# Return Gender when login / register

## LoginResponseDto.cs

Add a string property Gender and DisplayName

    public string Gender { get; set; }

    public string DisplayName { get; set; }

# TokenService

## Update Claims in CreateToken Method

Update the token service claims

|  |  |
| --- | --- |
| From | To |
| var claims = new List<Claim>()          {              new Claim(JwtRegisteredClaimNames.NameId, user.UserName),              new Claim("Guid", user.Guid.ToString())          }; | 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),          }; |

## Postman Test Login

A screenshot of a computer

Description automatically generated

## Check Token

<https://jwt.io/>

A screenshot of a computer

Description automatically generated

# UserClaimsReadDto

* Create a new class UserClaimsReadDto in folder Core/Dto/Claims
* This class will be used by claimsExtensions to return all the claims

namespace OAuth2.WebApi.Core.Dto;

public class UserClaimsReadDto

{

    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 != System.Guid.Empty;

}

# Extensions

## JsonExtensions

* These are helper extensions to convert Serialize and Deserialize
* Create new class JsonExtensions in folder Core/Extensions

using System.Text.Json;

namespace OAuth2.WebApi.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;

    }

}

## ClaimsPrincipalExtension.cs

This extension is used to read the logged in users claims

using System.Security.Claims;

using OAuth2.WebApi.Core.Dto;

namespace OAuth2.WebApi.Core.Extensions;

/// <summary>

/// 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 UserClaimsReadDto GetUserClaims(this ClaimsPrincipal principal)

    {

        if (principal == null) return null;

        var claimsDto = new UserClaimsReadDto()

        {

            UserId = principal.GetUserId(),

            UserName = principal.GetUserName(),

            Guid = principal.GetUserGuid(),

            DisplayName = principal.GetDisplayName()

        };

        return claimsDto;

    }

}

# Constants

## HeaderNameConstants

Create a class HeaderNameConstants in folder Core/Constants

namespace OAuth2.WebApi.Core.Constants;

public class HeaderNameConstants

{

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

    //Below - custom header names

    public const string Pagination = "Pagination";

}

## DataConstants

Create a class DataConstants in folder Core/Constants

namespace OAuth2.WebApi.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";

}

# Pagination

Create a new folder Pagination in folder Core >> Dto

## PagedList.cs

This is the paged list class to return data as pagination

using Microsoft.EntityFrameworkCore;

namespace OAuth2.WebApi.Core.Dto.Pagination;

/// <summary>

/// Pagination Helper Class

/// </summary>

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

public class PagedList<T> : List<T>

{

    /// <summary>

    ///

    /// </summary>

    /// <param name="items">List of items</param>

    /// <param name="count">Total count of items</param>

    /// <param name="pageNumber">Page number interested in</param>

    /// <param name="pageSize">Total records in a page</param>

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

    {

        //page number

        CurrentPage = pageNumber;

        //10 items with page size of 4 will end up with 3 pages

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

        //page size

        PageSize = pageSize;

        //total count

        TotalCount = count;

        //when we will create a new instance we will return the list of our items

        AddRange(items);

    }

    /// <summary>

    /// Current page number

    /// </summary>

    public int CurrentPage { get; private set; }

    /// <summary>

    /// Total pages by calculating against PageSize and TotalCount

    /// </summary>

    public int TotalPages { get; private set; }

    /// <summary>

    /// Number of records in a page

    /// </summary>

    public int PageSize { get; private set; }

    /// <summary>

    /// Total items

    /// </summary>

    public int TotalCount { get; private set; }

    /// <summary>

    /// Static method so that can call the pagedList and get back the page data

    /// </summary>

    /// <param name="source"></param>

    /// <param name="pageNumber"></param>

    /// <param name="pageSize"></param>

    /// <returns></returns>

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

    {

        //execute agains the database and get the total count

        var count = await source.CountAsync();

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

        //on page = 1 do not skip any pages

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

                                .Take(pageSize)

                                .ToListAsync();

        //return the data as paged list

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

        return data;

    }

}

## PaginationHeader

### PaginationHeader.cs

Helper method, this will be an object that we will return inside HTTP Response Headers

namespace OAuth2.WebApi.Core.Dto.Pagination;

/// <summary>

/// Helper method, this will be an object that we will return inside HTTP Response Headers

/// </summary>

public class PaginationHeader

{

    /// <summary>

    ///

    /// </summary>

    /// <param name="currentPage">The current page number</param>

    /// <param name="itemsPerPage">The page size</param>

    /// <param name="totalItems">The total items</param>

    /// <param name="totalPages">The total pages</param>

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

    {

        CurrentPage = currentPage;

        ItemsPerPage = itemsPerPage;

        TotalItems = totalItems;

        TotalPages = totalPages;

    }

    public int CurrentPage { get; private set; }

    public int ItemsPerPage { get; private set; }

    public int TotalItems { get; private set; }

    public int TotalPages { get; private set; }

}

### Write PaginationHeader to HTTP Response

* Create a new HttpExtensions in folder Core/Extensions
* Add a method to write PaginationHeader

using OAuth2.WebApi.Core.Constants;

using OAuth2.WebApi.Core.Dto.Pagination;

namespace OAuth2.WebApi.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);

        response.AddPaginationHeader(paginationHeader);

    }

    /// <summary>

    /// add pagination header onto the response

    /// </summary>

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

    /// <param name="paginationHeader"></param>

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

    {

        var paginationHeaderSerialize = paginationHeader.ToJson();

        //write custom header. No more adding X- to it. Give a sensible name, following will put "Pagination"

        var headerName = HeaderNameConstants.Pagination;

        //add header

        response.Headers.Add(headerName, 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, headerName);

    }

}

## Intake Params

Inorder for pagination to work will need to take parameters from the clients.

### PaginationParams.cs

Create a base class PaginationParams in folder Core/Dto/Pagination. It will be used by all other “params” classes like UserParams below

namespace OAuth2.WebApi.Core.Dto.Pagination;

public class PaginationParams

{

    //set a constant for the max page size

    private const int MaxPageSize = 50;

    //default for the page size

    private int \_pageSize = 10;

    /// <summary>

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

    /// </summary>

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

    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

Create a new class UserParams in folder Core/Dto/Pagination

using OAuth2.WebApi.Core.Constants;

namespace OAuth2.WebApi.Core.Dto.Pagination;

/// <summary>

/// user filtering parameters

/// </summary>

public class UserParams : PaginationParams

{

    public Guid? CurrentUserGuid { get; set; }

    public string Gender { get; set; }

    /// <summary>

    /// youngest

    /// </summary>

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

    /// <summary>

    /// oldest

    /// </summary>

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

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

}

# Extensions

## DatetimeExtensions.cs

Add the calculations for the MinDob and MaxDob

    /// <summary>

    /// The oldest the person can be. This is the minimum age. check against MaxAge

    /// </summary>

    /// <param name="age"></param>

    /// <returns></returns>

    public static DateOnly CalculateMinDob(this int age)

    {

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

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

        return dob;

    }

    /// <summary>

    /// The youngest the person can be. check against min age

    /// </summary>

    /// <param name="age"></param>

    /// <returns></returns>

    public static DateOnly CalculateMaxDob(this int age)

    {

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

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

        return dob;

    }

# Repository Updates

## IUserRepository.cs

Update GetUsersAsync

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

## UserRepository.cs

Update GetUsersAsync method

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

    {

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

        //apply filters

        if (userParams.CurrentUserGuid.HasValue)

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

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

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

        /\*

        var minDob = userParams.MinAge.CalculateMinDob();

        var maxDob = userParams.MaxAge.CalculateMaxDob();

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

        \*/

        //order by

        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)

            };

        }

        //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 PagedList<UserDto>.CreateAsync(finalQuery, userParams.PageNumber, userParams.PageSize);

        return pageList;

    }

# BusinessLogic Updates

## IUserBusinessLogic.cs

Update method GetUsersAsync

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

## UserBusinessLogic.cs

### Update GetUsersAsync

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

    {

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

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

        return users;

    }

### Update BuildLoginResponse

* This private method is used by both Register and Login.
* Add Gender and DisplayName both

    private LoginResponseDto BuildLoginResponse(AppUser appUser)

    {

        var loginResponse = new LoginResponseDto()

        {

            UserName = appUser.UserName,

            Guid = appUser.Guid,

            Token = \_tokenService.CreateToken(appUser),

            Gender = appUser.Gender,

            DisplayName = appUser.DisplayName

        };

        return loginResponse;

    }

# UsersController

## GetUsers EndPoint Update

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

    {

        //get the logged in users claims

        //User has the logged in users info

        var userName = User.GetUserName();

        //get the current user from the db

        var currentUser = await \_userBL.GetUserAsync(userName);

        if (currentUser == null)

            return BadRequest("User issue");

        //filter the current user

        userParams.CurrentUserGuid = currentUser.Guid;

        //if gender is not supplied then pick the opposite gender of the logged in user

        if (string.IsNullOrWhiteSpace(userParams.Gender))

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

        //get the users

        var users = await \_userBL.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);

    }

# Updating LastActiveDate via ActionFilter

## UserBusinessLogic

Add a new method to update LastActive date.

### IUserBusinessLogic.cs

    /// <summary>

    /// used by the LogUserAcitivty Action Filter

    /// </summary>

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

    /// <returns></returns>

    Task LogUserActivityAsync(string userName);

### UserBusinessLogic.cs

    /// <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 \_userRepository.GetAppUserAsync(userName);

        if (user == null) return;

        //update the last active date

        user.LastActive = DateTime.Now;

        //update

        await \_userRepository.SaveAllAsync();

    }

## LogUserActivityFilter

Create a new folder ActionFilters in folder Core.

### LogUserActivityFilter.cs

using Microsoft.AspNetCore.Mvc.Filters;

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Extensions;

namespace OAuth2.WebApi.Core.ActionFilters;

public class LogUserActivityFilter : IAsyncActionFilter

{

    public async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)

    {

        //update after the activity so use next

        //we will get ActionExecutedContext here

        var resultContext = await next();

        var user = resultContext.HttpContext.User;

        //user must be logged in

        if (!user.Identity.IsAuthenticated)

            return;

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

        var userName = user.GetUserName();

        var guid = user.GetUserGuid();

        var id = user.GetUserId();

        var claims = user.GetUserClaims();

        if (claims == null)

            return;

        //get the reference to the user business logic

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

        //call method to update the last active date

        await userBl.LogUserActivityAsync(userName);

    }

}

## Register LogUserActivityFilter as a Service

### ServiceExtensions.cs

#### RegisterServices Method

Add the action filter

    public static void RegisterServices(this IServiceCollection services, IConfiguration configuration)

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        //AutoMapper

        //old way

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

        //new way

        services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

        //Action filters

        services.AddScoped<LogUserActivityFilter>();

    }

## BaseApiController -Apply Filter

We can apply the filter to the BaseApiController since there is check to see user is logged in or not.

[ApiController]

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

[ServiceFilter(typeof(LogUserActivityFilter))]

public class BaseApiController : ControllerBase

{

}

# Postman Testing

## Get Users - Pagination Filtering - None - Login as Lisa

A screenshot of a computer

Description automatically generated

Response Headers

A screenshot of a computer

Description automatically generated

Response Result

A screenshot of a computer

Description automatically generated

## Get Users - Pagination - Login as Lisa

* pageNumber=1
* pageSize=5

A screenshot of a computer

Description automatically generated

Response Headers

A screenshot of a computer

Description automatically generated

Response Data

A screenshot of a computer

Description automatically generated

## Get Users Mins age filtering

A screenshot of a computer

Description automatically generated

## Get Users Max Age filtering

A screenshot of a computer

Description automatically generated

## Get Users Order By Created

A screenshot of a computer

Description automatically generated