Table of Contents

[Basics 2](#_Toc116677617)

[Current Resources with Changes 2](#_Toc116677618)

[New Resources 2](#_Toc116677619)

[/Core/Entities 2](#_Toc116677620)

[UserLike.cs 2](#_Toc116677621)

[AppUser.cs 2](#_Toc116677622)

[/Core/DB 2](#_Toc116677623)

[DataContext.cs 2](#_Toc116677624)

[Migrations 3](#_Toc116677625)

[Update DB 3](#_Toc116677626)

[/Core/Enums 3](#_Toc116677627)

[UserLikeType.cs 3](#_Toc116677628)

[/Core/Dto/Helpers 4](#_Toc116677629)

[BusinessResponse.cs 4](#_Toc116677630)

[LikeParams.cs 4](#_Toc116677631)

[/Core/Dto 4](#_Toc116677632)

[LikeDto.cs 4](#_Toc116677633)

[/Core/Repositories 4](#_Toc116677634)

[Likes Repository 4](#_Toc116677635)

[ILikesRepository.cs 4](#_Toc116677636)

[LikesRepository 5](#_Toc116677637)

[/Core/BusinessLogic 6](#_Toc116677638)

[Likes Business Logic 6](#_Toc116677639)

[ILikesBusinessLogic 6](#_Toc116677640)

[LikesBusinessLogic 6](#_Toc116677641)

[Register Like Repository and Business Logic for IoC 🡺 /Core/Extensions 7](#_Toc116677642)

[ServiceExtensions 7](#_Toc116677643)

[Controllers 7](#_Toc116677644)

[LikesController.cs 7](#_Toc116677645)

# Basics

|  |  |
| --- | --- |
| WorkingFolder | Copy the content of “Site-02-PagingSortingFiltering” in “Site-03-Like Feature” 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/Entities/AppUser
2. /Core/DB
3. /Core/Dto/Helpers/BusinessResponse
4. /Core/Extensions/ServiceExtensions

# New Resources

1. /Core/Entities/UserLike
2. /Core/Dto/LikeDto
3. /Core/Dto/Helpers/LikeParams
4. /Core/Enums
5. /Core/Repositories/LikesRepository
6. Controllers/LikesController

# /Core/Entities

## UserLike.cs

namespace MSC.Api.Core.Entities;

public class UserLike

{

    //fully defining the relationship between AppUser and UserLike

    public AppUser SourceUser { get; set; }

    public int SourceUserId { get; set; }

    //fully defining the relationship between AppUser and UserLike

    public AppUser LikedUser { get; set; }

    public int LikedUserId { get; set; }

}

## AppUser.cs

Add two properties that will be a collection

1. Users liked me
2. Users I liked

    /// <summary>

    /// The users that have liked logged in user

    /// </summary>

    public ICollection<UserLike> UsersLikedMe { get; set; }

    /// <summary>

    /// The users that the logged in user liked

    /// </summary>

    public ICollection<UserLike> UsersILiked { get; set; }

# /Core/DB

## DataContext.cs

Specify the database table name for the UserLike as Likes and also setup the relationships for the fields.

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.DB

{

    public class DataContext : DbContext

    {

        public DataContext(DbContextOptions options) : base(options)

        {

        }

        //AppUser will have a table name of Users

        public DbSet<AppUser> Users { get; set; }

        //UserLike will have a table name of Likes

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

        //give entities some configuration

        protected override void OnModelCreating(ModelBuilder builder)

        {

            base.OnModelCreating(builder);

            //user like configuration

            //key is combination of sourceUserId and LikedUserId

            builder.Entity<UserLike>()

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

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

            builder.Entity<UserLike>()

                    .HasOne(s => s.SourceUser)

                    .WithMany(l => l.UsersILiked)

                    .HasForeignKey(s => s.SourceUserId)

                    .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities. For sql server use DeleteBehavior.NoAction

            ;

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

            builder.Entity<UserLike>()

                    .HasOne(s => s.LikedUser)

                    .WithMany(l => l.UsersLikedMe)

                    .HasForeignKey(s => s.LikedUserId)

                    .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities. For sql server use DeleteBehavior.NoAction

            ;

        }

    }

}

### Migrations

Add to migrations

* dotnet ef migrations add LikeEntityAdded -o Core/DB/Migrations

To remove the migration use

* dotnet ef migrations remove

### Update DB

1. either start the application with dotnet watch run or
2. issue command dotnet ef database update

# /Core/Enums

## UserLikeType.cs

namespace MSC.Api.Core.Enums;

public enum UserLikeType

{

    Liked,

    LikedBy

}

# /Core/Dto/Helpers

## BusinessResponse.cs

using System.Net;

namespace MSC.Api.Core.Dto;

public class BusinessResponse

{

    public BusinessResponse()

    {

    }

    public BusinessResponse(HttpStatusCode httpStatusCode, string message = "")

    {

        HttpStatusCode = httpStatusCode;

        Message = message;

    }

    public System.Net.HttpStatusCode HttpStatusCode { get; set; }

    public string Message { get; set; }

}

## LikeParams.cs

using MSC.Api.Core.Enums;

namespace MSC.Api.Core.Dto.Helpers;

public class LikeParams : PaginationParams

{

    public UserLikeType UserLikeType { get; set; }

    public int UserId { get; set; }

}

# /Core/Dto

## LikeDto.cs

using System;

namespace MSC.Api.Core.Dto;

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; }

}

# /Core/Repositories

## Likes Repository

### ILikesRepository.cs

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Dto.Helpers;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Repositories;

public interface ILikesRepository

{

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

    Task<AppUser> GetUserWithLikes(int userId);

    Task<PageList<LikeDto>> GetUserLikes(LikeParams likeParams);

}

### LikesRepository

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.DB;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Dto.Helpers;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Enums;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Core.Repositories;

public class LikesRepository : ILikesRepository

{

    public readonly DataContext \_context;

    public LikesRepository(DataContext context)

    {

        \_context = context;

    }

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

    {

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

        return like;

    }

    public async Task<AppUser> GetUserWithLikes(int userId)

    {

        var user = await \_context.Users

                                    .Include(x => x.UsersILiked)

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

        return user;

    }

    public async Task<PageList<LikeDto>> GetUserLikes(LikeParams likeParams)

    {

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

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

        switch (likeParams.UserLikeType)

        {

            case UserLikeType.Liked:

                //users liked by the logged in user

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

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

                break;

            case UserLikeType.LikedBy:

                //others users have liked the logged in user

                likesQuery = likesQuery.Where(l => l.LikedUserId == likeParams.UserId);

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

                break;

            default:

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

                //break;

        }

        //project into likeDto

        var likes = usersQuery.Select(user => new LikeDto

        {

            UserName = user.UserName,

            DisplayName = user.DisplayName,

            Age = user.DateOfBirth.CalculateAge(),

            PhotoUrl = user.Photos.FirstOrDefault(u => u.IsMain).Url,

            GuId = user.GuId,

            City = user.City,

            Id = user.Id

        });

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

        return users;

    }

}

# /Core/BusinessLogic

## Likes Business Logic

### ILikesBusinessLogic

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Dto.Helpers;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Enums;

namespace MSC.Api.Core.BusinessLogic;

public interface ILikesBusinessLogic

{

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

    Task<AppUser> GetUserWithLikes(int userId);

    Task<PageList<LikeDto>> GetUserLikes(LikeParams likeParams);

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

}

### LikesBusinessLogic

using System.Collections.Generic;

using System.Net;

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Dto.Helpers;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Repositories;

namespace MSC.Api.Core.BusinessLogic;

public class LikesBusinessLogic : ILikesBusinessLogic

{

    private readonly ILikesRepository \_likesRepo;

    private readonly IUsersRepository \_usersRepo;

    public LikesBusinessLogic(ILikesRepository likesRepo, IUsersRepository usersRepo)

    {

        \_likesRepo = likesRepo;

        \_usersRepo = usersRepo;

    }

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

    {

        var like = await \_likesRepo.GetUserLike(sourceUserId, likedUserId);

        return like;

    }

    public async Task<AppUser> GetUserWithLikes(int userId)

    {

        var user = await \_likesRepo.GetUserWithLikes(userId);

        return user;

    }

    public async Task<PageList<LikeDto>> GetUserLikes(LikeParams likeParams)

    {

        var users = await \_likesRepo.GetUserLikes(likeParams);

        return users;

    }

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

    {

        //get source user

        var sourceUser = await \_usersRepo.GetAppUserAsync(claims.UserId, includePhotos: false);

        if (sourceUser == null)

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

        //get liked user

        var likedUser = await \_usersRepo.GetAppUserAsync(likeId, 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 yourself");

        var userLike = await \_likesRepo.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, LikedUserId = likedUser.Id };

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

        sourceUser.UsersILiked.Add(userLike);

        if (await \_usersRepo.SaveAllAsync())

            return new BusinessResponse(HttpStatusCode.OK);

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

    }

}

# Register Like Repository and Business Logic for IoC 🡺 /Core/Extensions

## ServiceExtensions

#### RegisterRepos

        services.AddScoped<ILikesRepository, LikesRepository>();

        services.AddScoped<ILikesBusinessLogic, LikesBusinessLogic>();

# Controllers

## LikesController.cs

using System.Collections.Generic;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MSC.Api.Core.BusinessLogic;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Dto.Helpers;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Controllers;

[Authorize]

public class LikesController : BaseApiController

{

    private readonly IUsersBusinessLogic \_userBl;

    private readonly ILikesBusinessLogic \_likesBl;

    public LikesController(IUsersBusinessLogic userBl, ILikesBusinessLogic likesBl)

    {

        \_userBl = userBl;

        \_likesBl = likesBl;

    }

    [HttpPost("{likeId:int}/like/{name}")]

    public async Task<ActionResult> AddLike(int likeId, string name)

    {

        if (likeId <= 0)

        {

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

        }

        //get the claims

        var userClaims = User.GetUserClaims();

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

        {

            return BadRequest("User issue");

        }

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

        ActionResult actionResult = BadRequest("Unable to add like");

        if (result != null)

        {

            switch (result.HttpStatusCode)

            {

                case HttpStatusCode.OK:

                    actionResult = Ok();

                    break;

                case HttpStatusCode.BadRequest:

                    actionResult = BadRequest(result.Message);

                    break;

                case HttpStatusCode.NotFound:

                    actionResult = NotFound(result.Message);

                    break;

                default:

                    actionResult = BadRequest("Unable to add like");

                    break;

            }

        }

        return actionResult;

    }

    [HttpGet("user/likes")]

    public async Task<ActionResult<IEnumerable<LikeDto>>> GetUserLikes([FromQuery] LikeParams likeParams)

    {

        //get the claims

        var userClaims = User.GetUserClaims();

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

        {

            return BadRequest("User issue");

        }

        likeParams.UserId = userClaims.UserId;

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

        if (users == null) //dont check !likes.Any(), good with Ok result with empty array

            return NotFound();

        //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);

    }

}