# Project

* Site-13-Api-Ng-like
  + Copied from: Site-12-Api-Ng-paging-sorting-filtering
* 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/DB/Entities.UserLike.cs | Controllers/LikesController.cs |
| Msc.Core/Dtos/LikeDto.cs |  |
| MSC.Core/LikeSearchParamDto.cs |  |
| MSC.Core/Enums/zUserLikeType.cs |  |
| MSC.Core/Repositories/LikeRepository |  |
| MSC.Core/BusinessLogic/LileBusinessLogic |  |
|  |  |
|  |  |
|  |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/DB/Entities/AppUser.cs |  |
| MSC.Core/DB/Data/DataContext.cs |  |
| MSC.Core/Extensions/AppServiceExtensions.cs |  |
| MSC.Core/Dtos/BusinessResponse.cs |  |
|  |  |
|  |  |

# Clear Database

Clear the users

dotnet ef database drop

dotnet ef database update

New migration add in this project [here](#_Migrations_and_database).

# Intro

We are setting up a table that will have two fields

* SourceUser
* TargetUser

We can track then

* Users I liked: where SourceUser is I
* Liked by other users: where source user is not I but TargetUser user is I
* In both cases “TargetUser” is the field to look at
* This is many to many relationship

# Database Update

## /MSC.Core/DB/Entities

### UserLike.cs

namespace  MSC.Core.DB.Entities;

//table name comes from datacontext as "Likes"

//key is combination of SourceUserId+argetUserId

public class UserLike

{

    //fully defining the relationship between AppUser and UserLike. CheckDB Context for relationships

    public AppUser SourceUser { get; set; }

    public int SourceUserId { get; set; }

    //fully defining the relationship between AppUser and UserLike. CheckDB Context for relationships

    public AppUser TargetUser { get; set; }

    public int TargetUserId { get; set; }

}

### AppUser.cs

    /// <summary>

    /// Other users that liked the logged in User. CheckDB Context for relationships

    /// </summary>

    public List<UserLike> LikedByUsers { get; set; }

    /// <summary>

    /// Users that the logged in user liked. CheckDB Context for relationships

    /// </summary>

    public List<UserLike> LikedUsers { get; set; }

## /MSC.Core/DB/Data

### DataContext.cs

Add the data set

    //table name will be created as "Likes"

    //creating the db set so that we can directly query

    public DbSet<UserLike> Likes { get; set; }

Configure the user like

    //override OnModelCreating to create the relationshops for the likes

    //give entities some configuration

    protected override void OnModelCreating(ModelBuilder builder)

    {

        base.OnModelCreating(builder);

        UserLikeSetup(builder);

    }

    //user like configuration

    private void UserLikeSetup(ModelBuilder builder)

    {

        //key is combination of SourceUserId and TargetUserId

        builder.Entity<UserLike>()

                .HasKey(k => new {k.SourceUserId, k.TargetUserId});

        //build relationships between AppUser and UserLike. Here the users liked by the logged in user

        builder.Entity<UserLike>()

            .HasOne(s => s.SourceUser) //UserLike field

            .WithMany(l => l.LikedUsers) //AppUser field

            .HasForeignKey(s => s.SourceUserId)

            //for sql server the same entity cannot have to cascades so one needs to DeleteBehavior.NoAction

            .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities.

        ;

        //build relationships between AppUser and UserLike. Here the logged in user liked by other users

        builder.Entity<UserLike>()

            .HasOne(t => t.TargetUser) //UserLike field

            .WithMany(l => l.LikedByUsers) //AppUser field

            .HasForeignKey(t => t.TargetUserId)

            //for sql server the same entity cannot have to cascades so one needs to DeleteBehavior.NoAction

            .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities

        ;

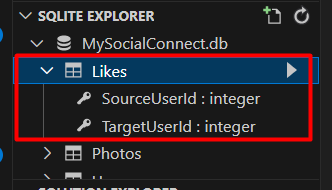
    }

## Migrations and database Update

> dotnet ef migrations add LikeEntityAdded

To undo above action, use 'ef migrations remove'

> dotnet ef database update



> dotnet watch --no-hot-reload

# MSC.Core/Enums

## zUserLiketype.cs

namespace MSC.Core.Enums;

public enum zUserLikeType

{

    None,

    Liked,

    LikedBy

}

# MSC.Core/Dtos

## LikeDto.cs

using System;

namespace MSC.Core.Dtos;

public class LikeDto

{

    public int Id { get; set; }

    public Guid GuId { get; set; }

    public string UserName { get; set; }

    public string PhotoUrl { get; set; } //custom where Photo isMain

    public int Age { get; set; }

    public string DisplayName { get; set; }

    public string City { get; set; }

}

## LikeSearchParamDto.cs

using MSC.Core.Dtos.Pagination;

using MSC.Core.Enums;

namespace MSC.Core.Dtos;

public class LikeSearchParamDto : PaginationParams

{

    public zUserLikeType UserLikeType {get; set;}

    public int UserId {get; set;}

}

## BusinessResponse.cs

Add the constructor.

using System.Net;

namespace MSC.Core.Dtos;

public class BusinessResponse

{

    public BusinessResponse()

    {

    }

    public BusinessResponse(HttpStatusCode httpStatusCode)

    {

        HttpStatusCode = httpStatusCode;

        Message = string.Empty;

    }

    public BusinessResponse(HttpStatusCode httpStatusCode, string message)

    {

        HttpStatusCode = httpStatusCode;

        Message = message;

    }

    public HttpStatusCode HttpStatusCode { get; set; }

    public string Message { get; set; }

}

# MSC.Core/Repositories/LikeRepository

## ILikesRepository.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

using MSC.Core.Dtos.Pagination;

namespace MSC.Core.Repositories;

public interface ILikesRepository

{

    //get the user like

    Task<UserLike> GetUserLike(int sourceUserId, int targetUserId);

    Task<AppUser> GetUserWithLikes(int userId);

    //Get UsersLiked and LikedBy

    Task<IEnumerable<LikeDto>> GetUserLikes(int userId, string predicate);

    //Get UsersLiked and LikedBy

    Task<PagedList<LikeDto>> GetUserLikes(LikeSearchParamDto search);

}

## LikesRepository.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Data;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

using MSC.Core.Dtos.Pagination;

using MSC.Core.Enums;

using MSC.Core.Extensions;

namespace MSC.Core.Repositories;

public class LikesRepository : ILikesRepository

{

    private readonly DataContext \_context;

    public LikesRepository(DataContext context)

    {

        \_context = context;

    }

    public async Task<UserLike> GetUserLike(int sourceUserId, int targetUserId)

    {

        var like = await \_context.Likes.FindAsync(sourceUserId, targetUserId);

        return like;

    }

    public async Task<AppUser> GetUserWithLikes(int userId)

    {

        var user = await \_context.Users

                                .Include(l => l.LikedUsers)

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

        return user;

    }

    public async Task<IEnumerable<LikeDto>> GetUserLikes(int userId, string predicate)

    {

        var userLikeType = zUserLikeType.None;

        if(predicate == "liked")

            userLikeType = zUserLikeType.Liked;

        else if(predicate == "likedBy")

            userLikeType = zUserLikeType.LikedBy;

        var search = new LikeSearchParamDto(){ UserId = userId, UserLikeType = userLikeType };

        var likes = await GetUserLikesExecute(search).ToListAsync();

        return likes;

    }

    public async Task<PagedList<LikeDto>> GetUserLikes(LikeSearchParamDto search)

    {

        var likes = GetUserLikesExecute(search);

        var users = await PagedList<LikeDto>.CreateAsync(likes, search.PageNumber, search.PageSize);

        return users;

    }

    private IQueryable<LikeDto> GetUserLikesExecute(LikeSearchParamDto search)

    {

        var usersQuery = \_context.Users.OrderBy(u => u.UserName).AsQueryable();

        var likesQuery = \_context.Likes.AsQueryable();

        switch(search.UserLikeType)

        {

            case zUserLikeType.Liked:

                //the logged in user has liked -- SourceUserId and TargetUser

                likesQuery = likesQuery.Where(l => l.SourceUserId == search.UserId);

                usersQuery = likesQuery.Select(l => l.TargetUser);

                break;

            case zUserLikeType.LikedBy:

                //others users that have liked the logged in user -- TargteUserId and SourceUser

                likesQuery = likesQuery.Where(l => l.TargetUserId == search.UserId);

                usersQuery = likesQuery.Select(l => l.SourceUser);

                break;

            default:

                throw new ValidationException($"Unable to GetUserLikes as UserLikeType '{search.UserLikeType.ToString()}' is not known");

                //break;

        }

        var likes = usersQuery.Select(u => new LikeDto{

            UserName = u.UserName,

            DisplayName = u.DisplayName,

            Age = u.DateOfBirth.CalculateAge(),

            PhotoUrl = u.Photos != null && u.Photos.Any(x => x.IsMain) ? u.Photos.FirstOrDefault(x => x.IsMain).Url : string.Empty,

            City = u.City,

            Id = u.Id,

            GuId = u.Guid

        });

        return likes;

    }

}

## Add to Services

### MSC.Core/Extensions/AppServiceExtensions.cs

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

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ILikesRepository, LikesRepository>();

# MSC.Core/Businesslogic/LikeBusinessLogic

## ILikesBusinsessLogic.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

using MSC.Core.Dtos.Pagination;

namespace MSC.Core.BusinessLogic;

public interface ILikesBusinessLogic

{

    //get the user like

    Task<UserLike> GetUserLike(int sourceUserId, int targetUserId);

    Task<AppUser> GetUserWithLikes(int userId);

    //Get UsersLiked and LikedBy

    Task<IEnumerable<LikeDto>> GetUserLikes(int userId, string predicate);

    //Get UsersLiked and LikedBy

    Task<PagedList<LikeDto>> GetUserLikes(LikeSearchParamDto search);

    //Add Like

    Task<BusinessResponse> AddLike(int userId, UserClaimGetDto claims);

}

## LikesBusinessLogic.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

using MSC.Core.Dtos.Pagination;

using MSC.Core.Repositories;

namespace MSC.Core.BusinessLogic;

public class LikesBusinessLogic : ILikesBusinessLogic

{

    private readonly ILikesRepository \_likeRepo;

    private readonly IUserRepository \_userRepo;

    public LikesBusinessLogic(ILikesRepository likeRepo, IUserRepository userRepo)

    {

        \_likeRepo = likeRepo;

        \_userRepo = userRepo;

    }

    public async Task<UserLike> GetUserLike(int sourceUserId, int targetUserId)

    {

        var like = await \_likeRepo.GetUserLike(sourceUserId, targetUserId);

        return like;

    }

    public async Task<AppUser> GetUserWithLikes(int userId)

    {

        var like = await \_likeRepo.GetUserWithLikes(userId);

        return like;

    }

    //Get UsersLiked and LikedBy

    public async Task<BusinessResponse> AddLike(int userId, UserClaimGetDto claims)

    {

        //get source user

        var sourceUser = await \_likeRepo.GetUserWithLikes(claims.Id);

        if(sourceUser == null)

            return new BusinessResponse(HttpStatusCode.NotFound, "Logged in user not found");

        //get likedUser

        var likedUser = await \_userRepo.GetUserAsync(userId, includePhotos: false);

        if(likedUser == null)

           return new BusinessResponse(HttpStatusCode.NotFound, "Liked user not found");

        if(likedUser.UserName == sourceUser.UserName)

            return new BusinessResponse(HttpStatusCode.BadRequest, "You cannot like your self");

        var userLike = await \_likeRepo.GetUserLike(sourceUser.Id, likedUser.Id);

        if(userLike != null)

            return new BusinessResponse(HttpStatusCode.BadRequest, "You already liked this user");

        //save - add to the source user

        userLike = new UserLike(){ SourceUserId = sourceUser.Id, TargetUserId = likedUser.Id };

        if(sourceUser.LikedUsers == null) sourceUser.LikedUsers = new List<UserLike>();

        sourceUser.LikedUsers.Add(userLike);

        if(await \_userRepo.SaveAllAsync())

            return new BusinessResponse(HttpStatusCode.OK);

        return new BusinessResponse(HttpStatusCode.BadRequest, "Unable to add like");

    }

## Add to Services

### MSC.Core/Extensions/AppServiceExtensions.cs

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

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ILikesRepository, LikesRepository>();

        services.AddScoped<ILikesBusinessLogic, LikesBusinessLogic>();

# Controllers

## LikesController.cs

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using MSC.Core.BusinessLogic;

using MSC.Core.Dtos;

using MSC.Core.Dtos.Pagination;

using MSC.Core.Extensions;

namespace MSC.WebApi.Controller;

[Authorize]

public class LikesController : BaseApiController

{

    private readonly IUserBusinessLogic \_userBl;

    private readonly ILikesBusinessLogic \_likesBl;

    public LikesController(IUserBusinessLogic userBl, ILikesBusinessLogic likesBl)

    {

        \_userBl = userBl;

        \_likesBl = likesBl;

    }

    [HttpPost("{likedUserId:int}")]

    public async Task<ActionResult> AddLike(int likedUserId)

    {

        if(likedUserId <= 0)

            return BadRequest("The user to like userId is required");

        //get the claims

        var userClaims = User.GetUserClaims();

        if(userClaims == null || (!userClaims.HasGuid || !userClaims.HasUserName || !userClaims.HasId))

            return BadRequest("User issue");

        var result = await \_likesBl.AddLike(likedUserId, userClaims);

        ActionResult actionResult =  result.HttpStatusCode switch

        {

            HttpStatusCode.OK => Created(),

            HttpStatusCode.BadRequest => BadRequest(result.Message),

            HttpStatusCode.NotFound => NotFound(result.Message),

            \_ => BadRequest("Unable to add like")

        };

        return actionResult;

    }

    [HttpGet]

    public async Task<ActionResult<PagedList<LikeDto>>> GetUserLikes([FromQuery] LikeSearchParamDto search)

    {

        //get the claims

        var userClaims = User.GetUserClaims();

        if(userClaims == null || (!userClaims.HasGuid || !userClaims.HasUserName || !userClaims.HasId))

            return BadRequest("User issue");

        search.UserId = userClaims.Id;

        var users = await \_likesBl.GetUserLikes(search);

        if(users == null)

            return NotFound("No users found");

        //write pagination header

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

        return Ok(users);

    }

}

# Postman

A new collection Site-13 added