# Project

* Site-12-Api-Ng-paging-sorting-filtering
  + Copied from: Site-11-Api-Ng-reactiveforms-reusable-input-validators
* For the “MySocialConnect-API”
  + dotnet restore
  + dotnet build
  + Go to project : MSC.WebApi
    - dotnet build : to build
    - dotnet run : to run the api

# New Resources

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/Dtos/Pagination/PagedList.cs |  |
| MSC.Core/Dtos/Pagination/PaginationHeader.cs |  |
| MSC.Core/Dtos/Pagination/PaginationParams.cs |  |
| MSC.Core/Dtos/UsersSearchParamsDto.cs |  |
| MSC.Core/Extensions/JsonExtensions.cs |  |
| MSC.Core/Extensions/HttpExtensions.cs |  |
| MSC.Core/Constants/HeaderNameConstants |  |
| MSC.Core/Constants/DataConstants.cs |  |
| MSC.Core/ActionFilters/LogUserActivityFilter.cs |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/Repositories/UserRepository | USersController |
| MSC.Core/Extensions/DateTimeExtensions.cs |  |
| MSc.Core/Extensions/AppServiceExtensions.cs |  |
|  |  |
|  |  |
|  |  |

# Clear Database

Clear the users

dotnet ef database drop

dotnet ef database update

# MSC.Core/Constants

## HeaderNameConstants.cs

namespace MSC.Core.Constants;

public class HeaderNameConstants

{

    public const string AccessControlExposeHeaders = "Access-Control-Expose-Headers";

    //Below - custom header names

    public const string Pagination = "Pagination";

}

## DataConstants.cs

namespace MSC.Core.Constants;

public class DataConstants

{

    public const int MinAge = 18;

    public const int MaxAge = 150;

    public const string LastActive = "lastActive";

    public const string Created = "created";

}

# MSC.Core/Dtos

## Pagination

### PagedList.cs

* This will work with any type of object/class so PagedList<T> where T is UserDto
* We are getting users (so List<T>) and will send through this class which will help with pagination
* There is static metod which will receive the data, it will call the contructor and then return the data

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

namespace MSC.Core.Dtos.Pagination;

// This will work with any type of object/class so PagedList<T> where T is UserDto

// We are getting users (so List<T>) and will send through this class which will help with pagination

// There is static metod which will receive the data, it will call the contructor and then return the data

public class PagedList<T> : List<T>

{

    /// <summary>

    ///

    /// </summary>

    /// <param name="items">The data</param>

    /// <param name="count">The total number of records</param>

    /// <param name="pageNumber">The pageNumber interested in - currentPage</param>

    /// <param name="pageSize">The total number of records per page</param>

    public PagedList(IEnumerable<T> items, int count, int pageNumber, int pageSize)

    {

        CurrentPage = pageNumber;

        PageSize = pageSize;

        TotalCount = count;

        //lets say we have count=10, and pageSize=4, we will get TotalPages=3

        TotalPages = (int) Math.Ceiling(count/(double) pageSize);

        //to have access to the items in the page list

        AddRange(items);

    }

    /// <summary>

    /// Current page number

    /// </summary>

    public int CurrentPage { get; set; }

    /// <summary>

    /// Total pages

    /// </summary>

    public int TotalPages { get; set; }

    /// <summary>

    /// Page size, total records to pull for the page

    /// </summary>

    public int PageSize { get; set; }

    /// <summary>

    /// Total records available

    /// </summary>

    public int TotalCount { get; set; }

    //static method which will receive the IQueryable pageNumber and pageSize and return the page data

    public static async Task<PagedList<T>> CreateAsync(IQueryable<T> source, int pageNumber, int pageSize)

    {

        //total count

        var count = await source.CountAsync();

        //items, when on page #1 we do not want to skip any thing. So use pagenumber - 1

        var items = await source.Skip((pageNumber-1)\*pageSize).Take(pageSize).ToListAsync();

        //return new paged list

        var data = new PagedList<T>(items, count, pageNumber, pageSize);

        return data;

    }

}

### PaginationHeader.cs

When the response will go to the client the information will be taken from Pagination Header.

namespace MSC.Core.Dtos.Pagination;

/// <summary>

/// When the response will go to the client the information will be taken from Pagination Header

/// </summary>

public class PaginationHeader

{

    public PaginationHeader(int currentPage, int totalPages, int itemsPerPage, int totalItems)

    {

        CurrentPage = currentPage;

        TotalPages = totalPages;

        ItemsPerPage = itemsPerPage;

        TotalItems = totalItems;

    }

    /// <summary>

    /// Current page number

    /// </summary>

    public int CurrentPage { get; set; }

    /// <summary>

    /// Total pages

    /// </summary>

    public int TotalPages { get; set; }

    /// <summary>

    /// Page size, total records to pull for the page

    /// </summary>

    public int ItemsPerPage { get; set; }

    /// <summary>

    /// Total records available

    /// </summary>

    public int TotalItems { get; set; }

}

### PaginationParams.cs

This is the base class for

namespace MSC.Core.Dtos.Pagination;

/// <summary>

/// Base class to be used by search  classes like UserSearchParamsDto

/// </summary>

public class PaginationParams

{

    //set a constant for the max page size

    private const int MaxPageSize = 50;

    /// <summary>

    /// page number user is requesting. Default is page #1

    /// </summary>

    public int PageNumber { get; set; } = 1;

    //default for the page size

    private int \_pageSize = 10;

    public int PageSize

    {

        //the default in this case will be 10

        get => \_pageSize;

        //when pageSize is greater than MaxPageSize then return MaxPageSize

        set => \_pageSize = (value > MaxPageSize) ? MaxPageSize : value;

    }

}

## UsersSearchParamsDto

using MSC.Core.Constants;

using MSC.Core.Dtos.Pagination;

namespace MSC.Core.Dtos;

public class UsersSearchParamDto : PaginationParams

{

    public string Gender { get; set; }

    public int MinAge { get; set; } = DataConstants.MinAge;

    public int MaxAge { get; set; } = DataConstants.MaxAge;

    public string OrderBy { get; set; } = DataConstants.LastActive;

}

# MSC.Core/Extensions

## JsonExtensions.cs

using System.Text.Json;

namespace MSC.Core.Extensions;

public static class JsonExtensions

{

    /// <summary>

    /// ToJson

    /// </summary>

    /// <typeparam name="T"></typeparam>

    /// <param name="data"></param>

    /// <param name="isCamelCase">Default is true</param>

    /// <returns></returns>

    public static string Serialize<T>(this T data, bool isCamelCase = true)

    {

        var jsonString = "";

        if(data == null) return jsonString;

        if(isCamelCase){

            var jsonOptions = new JsonSerializerOptions{PropertyNamingPolicy = JsonNamingPolicy.CamelCase};

            jsonString = JsonSerializer.Serialize<T>(data, jsonOptions);

        }

        else{

            jsonString = JsonSerializer.Serialize<T>(data);

        }

        return jsonString;

    }

    /// <summary>

    /// ToJson Indented

    /// </summary>

    /// <typeparam name="T"></typeparam>

    /// <param name="data"></param>

    /// <param name="isCamelCase"></param>

    /// <returns></returns>

    public static string SerializeIndented<T>(this T data, bool isCamelCase = true)

    {

        var jsonString = "";

        if(data == null) return jsonString;

        var options = new JsonSerializerOptions { WriteIndented = true };

        if(isCamelCase)

            options.PropertyNamingPolicy = JsonNamingPolicy.CamelCase;

        jsonString = JsonSerializer.Serialize<T>(data);

        return jsonString;

    }

    /// <summary>

    /// From Json

    /// </summary>

    /// <typeparam name="T"></typeparam>

    /// <param name="jsonString"></param>

    /// <returns></returns>

    public static T Deserialize<T>(this string jsonString)

    {

        if(string.IsNullOrWhiteSpace(jsonString)) return default(T);

        var data = JsonSerializer.Deserialize<T>(jsonString);

        return data;

    }

}

## HttpExtensions.cs

using System.Collections.Generic;

using Microsoft.AspNetCore.Http;

using MSC.Core.Constants;

using MSC.Core.Dtos.Pagination;

namespace MSC.Core.Extensions;

public static class HttpExtensions

{

    public static void AddPaginationHeader(this HttpResponse response, PaginationHeader header)

    {

        var headerJson = header.Serialize(); //toJson

        //response.Headers.Add(HeaderNameConstants.Pagination, headerJson);

        response.Headers.TryAdd(HeaderNameConstants.Pagination, headerJson);

        //since custom header, allow cors policy. Otherwise the clients will not be able to access the header

        response.Headers.TryAdd(HeaderNameConstants.AccessControlExposeHeaders, HeaderNameConstants.Pagination);

    }

    public static void AddPaginationHeader(this HttpResponse response, int currentPage, int itemsPerPage, int totalItems, int totalPages)

    {

        var header = new PaginationHeader(currentPage, totalPages, itemsPerPage, totalItems);

        var headerJson = header.Serialize(); //toJson

        //response.Headers.Add(HeaderNameConstants.Pagination, headerJson);

        response.Headers.TryAdd(HeaderNameConstants.Pagination, headerJson);

        //since custom header, allow cors policy. Otherwise the clients will not be able to access the header

        response.Headers.TryAdd(HeaderNameConstants.AccessControlExposeHeaders, HeaderNameConstants.Pagination);

    }

}

## DateTimeExtensions.cs

Add two new extensions to calculate min and max date

    /// <summary>

    /// The oldest the person can be

    /// </summary>

    /// <param name="maxAge">where dob >= this date</param>

    /// <returns></returns>

    public static DateOnly CalculateMinDob(this int maxAge)

    {

        if (maxAge <= 0) maxAge = DataConstants.MaxAge;

        var dob = DateOnly.FromDateTime(DateTime.Today.AddYears(-maxAge - 1));

        return dob;

    }

    /// <summary>

    /// The youngest the person can be

    /// </summary>

    /// <param name="minAge">where dob <= this date</param>

    /// <returns></returns>

    public static DateOnly CalculateMaxDob(this int minAge)

    {

        if (minAge <= 0) minAge = DataConstants.MinAge;

        var dob = DateOnly.FromDateTime(DateTime.Today.AddYears(-minAge));

        return dob;

    }

# MSC.Core/Repositories

## UsersRepository

Use the pagination and filtering and sorting

### IUserRepository.cs

Updated method

    //same as above "GetUsersAsync" but using auto mapper queryable extensions

    //with pagination changed the signature

    //Task<IEnumerable<UserDto>> GetUsersAMQEAsync();

    Task<PagedList<UserDto>> GetUsersAMQEAsync(UsersSearchParamDto userParams, Guid userGuid);

Updated method

    Task<AppUser> GetUserAsync(int id, bool includePhotos = false);

### UserRepository.cs

Updated method

//same as above "GetUsersAsync" but using auto mapper queryable extensions

    //with pagination changed the signature

    //public async Task<IEnumerable<UserDto>> GetUsersAMQEAsync()

    public async Task<PagedList<UserDto>> GetUsersAMQEAsync(UsersSearchParamDto userParams, Guid userGuid)

    {

        /\*

         var users = await \_context.Users

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .ToListAsync();

        return users;

        \*/

        var query = \_context.Users.AsQueryable();

        //apply filters

        query = query.Where(u =>u.Guid != userGuid);

        if(!string.IsNullOrWhiteSpace(userParams.Gender))

            query = query.Where(u => u.Gender == userParams.Gender);

        var minDob = userParams.MaxAge.CalculateMinDob();

        var maxDob = userParams.MinAge.CalculateMaxDob();

        query = query.Where(u => u.DateOfBirth >= minDob && u.DateOfBirth <= maxDob);

        if (!string.IsNullOrWhiteSpace(userParams.OrderBy))

        {

            //the new switch statement. \_ is the default

            query = userParams.OrderBy switch

            {

                DataConstants.Created => query.OrderByDescending(u => u.CreatedOn),

                \_ => query.OrderByDescending(u => u.LastActive)

            };

        }

        var finalQuery = query

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsNoTracking() //EF will not track

                    ;

        //user pagedList class

        var pageList = await PagedList<UserDto>.CreateAsync(finalQuery, userParams.PageNumber, userParams.PageSize);

        return pageList;

    }

Updated method

    public async Task<AppUser> GetUserAsync(int id, bool includePhotos = false)

    {

        AppUser user  = null;

        if(includePhotos)

        {

            user = await \_context.Users

                                .Include(p => p.Photos)

                                .FirstOrDefaultAsync(x => x.Id == id);

        }

        else{

            user = await \_context.Users.FindAsync(id);

        }

        return user;

    }

# MSC.Core/BusinessLogic

## UserBusinessLogic

### IUserBusinessLogic.cs

    //same as above "GetUsersAsync" but using auto mapper queryable extensions

    //signature changed due to pagination

    //Task<IEnumerable<UserDto>> GetUsersAMQEAsync();

    Task<PagedList<UserDto>> GetUsersAMQEAsync(UsersSearchParamDto userParams, Guid userGuid);

And also add another signature for the action filter to update lastActiveDate for the user

    Task LogUserActivityAsync(int id);

### UserBusinessLogic.cs

    //same as above "GetUsersAsync" but using auto mapper queryable extensions

    //signature changed due to pagination

    public async Task<PagedList<UserDto>> GetUsersAMQEAsync(UsersSearchParamDto userParams, Guid userGuid)

    {

        var users = await \_userRepo.GetUsersAMQEAsync(userParams, userGuid);

        if(users == null || !users.Any()) return null;

        return users;

    }

And the method for lastActive date

    public async Task LogUserActivityAsync(int id)

    {

       if(id <= 0) return;

        //app user

        var user = await \_userRepo.GetUserAsync(id);

        if (user == null) return;

        //update the last active date

        user.LastActive = DateTime.UtcNow;

        //update

        await \_userRepo.SaveAllAsync();

    }

# UsersController

## Updated GetUsers with pagedlist

    /// <summary>

    ///

    /// </summary>

    /// <param name="userParams">Passed as query string</param>

    /// <returns></returns>

    [HttpGet]

    public async Task<ActionResult<PagedList<UserDto>>> GetUsers([FromQuery]UsersSearchParamDto userParams)

    {

        //get current user

        var currentUser = await \_userBusinessLogic.GetUserAMQEAsync(User.GetUserName());

        if(currentUser == null)

            return BadRequest("User issue");

        //var users = await \_userBusinessLogic.GetUsersAsync();

        if(string.IsNullOrWhiteSpace(userParams.Gender))

            userParams.Gender = currentUser.Gender.ToLower() == "male" ? "female" : "male";

        var users = await \_userBusinessLogic.GetUsersAMQEAsync(userParams, currentUser.GuId);

        if (users == null || !users.Any())

        {

            return NotFound("No users found!");

        }

        //write pagination header

        Response.AddPaginationHeader(users.CurrentPage, users.PageSize, users.TotalCount, users.TotalPages);

        return Ok(users);

    }

# ActionFilters

## MSC.Core/ActionFilters

### LogUserActivityFilter.cs

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.Extensions.DependencyInjection;

using MSC.Core.BusinessLogic;

using MSC.Core.Extensions;

namespace MSC.Core.ActionFilters;

//applied to the base controller

//also, added as a service using extention AddServices

public class LogUserActivityFilter : IAsyncActionFilter

{

    public async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)

    {

        //gives us ActionExecutedContext meaning the api action has completed and wll get the result context back from it

        //if we want to something before the action then use ActionExecutingContext as context

        var resultContext = await next();

        //ensure that user is authenicated

        if(!resultContext.HttpContext.User.Identity.IsAuthenticated)

            return;

        //can also use individual items or full user

        var userName = resultContext.HttpContext.User.GetUserName();

        var guid = resultContext.HttpContext.User.GetGuid();

        var id = resultContext.HttpContext.User.GetId();

        var claims = resultContext.HttpContext.User.GetUserClaims();

        if (claims == null)

            return;

        //get the reference to the user business logic

        var userBl = resultContext.HttpContext.RequestServices.GetRequiredService<IUserBusinessLogic>();

        //log activity

        await userBl.LogUserActivityAsync(id);

    }

}

## MSC.Core/Extensions

### AppServiceExtensions.cs

#### AddServices

    public static IServiceCollection AddServices(this IServiceCollection services, IConfiguration config)

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        services.AddScoped<IPhotoService, PhotoService>();

        //services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()); //when have single project/assembly

        services.AddAutoMapper(typeof(AutoMapperProfiles).Assembly);

        //adding the Cloudinary to read data from

        //check programs.cs for ref: builder.Services.Configure<EnvConfig>(configuration);

        services.Configure<CloudinaryConfig>(config.GetSection(ConfigKeyConstants.CloudinarySettingsKey));

        //add the action filter as a service, it wil get applied to the abse controller

        services.AddScoped<LogUserActivityFilter>();

        return services;

    }

## Controllers

### BaseApiController

using Microsoft.AspNetCore.Mvc;

using MSC.Core.ActionFilters;

namespace MSC.WebApi.Controller;

[ApiController]

[Route("api/[controller]")]

[ServiceFilter(typeof(LogUserActivityFilter))]

public class BaseApiController : ControllerBase

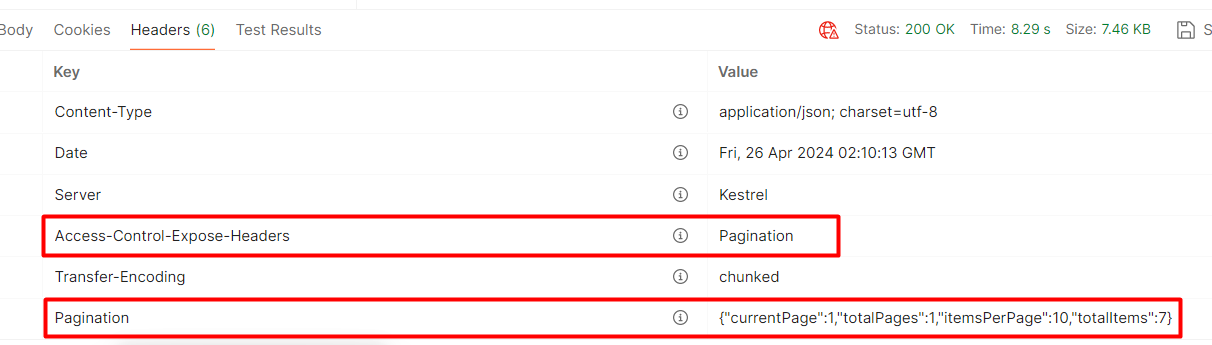
{

}

# Postman

Check Site-12 collection for details in Postman\_collection.json

## Test 1: No params



## Test 2: 2 Items Per Page

