Table of Contents

[Basics 3](#_Toc116512777)

[Current Resources with Changes 3](#_Toc116512778)

[New Resources 3](#_Toc116512779)

[/Core/Constants 3](#_Toc116512780)

[DataConstants.cs 3](#_Toc116512781)

[HeaderNameConstants.cs 3](#_Toc116512782)

[/Core/Dto 4](#_Toc116512783)

[UserClaimsGetDto.cs 4](#_Toc116512784)

[UserTokenDto.cs 4](#_Toc116512785)

[/Core/Dto/Helpers 4](#_Toc116512786)

[PageList.cs 4](#_Toc116512787)

[PaginationHeader.cs 5](#_Toc116512788)

[PaginationParams.cs 5](#_Toc116512789)

[UserParams.cs 6](#_Toc116512790)

[/Core/Services 6](#_Toc116512791)

[TokenService 6](#_Toc116512792)

[/Core/Extensions 7](#_Toc116512793)

[ClaimsPrincipalExtensions.cs 7](#_Toc116512794)

[JsonExtensions.cs 8](#_Toc116512795)

[HttpExtensions.cs 8](#_Toc116512796)

[DateTimeExtensions.cs 9](#_Toc116512797)

[/Core/Repositories 9](#_Toc116512798)

[UsersRepository 9](#_Toc116512799)

[IUsersRepository.cs 9](#_Toc116512800)

[UsersRepository.cs 9](#_Toc116512801)

[/Core/BusinessLogic 10](#_Toc116512802)

[UsersBisnessLogic 10](#_Toc116512803)

[IUsersBusinessLogic.cs 10](#_Toc116512804)

[UsersBusinessLogic.cs 10](#_Toc116512805)

[/Core/ActionFilters 13](#_Toc116512806)

[LogUserActivityFilter.cs 13](#_Toc116512807)

[/Core/Extensions/ServiceExtensions.cs 14](#_Toc116512808)

[RegisterRepos Method 14](#_Toc116512809)

[/Controllers/BaseApiController 14](#_Toc116512810)

[Controllers 14](#_Toc116512811)

[UsersController.cs 14](#_Toc116512812)

[Postman Testing 14](#_Toc116512813)

# Basics

|  |  |
| --- | --- |
| WorkingFolder | Copy the content of “Site-01-Basics” in “Site-02-PagingSortingFiltering” and issue   * dotnet build |
| Cloudinary Setting | appsetting.json is not checked in so make sure to bring in this from “Site-01-Basics” |

# Current Resources with Changes

1. /Core/Dto/UserTokenDto
2. /Core/Repositories
3. /Core/BusinessLogic
4. /Core/Extensions/DateTimeExtensions
5. /Core/TokenService
6. /Core/Extensions/ClaimsPrincipalExtensions
7. /Controllers/UsersControllers

# New Resources

1. /Core/Dto/Helpers/PageList
2. /Core/Dto/Helpers/PaginationHeader
3. /Core/Dto/Helpers/PaginationParams
4. /Core/Dto/Helpers/UserParams
5. /Core/Constants/DataConstantes
6. /Core/Constants/HeaderNameConstants
7. /Core/Extensions/JsonExtensions
8. /Core/Extensions/HttpExtensions
9. /Core/ActionFilters

# /Core/Constants

## DataConstants.cs

namespace MSC.Api.Core.Constants;

public class DataConstents

{

    public const int MinAge = 18;

    public const int MaxAge = 150;

    public const string LastActive = "lastActive";

    public const string Created = "created";

}

## HeaderNameConstants.cs

namespace MSC.Api.Core.Constants;

public class HeaderNameConstants

{

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

    //Below - custom header names

    public const string Pagination = "Pagination";

}

# /Core/Dto

## UserClaimsGetDto.cs

Add a new int property UserId to it

using System;

namespace MSC.Api.Core.Dto;

public class UserClaimGetDto

{

    public string UserName { get; set; }

    public int UserId { get; set; }

    public Guid Guid { get; set; }

    public string DisplayName { get; set; }

    public bool HasUserName => !string.IsNullOrWhiteSpace(UserName);

    public bool HasGuid => Guid != Guid.Empty;

}

## UserTokenDto.cs

Return Gender as well. No other change is needed since the auto mapper will automatically add the Gender.

using System;

namespace MSC.Api.Core.Dto;

public class UserTokenDto

{

    public string UserName { get; set; }

    public string DisplayName { get; set; }

    public Guid GuId { get; set; }

    public string Token { get; set; }

    public string MainPhotoUrl { get; set; }

    public string Gender { get; set; }

}

# /Core/Dto/Helpers

## PageList.cs

A generic helper class to help with pagination

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

namespace MSC.Api.Core.Dto.Helpers;

public class PageList<T> : List<T>

{

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

    {

        CurrentPage = pageNumber;

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

        PageSize = pageSize;

        TotalCount = count;

        //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<PageList<T>> CreateAsync(IQueryable<T> source, int pageNumber, int pageSize)

    {

        //get the total count

        var count = await source.CountAsync();

        //skip the pages to go to the intended page pick the records

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

        //return the data as paged list

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

        return data;

    }

}

## PaginationHeader.cs

Pagination info will be sent back in the Headers so need a helper class for it

namespace MSC.Api.Core.Dto.Helpers;

public class PaginationHeader

{

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

    {

        CurrentPage = currentPage;

        ItemsPerPage = itemsPerPage;

        TotalItems = totalItems;

        TotalPages = totalPages;

    }

    public int CurrentPage { get; set; }

    public int ItemsPerPage { get; set; }

    public int TotalItems { get; set; }

    public int TotalPages { get; set; }

}

## PaginationParams.cs

To receive instructions from the suer like the page number and max page size

namespace MSC.Api.Core.Dto.Helpers;

//this will be used by get controller actions to receieve instruction from client

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;

    }

}

## UserParams.cs

UserParams inherit from RequestRecordParams for common items

using System;

using MSC.Api.Core.Constants;

namespace MSC.Api.Core.Dto.Helpers;

public class UserParams : PaginationParams

{

    //user filtering parameters

    public Guid? CurrentUserGuid { get; set; }

    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;

}

# /Core/Services

## TokenService

Update the claims to use the id and username both

    public string CreateToken(AppUser user)

    {

        if (user == null)

            throw new Exception("User info missing");

        //claims

        var claims = new List<Claim>

        {

            new Claim(JwtRegisteredClaimNames.NameId, user.Id.ToString()),

            new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName),

            new Claim("Guid", user.GuId.ToString()),

            new Claim("DisplayName", user.DisplayName),

        };

        //credentials with key

        var creds = new SigningCredentials(\_key, SecurityAlgorithms.HmacSha512Signature);

        //describe token

        var tokenDescriptor = new SecurityTokenDescriptor

        {

            Subject = new ClaimsIdentity(claims),

            Expires = DateTime.Now.AddDays(7),

            SigningCredentials = creds

        };

        //token handler

        var tokenHandler = new JwtSecurityTokenHandler();

        //token

        var token = tokenHandler.CreateToken(tokenDescriptor);

        var writeToken = tokenHandler.WriteToken(token);

        return writeToken;

    }

# /Core/Extensions

## ClaimsPrincipalExtensions.cs

Update to pick the userId int as well

using System;

using System.Security.Claims;

using MSC.Api.Core.Dto;

namespace MSC.Api.Core.Extensions;

/// <summary>

/// Moved the method GetLoggedInCalims from base controller here. Original method is commented there

/// https://www.jerriepelser.com/blog/useful-claimsprincipal-extension-methods/

/// </summary>

public static class ClaimsPrincipalExtensions

{

    public static string GetUserEmail(this ClaimsPrincipal principal)

    {

        if (principal == null) return string.Empty;

        //return principal.FindFirstValue(ClaimTypes.Email);

        var email = principal.FindFirst(ClaimTypes.Email)?.Value;

        return email;

    }

    public static string GetUserName(this ClaimsPrincipal principal)

    {

        if (principal == null) return string.Empty;

        //return principal.FindFirstValue(ClaimTypes.NameIdentifier);

        var userName = principal.FindFirst(ClaimTypes.Name)?.Value;

        return userName;

    }

    public static int GetUserId(this ClaimsPrincipal principal)

    {

        if (principal == null) return 0;

        //return principal.FindFirstValue(ClaimTypes.NameIdentifier);

        var userId = int.Parse(principal.FindFirst(ClaimTypes.NameIdentifier)?.Value);

        return userId;

    }

    public static string GetDisplayName(this ClaimsPrincipal principal)

    {

        if (principal == null) return string.Empty;

        var displayName = principal.FindFirst("DisplayName")?.Value;

        return displayName;

    }

    public static Guid GetUserGuid(this ClaimsPrincipal principal)

    {

        var getGuid = Guid.Empty;

        if (principal == null) return getGuid;

        var guid = principal.FindFirst("Guid")?.Value;

        if (string.IsNullOrWhiteSpace(guid)) return getGuid;

        try

        {

            getGuid = new Guid(guid);

        }

        catch { }

        return getGuid;

    }

    public static UserClaimGetDto GetUserClaims(this ClaimsPrincipal principal)

    {

        if (principal == null) return null;

        var claimsDto = new UserClaimGetDto()

        {

            UserId = principal.GetUserId(),

            UserName = principal.GetUserName(),

            Guid = principal.GetUserGuid(),

            DisplayName = principal.GetDisplayName()

        };

        return claimsDto;

    }

}

## JsonExtensions.cs

using System.Text.Json;

namespace MSC.Api.Core.Extensions;

public static class JsonExtensions

{

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

    {

        if (data == null) return string.Empty;

        var jsonString = "";

        if (isCamelCase)

        {

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

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

        }

        else

        {

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

        }

        return jsonString;

    }

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

    {

        if (data == null) return string.Empty;

        var jsonString = "";

        var options = new JsonSerializerOptions { WriteIndented = true };

        if (isCamelCase)

        {

            options.PropertyNamingPolicy = JsonNamingPolicy.CamelCase;

        }

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

        return jsonString;

    }

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

    {

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

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

        return data;

    }

}

## HttpExtensions.cs

An extension method to add pagination header to HTTP response.

using Microsoft.AspNetCore.Http;

using MSC.Api.Core.Constants;

using MSC.Api.Core.Dto.Helpers;

namespace MSC.Api.Core.Extensions;

public static class HttpExtensions

{

    /// <summary>

    /// add pagination header onto the response

    /// </summary>

    /// <param name="response"></param>

    /// <param name="currentPage"></param>

    /// <param name="itemsPerPage"></param>

    /// <param name="totalItems"></param>

    /// <param name="totalPages"></param>

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

    {

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

        var paginationHeaderSerialize = paginationHeader.ToJson();

        //write custom header. No more adding X- to it. Give a sensible name

        response.Headers.Add(HeaderNameConstants.Pagination, paginationHeaderSerialize);

        //need to add the CORS header as well since a custom header is being used to make it available

        //cors header must be specific name

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

    }

}

## DateTimeExtensions.cs

Add two new methods to calculate the min and max don dates

    /// <summary>

    /// The oldest the person can be

    /// </summary>

    /// <param name="maxAge"></param>

    /// <returns></returns>

    public static DateTime CalculateMinDob(this int maxAge)

    {

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

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

        return dob;

    }

    /// <summary>

    /// The youngest the person can be

    /// </summary>

    /// <param name="minAge"></param>

    /// <returns></returns>

    public static DateTime CalculateMaxDob(this int minAge)

    {

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

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

        return dob;

    }

# /Core/Repositories

## UsersRepository

### IUsersRepository.cs

    //Rather than returning list of userDto, will return pageList<userDto>

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

    Task<PageList<UserDto>> GetUsersAsync(UserParams userParams);

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

### UsersRepository.cs

    //return PageList<userDto> rather than list of userDto

    /\*

    public async Task<IEnumerable<UserDto>> GetUsersAsync()

    {

        //var users = await \_context.Users.ToListAsync();

        //add photos as eager loading

        //var users = await \_context.Users.Include(p => p.Photos).ToListAsync();

        //return users;

        //using automapper queryable extensions

        var users = await \_context.Users

                            .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                            .AsSplitQuery()

                            .AsNoTracking()

                            .ToListAsync();

        return users;

    }

    \*/

    public async Task<PageList<UserDto>> GetUsersAsync(UserParams userParams)

    {

        //check above for initial implementation

        //query is IQueryable

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

        //apply filters

        query = query.Where(u => u.GuId != userParams.CurrentUserGuid.Value);

        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);

        //projectTo to get the photos

        var finalQuery = query

                        .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                        .AsNoTracking();

        //page list has the static method that receive the IQueryable so use it and will return the object

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

        return pageList;

    }

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

    {

        if (id <= 0)

            throw new ValidationException("Invalid userName");

        AppUser user = null;

        if (!includePhotos)

            user = await \_context.Users.SingleOrDefaultAsync(x => x.Id == id);

        else

            user = await \_context.Users.Include(p => p.Photos).SingleOrDefaultAsync(x => x.Id == id);

        return user;

    }

# /Core/BusinessLogic

## UsersBisnessLogic

### IUsersBusinessLogic.cs

    //it is returning pageList<userDto>

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

Task<PageList<UserDto>> GetUsersAsync(UserParams userParams);

    /// <summary>

    /// used by the LogUserAcitivty Action Filter

    /// </summary>

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

    /// <returns></returns>

    Task LogUserActivityAsync(int userId);

### UsersBusinessLogic.cs

Add the new LogUserActivityAsync and also update all the methods to use the new GetAppUser by id method since that is much faster where ever by username is being used.

    //it is returning pageList<userDto>

    /\*

    public async Task<IEnumerable<UserDto>> GetUsersAsync()

    {

        var users = await \_usersRepo.GetUsersAsync();

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

        //var userDto = users.Select(x => new UserDto { Id = x.Id, UserName = x.UserName }).ToList();

        //var userDto = \_mapper.Map<IEnumerable<UserDto>>(users);

        //return userDto;

        return users;

    }

    \*/

    public async Task<PageList<UserDto>> GetUsersAsync(UserParams userParams)

    {

        var users = await \_usersRepo.GetUsersAsync(userParams);

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

        return users;

    }

    public async Task<bool> UpdateUserAsync(UserUpdateDto userUpdateDto, UserClaimGetDto claims)

    {

        var user = await \_usersRepo.GetAppUserAsync(claims.UserId);

        if (user == null || user.GuId != claims.Guid)

            return false;

        //data from the userUpdateDto will be moved to user while the rest of the properties will be kept as is

        var updates = \_mapper.Map(userUpdateDto, user);

        //issue update but it will not save

        \_usersRepo.Update(updates);

        //save update

        if (await \_usersRepo.SaveAllAsync())

            return true;

        return false;

    }

    /// <summary>

    /// used by the LogUserAcitivty Action Filter

    /// </summary>

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

    /// <returns></returns>

    public async Task LogUserActivityAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName)) return;

        //app user

        var user = await \_usersRepo.GetAppUserAsync(userName);

        if (user == null) return;

        //update the last active date

        user.LastActive = DateTime.Now;

        //update

        await \_usersRepo.SaveAllAsync();

    }

    public async Task<PhotoDto> AddPhoto(IFormFile file, UserClaimGetDto claims)

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserId, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

        var result = await \_photoService.AddPhotoAsync(file);

        //error

        if (result.Error != null)

            throw new DataFailException(result.Error?.Message ?? "Photo updload error"); //exception middleware

        //success, build photo entity and save

        var photo = new Photo

        {

            Url = result.SecureUrl.AbsoluteUri, //set photo url

            PublicId = result.PublicId, //set public id

            IsMain = appUser.Photos == null || !appUser.Photos.Any() //mark it active when no other photos are available

        };

        //add the photo. Photos is an abstract method so cannot be null

        appUser.Photos.Add(photo);

        if (await \_usersRepo.SaveAllAsync())

        {

            return \_mapper.Map<PhotoDto>(photo);

        }

        return null;

    }

    public async Task<bool> SetPhotoMain(int photoId, UserClaimGetDto claims)

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserId, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

        var photo = appUser.Photos?.FirstOrDefault(x => x.Id == photoId);

        if (photo == null)

            return false;

        if (photo.IsMain)

            throw new DataFailException("This is already your main photo"); //exception middleware

        var currentMain = appUser.Photos.FirstOrDefault(x => x.IsMain == true);

        if (currentMain != null)

            currentMain.IsMain = false;

        photo.IsMain = true;

        if (await \_usersRepo.SaveAllAsync())

            return true;

        return false;

    }

   public async Task<BusinessResponse> DeletePhoto(int photoId, UserClaimGetDto claims)

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserId, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

        var response = new BusinessResponse();

        var photo = appUser.Photos?.FirstOrDefault(x => x.Id == photoId);

        if (photo == null)

        {

            response.HttpStatusCode = HttpStatusCode.NotFound;

            response.Message = "Photo not found";

            return response;

        }

        if (photo.IsMain)

        {

            response.HttpStatusCode = HttpStatusCode.BadRequest;

            response.Message = "You cannot delete your main photo";

            return response;

        }

        //delete from cloudinary

        if (photo.PublicId != null)

        {

            var result = await \_photoService.DeletePhotoAync(photo.PublicId);

            if (result.Error != null)

            {

                response.HttpStatusCode = HttpStatusCode.BadRequest;

                response.Message = result.Error.Message;

                return response;

            }

        }

        //remove from the database as well

        appUser.Photos.Remove(photo);

        if (await \_usersRepo.SaveAllAsync())

        {

            response.HttpStatusCode = HttpStatusCode.OK;

            return response;

        }

        //it is an error then

        response.HttpStatusCode = HttpStatusCode.BadRequest;

        response.Message = "Unable to delete photo";

        return response;

    }

# /Core/ActionFilters

Create a new folder under /Core and name it ActionFilters

## LogUserActivityFilter.cs

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.Extensions.DependencyInjection;

using MSC.Api.Core.BusinessLogic;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Core.ActionFilters;

public class LogUserActivityFilter : IAsyncActionFilter

{

    public async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)

    {

        //update after the activity so use next

        var resultContext = await next();

        //user must be logged in

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

            return;

        //we can get the individual properties or the full claims object that has every thing

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

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

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

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

        if (claims == null)

            return;

        //get the reference to the user business logic

        var userBl = resultContext.HttpContext.RequestServices.GetService<IUsersBusinessLogic>();

        //call method to update the last active date

        await userBl.LogUserActivityAsync(id);

    }

}

## /Core/Extensions/ServiceExtensions.cs

### RegisterRepos Method

Register the LogUserActivityFilter

        //action filters

        services.AddScoped<LogUserActivityFilter>();

## /Controllers/BaseApiController

Add the action filter as ServiceFilter to the baseApicontroller. This way it will get applied to all the controllers

[ServiceFilter(typeof(LogUserActivityFilter))]

[ApiController]

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

public class BaseApiController : ControllerBase

# Controllers

## UsersController.cs

    //due to pagination check the new implementation

    /\*

    [HttpGet]

    public async Task<ActionResult<IEnumerable<UserDto>>> GetUsers()

    {

        var users = await \_usersBl.GetUsersAsync();

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

        {

            return NotFound("No users found!");

        }

        return Ok(users);

    }

    \*/

    [HttpGet]

    public async Task<ActionResult<IEnumerable<UserDto>>> GetUsers([FromQuery] UserParams userParams)

    {

        //get the current user

        var user = await \_usersBl.GetUserAsync(User.GetUserName());

        if (user == null)

        {

            return BadRequest("User issue");

        }

        //for removing the curren tuser from the list

        userParams.CurrentUserGuid = user.GuId;

        //by default picking the opposite gender when not supplied

        if (string.IsNullOrWhiteSpace(userParams.Gender))

            userParams.Gender = user.Gender.ToUpperInvariant() == "MALE" ? "female" : "male";

        var users = await \_usersBl.GetUsersAsync(userParams);

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

        {

            return NotFound("No users found!");

        }

        //users has the pagination information so will need to write the pagination header using the extension we created

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

        return Ok(users);

    }

### Postman Testing

Method: Get

Url: <https://localhost:5000/api/users?pageNumber=3&pageSize=5>

User requested 3rd page with the page size of 5 items per page. This is the last page. Since total items are 13, the last page returned 3 items only.

Graphical user interface, application

Description automatically generated

Can keep on chaining query params

* >= 45 years old: ?minAge=45
* <= 35 years old: ?maxAge=35
* >= 18 and <= 32 years old: ?minAge=18&maxAge=32