.userParamsUpdate(this.userParams);

    this.membersSubscription = this.memberService.getMembers(this.userParams).subscribe({

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

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

          this.members = response.result;

          this.pagination = response.pagination;

        }

      },

      error: e => {},

      complete: () => {}

    })

  }

  onPageChanged(event: any){

    if(this.userParams.pageNumber !== event.page){

      this.userParams.pageNumber = event.page;

      this.loadMembers();

    }

  }

  onResetFilters(){

    this.memberService.userParamsSet();

    this.userParams = this.memberService.userParamsGet();

    this.memberService.memberCacheClear();

    this.loadMembers();

  }

  onGenderSelect() {

    //reset the page number to 1 since the page the user is on my not exist when the gender is changed

    this.userParams.pageNumber = 1;

  }

}

## member-list.component.html

<div class="text-center mt-3" \*ngIf="pagination && pagination.totalItems">

    <h2>Your matches - {{pagination.totalItems}} found</h2>

</div>

<div class="container mt-3" \*ngIf="userParams">

    <!--d-flex will create inline items-->

    <form #filterForm="ngForm" class="d-flex mb-3" (ngSubmit)="loadMembers()" autocomplete="off">

        <div class="d-flex mx-2">

            <label for="minAge" class="col-form-label">Age from: </label>

            <input type="number" class="form-cotrol ms-1" style="width: 70px !important" name="minAge" [(ngModel)]="userParams.minAge">

        </div>

        <div class="d-flex mx-2">

            <label for="maxAge" class="col-form-label">Age to: </label>

            <input type="number" class="form-cotrol ms-1" style="width: 70px !important" name="maxAge" [(ngModel)]="userParams.maxAge">

        </div>

        <div class="d-flex px-2">

            <label for="gender" class="col-form-label me-2">Show: </label>

            <!--on gender change reste the pageMumber to 1 as the page may not exists when gender is changed-->

            <select name="gender" style="width: 130px !important;" class="form-select ms-1" [(ngModel)]="userParams.gender" (change)="onGenderSelect()">

                <option \*ngFor="let gender of genderList" [value]="gender.value">

                    {{gender.display}}

                </option>

            </select>

        </div>

        <button class="btn btn-small btn-primary ms-1" type="submit">Apply Filters</button>

        <button class="btn btn-small btn-info ms-1" type="button" (click)="onResetFilters()">Reset Filters</button>

        <div class="col">

            <div class="btn-group float-end">

                <button type="button" class="btn btn-small btn-primary" name="orderBy" [(ngModel)]="userParams.orderBy" btnRadio="lastActive" (click)="loadMembers()">Last Active</button>

                <button type="button" class="btn btn-small btn-info" name="orderBy" [(ngModel)]="userParams.orderBy" btnRadio="created" (click)="loadMembers()">Created</button>

            </div>

        </div>

    </form>

</div>

<div class="row mt-4">

    <div class="col-2" \*ngFor="let member of members">

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

    </div>

    <!--

    <div class="col-2" \*ngFor="let member of membersAsync$ | async">

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

    </div>

    -->

</div>

<div class="d-flex justify-content-center" \*ngIf="pagination">

    <!--[maxSize]="10"-->

    <pagination

            [boundaryLinks]="true"

            [totalItems]="pagination.totalItems"

            [itemsPerPage]="pagination.itemsPerPage"

            [(ngModel)]="pagination.currentPage"

            (pageChanged)="onPageChanged($event)"

            previousText="&lsaquo;"

            nextText="&rsaquo;"

            firstText="&laquo;"

            lastText="&raquo;">

</pagination>

</div>

# ShowAge

## /site/members/member-card

### member-card.component.html

show the age here as well

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

# FormatDates

## /site/members/member-detail

### member-detail.componenet.ts

Since this is a stand alone component will need to import the time ago module here before using it

  imports: [CommonModule, TabsModule, GalleryModule, TimeagoModule]

### member-detail.componenet.html

                <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>

## /site/members/member-edit

### member-edit.component.html

                <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>