# Project

* Site-20-Api-Ng-PhotoManagement
  + Copied from: Site-19-Api-Ng-UnitOfWork-OptQueries-CofirmService-MemMessage
* 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/PhotoForApprovalDto.cs |  |
| MSC.Core/Repositories/PhotoRepository |  |
| MSC.Core/BusinessLogic/PhotoBusinessLogic |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/DB/Entities/Photo.cs | MSC.WebApi/Controllers/Usercontrollers |
| MSC.Core/DB/Data/SeedData.cs |  |
| MSC.Core/DB/Data/DataContext.cs |  |
| MSC.Core/Dtos/PotoDto.cs |  |
| MSC.Core/Mappers/ManualMapperExtension.cs |  |
| MSC.Core/Repositores/UserRepository |  |
| MSC.Core/Repositories/BusinessLogic |  |
| MSC.Core/DB/UnitOfWork/UnitOfWork |  |
| MSC.Core/Extensions/AppServiceExensions |  |

# Clear Database

* New migrations created
* Clear the users
* dotnet ef database drop
* dotnet ef database update

# MSC.Core/DB

## Entities

### Photo.cs

Add a property IsApproved

using System.ComponentModel.DataAnnotations.Schema;

namespace MSC.Core.DB.Entities;

//Database table will be called Photos

//with photo managment created a dbset for photos

[Table("Photos")]

public class Photo

{

    [Column(Order = 1)]

    public int Id { get; set; }

    [Column(Order = 2)]

    public string Url { get; set; }

    [Column(Order = 3)]

    public bool IsMain { get; set; }

    [Column(Order = 4)]

    public bool IsApproved {get; set;}

    [Column(Order = 5)]

    public string PublicId { get; set; }

    //fully defining the relationship between AppUser and Photos

    //https://learn.microsoft.com/en-us/ef/core/modeling/relationships/one-to-many

    //alternatively can also do OnModelCreating in Data context

    //[Column(Order = 6)]

    public int AppUserId { get; set; }

    public AppUser AppUser { get; set; }

}

## Data

### DataContext.cs

Add db set

    //for photos after adding photo management. Before it was getting created due to use in AppUser

    public DbSet<Photo> Photos {get; set;}

Also add new method UserPhotSetup and call it inside OnModelCreateing.

Add a Query filter to only return approved photos

    protected override void OnModelCreating(ModelBuilder builder)

    {

        base.OnModelCreating(builder);

        //IR\_REFATCOR Due to use of Identity

        CreateUserRole(builder);

        UserLikeSetup(builder);

        UserMessageSetup(builder);

        //after photo managemnt

        UserPhotoSetup(builder);

    }

And new method

    //add with photo management

    private void UserPhotoSetup(ModelBuilder builder)

    {

        //Add a Query filter to only return approved photos

        builder.Entity<Photo>().HasQueryFilter(p => p.IsApproved);

    }

### SeedData.cs

In seed users method make the first photo as approved

        //convert the username to lower case

        foreach(var user in users){

            //with photo managment, make the first photo isApproved

            if(user.Photos != null && user.Photos.Any()){

                user.Photos.First().IsApproved = true;

            }

            user.UserName = user.UserName.ToLowerInvariant();

            await userManager.CreateAsync(user, defaultPassword); //will save as well.

            //add the user to Member Role

            await userManager.AddToRoleAsync(user, SiteIdentityConstants.Role\_Member);

        }

# MSC.Core/Dtos

## PhotoDto.cs [update]

namespace MSC.Core.Dtos;

public class PhotoDto

{

    public int Id { get; set; }

    public string Url { get; set; }

    public bool IsMain { get; set; }

    //Added IsApproved with with PhotoManagement and removed PublicId since this is not being used on the client.

    //public string PublicId { get; set; }

    public bool IsApproved { get; set; }

}

## PhotoForApprovalDto.cs

using System;

namespace MSC.Core.Dtos;

public class PhotoForApprovalDto

{

    public int Id { get; set; }

    public string Url { get; set; }

    public string UserName { get; set; }

    public int UserId { get; set; }

    public Guid UserGuid { get; set; }

    public bool IsApproved { get; set; }

}

# MSC.Core/Mappers

## ManualMapperExtension.cs

    public static PhotoDto ManualMapPhoto(this Photo photo)

    {

        if(photo == null)

            return null;

        var photoDto = new PhotoDto()

        {

            Id = photo.Id,

            Url = photo.Url,

            IsMain = photo.IsMain

            //PublicId = photo.PublicId

        };

        return photoDto;

    }

## AutoMapperProfile.cs

Add mapping Photo => PhotoForApprovalDto

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

        Map\_Photo\_To\_PhotoForApproval();

And then

    private void Map\_Photo\_To\_PhotoForApproval()

    {

        //the rest of the properties will be auto mapped

        CreateMap<Photo, PhotoForApprovalDto>()

                .ForMember(dest => dest.UserGuid, opt => opt.MapFrom(src => src.AppUser.Guid))

                .ForMember(dest => dest.UserName, opt => opt.MapFrom(src => src.AppUser.UserName))

                .ForMember(dest => dest.UserId, opt=> opt.MapFrom(src => src.AppUser.Id))

                ;

    }

# DropDatabase and Add migrations

> dotnet ef database drop

> dotnet ef migrations add PhotoApprovalAdded

> dotnet ef database update

# MSC.Core/Repositories

## UserRepository

### IUserRepository.cs

Add a new property for the 3 methods that gets the users and return the UserDto.

This is to ignore query filter as setup in Datacontext above

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

    //Need to Ignore Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(int id, bool isCurrentUser);

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

    //Need to Ignore Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(string userName, bool isCurrentUser);

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

    //Need to Ignore Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(Guid guid, bool isCurrentUser);

And add a new signature

    Task<AppUser> GetUserByPhotoIdAsync(int photoId);

### UserRepository.cs

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

     //Need to Ignore Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(int id, bool isCurrentUser)

    {

        /\*var user = await \_context.Users

                    .Where(x => x.Id == id)

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();\*/

        var query = \_context.Users

                    .Where(x => x.Id == id)

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsQueryable();

        if(isCurrentUser) query = query.IgnoreQueryFilters();

        var user = await query.FirstOrDefaultAsync();

        return user;

    }

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

    //Need to Ignore Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(string userName, bool isCurrentUser)

    {

        /\*var user = await \_context.Users

                    .Where(x => x.UserName.ToLower() == userName.ToLower())

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();\*/

        var query =  \_context.Users

                    .Where(x => x.UserName.ToLower() == userName.ToLower())

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsQueryable();

        if(isCurrentUser) query = query.IgnoreQueryFilters();

        var user = await query.FirstOrDefaultAsync();

        return user;

    }

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

     //Need to Ignore Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(Guid guid, bool isCurrentUser)

    {

        /\*var user = await \_context.Users

                    .Where(x => x.Guid == guid)

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();\*/

        var query = \_context.Users

                    .Where(x => x.Guid == guid)

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsQueryable();

        if(isCurrentUser) query = query.IgnoreQueryFilters();

        var user = await query.FirstOrDefaultAsync();

        return user;

    }

Add the new method

    public async Task<AppUser> GetUserByPhotoIdAsync(int photoId)

    {

        var user = await \_context.Users

                        .Include(p => p.Photos)

                        .IgnoreQueryFilters()

                        .Where(p => p.Photos.Any(x => x.Id == photoId))

                        .FirstOrDefaultAsync();

        return user;

    }

## PhotoRepository

### IPhotoRepocitory.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

namespace MSC.Core.Repositories;

public interface IPhotoRepository

{

    Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotosAsync();

    Task<Photo> GetPhotoByIdAsync(int id);

    void RemovePhoto(Photo photo);

}

### PhotoRepository.cs

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using AutoMapper;

using AutoMapper.QueryableExtensions;

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Data;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

namespace MSC.Core.Repositories;

public class PhotoRepository : IPhotoRepository

{

    private readonly DataContext \_context;

    private readonly IMapper \_mapper;

    public PhotoRepository(DataContext context, IMapper mapper)

    {

        \_context = context;

        \_mapper = mapper;

    }

    public async Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotosAsync()

    {

        /\*var photos = await \_context.Photos

                            .IgnoreQueryFilters()

                            .Where(p => p.IsApproved == false)

                            .Select(u => new PhotoForApprovalDto{

                                Id = u.Id,

                                UserName = u.AppUser.UserName,

                                UserGuid = u.AppUser.Guid,

                                UserId = u.AppUser.Id,

                                Url = u.Url,

                                IsApproved = u.IsApproved

                            }).ToListAsync();\*/

        var photos = await \_context.Photos

                            .IgnoreQueryFilters()

                            .Where(p => p.IsApproved == false)

                            .ProjectTo<PhotoForApprovalDto>(\_mapper.ConfigurationProvider)

                            .ToListAsync();

        return photos;

    }

    public async Task<Photo> GetPhotoByIdAsync(int id)

    {

        var photo = await \_context.Photos

                            .IgnoreQueryFilters()

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

        return photo;

    }

    public void RemovePhoto(Photo photo)

    {

        \_context.Photos.Remove(photo);

    }

}

# MSC.Core/DB/UnitOfWork

## IUnitOfWork.cs

    ISignalRRepository SignalRRepo {get;}

## UnitOfWork.cs

Add photo repository to UnitOfWork

    private readonly IPhotoRepository \_photoRepo;

    public UnitOfWork(DataContext context,

                        IUserRepository userRepo, ILikesRepository likesRepo,

                        IMessageRepository msgRepo, ISignalRRepository sigrRepo,

                        IPhotoRepository photoRepo)

    {

        \_context = context;

        \_userRepo = userRepo;

        \_likesRepo = likesRepo;

        \_msgRepo = msgRepo;

        \_sigrRepo = sigrRepo;

        \_photoRepo = photoRepo;

    }

And

    public IPhotoRepository PhotoRepo => \_photoRepo;

# MSC.Core/BusinessLogic

## UserBusinessLogic

### IUserBusinessLogic.cs

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

    //for ignoring Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(int id, UserClaimGetDto claims);

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

    //for ignoring Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(string userName, UserClaimGetDto claims);

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

    //for ignoring Query filter for the current user as setup via DataContext

    Task<UserDto> GetUserAMQEAsync(Guid guid, UserClaimGetDto claims);

### UserBusinessLogic.cs

Update the following methods t also pass currentUserFlag

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

    //for ignoring Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(int id, UserClaimGetDto claims)

    {

        var isCurrent = claims != null && claims.Id == id;

        var user = await \_uow.UserRepo.GetUserAMQEAsync(id, isCurrent);

        if(user == null) return null;

        return user;

    }

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

    //for ignoring Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(string userName, UserClaimGetDto claims)

    {

        var isCurrent = claims != null && claims.UserName.ToLower() == userName.ToLower();

        var user = await \_uow.UserRepo.GetUserAMQEAsync(userName, isCurrent);

        if(user == null) return null;

        return user;

    }

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

    //for ignoring Query filter for the current user as setup via DataContext

    public async Task<UserDto> GetUserAMQEAsync(Guid guid, UserClaimGetDto claims)

    {

        var isCurrent = claims != null && claims.Guid == guid;

        var user = await \_uow.UserRepo.GetUserAMQEAsync(guid, isCurrent);

        if(user == null) return null;

        return user;

    }

Do not make the phot as primary when adding a new photo and it is the only one.

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

    {

        var appUser = await \_uow.UserRepo.GetUserRawAsync(claims.UserName, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found");

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

        if(result.Error != null)

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

        //success, build photo entity and save

        var photo = new Photo

        {

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

            PublicId = result.PublicId //setup public id

            //cannot make unapproved photo as main

            //IsMain = appUser.Photos == null || !appUser.Photos.Any() //mark it active

        };

        //add the photo

        appUser.Photos.Add(photo);

        //\_uow.UserRepo.Update(appUser);

        if(await \_uow.SaveChangesAsync())

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

        return null;

    }

Allow non-approved photos to be deleted as well

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

    {

        var appUser = await \_uow.UserRepo.GetUserRawAsync(claims.UserName, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found");

        //user should be able to delete non approved photos

        //above user method will not be able to pull non approved photos due to

        //queryFilter is applied via DataContext.UserPhotoSetup

        var photo = await \_uow.PhotoRepo.GetPhotoByIdAsync(photoId);

        if(photo == null)

            return new BusinessResponse(HttpStatusCode.NotFound, "Photo Not found", null);

        if(photo.IsMain)

            return new BusinessResponse(HttpStatusCode.BadRequest, "You cannot delete your main photo", null);

        //delete from cludinary

        if(!string.IsNullOrWhiteSpace(photo.PublicId))

        {

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

            if(result.Error != null)

            {

                return new BusinessResponse(HttpStatusCode.BadRequest, result.Error?.Message ?? "Unable to delete photo from service", null);

            }

        }

        //remove from data base

        appUser.Photos.Remove(photo);

        if(await \_uow.SaveChangesAsync())

        {

            return new BusinessResponse(HttpStatusCode.OK, "Photo removed successfully", null);

        }

        return new BusinessResponse(HttpStatusCode.BadRequest, "Unable to delete photo", null);

    }

## PhotoBusinessLogic

### IPhotoBusinessLogic.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

namespace MSC.Core.BusinessLogic;

public interface IPhotoBusinessLogic

{

    Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotosAsync();

    Task<Photo> GetPhotoByIdAsync(int id);

    Task<BusinessResponse> ApprovePhotoAsync(int photoId);

    Task<BusinessResponse> RemovePhotoAsync(int photoId);

}

# MSC.Core/Extensions

## AppServiceExtensons.cs

### AddServices

Add to service for DI => Photo business logic and repo

        services.AddScoped<IPhotoRepository, PhotoRepository>();

        services.AddScoped<IPhotoBusinessLogic, PhotoBusinessLogic>();

# MSC.WebApi/Controllers

## UsersControllers.cs

    [HttpGet("{id}", Name = "GetUserById")] // /api/users/2

    public async Task<ActionResult<UserDto>> GetUser(int id)

    {

        //var user = await \_userBusinessLogic.GetUserAsync(id);

        var user = await \_userBusinessLogic.GetUserAMQEAsync(id, User.GetUserClaims());

        if (user == null)

        {

            return NotFound($"No user found by id {id}");

        }

        return Ok(user);

    }

    [HttpGet("{userName}/name", Name = "GetUserByName")] // /api/users/Bob/name

    public async Task<ActionResult<UserDto>> GetUser(string userName)

    {

        //var user = await \_userBusinessLogic.GetUserAsync(userName);

        var user = await \_userBusinessLogic.GetUserAMQEAsync(userName, User.GetUserClaims());

        if (user == null)

        {

            return NotFound($"No user found by name {userName}");

        }

        return Ok(user);

    }

    [HttpGet("{guid}/guid", Name = "GetUserByGuid")] // /api/users/---/guid

    public async Task<ActionResult<UserDto>> GetUser(Guid guid)

    {

        //var user = await \_userBusinessLogic.GetUserAsync(guid);

        var user = await \_userBusinessLogic.GetUserAMQEAsync(guid, User.GetUserClaims());

        if (user == null)

        {

            return NotFound($"No user found by guid {guid}");

        }

        return Ok(user);

    }

## AdminController.cs

Inject photoBusinesslogic in constructor

    private readonly IUserBusinessLogic \_userBl;

    private readonly IPhotoBusinessLogic \_photoBl;

    public AdminController(IUserBusinessLogic \_userBl, IPhotoBusinessLogic \_photoBl)

    {

        this.\_userBl = \_userBl;

        this.\_photoBl = \_photoBl;

    }

Complete method GetPhotosForModeration

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_Moderator\_Photos)]

    [HttpGet("photos-to-moderate")]

    public async Task<ActionResult<IEnumerable<PhotoForApprovalDto>>> GetPhotosForModeration()

    {

        var photos = await \_photoBl.GetUnapprovedPhotosAsync();

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

            return NoContent();

        return Ok(photos);

    }

Add methods to Approve and Reject photos

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_Moderator\_Photos)]

    [HttpPut("photo-to-approve/{photoId}")]

    public async Task<ActionResult> ApprovePhoto([FromRoute]int photoId)

    {

        var result = await \_photoBl.ApprovePhotoAsync(photoId);

        ActionResult actionResult = result.HttpStatusCode switch{

            HttpStatusCode.OK => Ok(),

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

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

            \_ => BadRequest("Unable to approve photo")

        };

        return actionResult;

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_Moderator\_Photos)]

    [HttpPut("photo-to-reject/{photoId}")]

    public async Task<ActionResult> RejectPhoto([FromRoute]int photoId)

    {

        var result = await \_photoBl.RemovePhotoAsync(photoId);

        ActionResult actionResult = result.HttpStatusCode switch{

            HttpStatusCode.OK => Ok(),

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

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

            \_ => BadRequest("Unable to reject photo")

        };

        return actionResult;

    }

# Run App

Run app

Register user kevin/Password@1

Login as admin/Password@1

Chang blanca and kevin roles to member and moderator

See site 20 section in file [UserIds And Passwords.txt](../UserIds%20And%20Passwords.txt) in the root.