Table of Contents

[Basics 3](#_Toc120029103)

[Current Resources with Changes 3](#_Toc120029104)

[New Resources 3](#_Toc120029105)

[/Core/Entities 3](#_Toc120029106)

[Photo.cs 3](#_Toc120029107)

[/Core/DB 3](#_Toc120029108)

[DataContext.cs 3](#_Toc120029109)

[Add the photo DBSet 3](#_Toc120029110)

[Add the QueryFilter 3](#_Toc120029111)

[Seed.cs 3](#_Toc120029112)

[Migrations and Update Database 4](#_Toc120029113)

[Migrations 4](#_Toc120029114)

[Update DB 4](#_Toc120029115)

[/Core/Dto 5](#_Toc120029116)

[PhotoDto.cs 5](#_Toc120029117)

[PhotoForApprovalDto.cs 5](#_Toc120029118)

[/Core/Repositories 5](#_Toc120029119)

[UsersRepository 5](#_Toc120029120)

[IUsersRepository.cs 5](#_Toc120029121)

[UsersRepositry.cs 5](#_Toc120029122)

[PhotoRepository 7](#_Toc120029123)

[IPhotoRepository.cs 7](#_Toc120029124)

[PhotoRepository.cs 7](#_Toc120029125)

[Add Repository to UnitOfWork 8](#_Toc120029126)

[/Core/BusinessLogic 8](#_Toc120029127)

[UsersBusinessLogic 8](#_Toc120029128)

[IUsersBusinessLogic.cs 8](#_Toc120029129)

[UsersBusinessLogic.cs 8](#_Toc120029130)

[PhotoBusinessLogic 9](#_Toc120029131)

[IPhotoBusinessLogic.cs 9](#_Toc120029132)

[PhotoBusinessLogic.cs 9](#_Toc120029133)

[Controllers 10](#_Toc120029134)

[BuggyController.cs 10](#_Toc120029135)

[UsersController.cs 10](#_Toc120029136)

[PhotoBusinessLogic 10](#_Toc120029137)

[IPhotoBusinessLogic.cs 10](#_Toc120029138)

[PhotoBusinessLogic.cs 10](#_Toc120029139)

[AdminController.cs 11](#_Toc120029140)

# Basics

|  |  |
| --- | --- |
| WorkingFolder | Copy the content of “Site-07-UnitOfWork FinishingTouces” in “Site-08-PhotoManagement” 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/Photo
2. /Core/DB/DataContext
3. /Core/DB/Seed
4. /Core/Dto/PhotoDto
5. /Core/Repositories/UserRepository
6. /Core/BusinessLogic/UsersBusinessLogic
7. Controllers/BuggyController
8. Controllers/UsersController
9. Controller/AdminController

# New Resources

1. /Core/Dto/PhotoForApprovalDto
2. /Core/Repositories/PhotoRepository
3. /Core/BusinessLogic/PhotoBusinessLogic

# /Core/Entities

## Photo.cs

Add new bool property IsApproved

    public bool IsApproved { get; set; }

# /Core/DB

## DataContext.cs

### Add the photo DBSet

Add dbset for the Photos so that we can query it

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

### Add the QueryFilter

        //add a Query filter to only return approved photos

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

        //keep this at the bottom. UTC date fix per EF github

        builder.ApplyUtcDateTimeConverter();

## Seed.cs

Update the Seed Users so the initial photo is approved for seeded users

        //add password to the users, make username lower case and track users

        foreach (var user in users)

        {

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

            //make the photo IsApproved

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

            //saves the users to the database as well. no need to do SaveChangesAsync

            await userManager.CreateAsync(user, "A1abcd");

            //assign a role to the user as well

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

        }

## Migrations and Update Database

### Migrations

Drop database add to migrations

* dotnet ef database drop
* dotnet ef migrations add PhotoApprovalAdded -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

λ dotnet ef database drop

Build started...

Build succeeded.

The Entity Framework tools version '6.0.6' is older than that of the runtime '6.0.10'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAc1fbw for more information.

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 initialized 'DataContext' using provider 'Microsoft.EntityFrameworkCore.Sqlite:6.0.6' with options: None

Are you sure you want to drop the database 'main' on server 'Core/DB/MySocialConnect.db'? (y/N)

y

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 initialized 'DataContext' using provider 'Microsoft.EntityFrameworkCore.Sqlite:6.0.6' with options: None

Dropping database 'main' on server 'Core/DB/MySocialConnect.db'.

Successfully dropped database 'main'.

C:\001LocTraining\Learning-Examples\Api-Angular2018\2022-Angular13-Udemy-MySocialConnect\Site-08-PhotoManagement\MySocialConnect-API\MSC.Api(main -> origin)

λ dotnet ef migrations add PhotoApprovalAdded -o Core/DB/Migrations

Build started...

Build succeeded.

The Entity Framework tools version '6.0.6' is older than that of the runtime '6.0.10'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAc1fbw for more information.

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 initialized 'DataContext' using provider 'Microsoft.EntityFrameworkCore.Sqlite:6.0.6' with options: None

Done. To undo this action, use 'ef migrations remove'

# /Core/Dto

## PhotoDto.cs

Add new bool property IsApproved

    public bool IsApproved { get; set; }

## PhotoForApprovalDto.cs

using System;

namespace MSC.Api.Core.Dto;

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; }

}

# /Core/Repositories

## UsersRepository

Need to Ignore Query filter for the current user un user methods.

### IUsersRepository.cs

    Task<UserDto> GetUserByGuidAsync(Guid id, bool isCurrentUser);

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

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

Add new to get user b the photo id

    Task<AppUser> GetUserByPhotoId(int photoId);

### UsersRepositry.cs

    public async Task<UserDto> GetUserByGuidAsync(Guid id, bool isCurrentUser)

    {

        //var user = await \_context.Users.FindAsync(id);

        //add photos as eager loading

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

        //return user;

        //using automapper queryable extensions

        /\*

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsSplitQuery()

                    .AsNoTracking()

                    .SingleOrDefaultAsync();

        return user;

        \*/

        //ignore query filter for the current user as it is setup via dbcontext

        var query = \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsQueryable();

        if (isCurrentUser)

            query = query.IgnoreQueryFilters();

        var user = await query.FirstOrDefaultAsync();

        return user;

    }

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

    {

        //var user = await \_context.Users.FindAsync(id);

        //add photos as eager loading

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

        //return user;

        //using automapper queryable extensions

        /\*

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsSplitQuery()

                    .AsNoTracking()

                    .SingleOrDefaultAsync();

        return user;

        \*/

        //ignore query filter for the current user as it is setup via dbcontext

        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;

    }

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

    {

        if (userName == null)

            throw new ValidationException("Invalid userName");

        //add photos as eager loading

        //var user = await \_context.Users.Include(p => p.Photos).SingleOrDefaultAsync(x => x.UserName.ToLower() == userName.ToLower());

        //return user;

        //using automapper queryable extensions

        /\*

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .AsSplitQuery()

                    .AsNoTracking()

                    .SingleOrDefaultAsync();

        return user;

        \*/

        //ignore query filter for the current user as it is setup via dbcontext

        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;

    }

    public async Task<AppUser> GetUserByPhotoId(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

### IPhotoRepository.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Repositories;

public interface IPhotoRepository

{

    Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotos();

    Task<Photo> GetPhotoById(int id);

    void RemovePhoto(Photo photo);

}

### PhotoRepository.cs

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.DB;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Repositories;

public class PhotoRepository : IPhotoRepository

{

    private readonly DataContext \_context;

    public PhotoRepository(DataContext context)

    {

        \_context = context;

    }

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

    {

        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();

        return photos;

    }

    public async Task<Photo> GetPhotoById(int id)

    {

        var photo = await \_context.Photos

                            .IgnoreQueryFilters()

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

        return photo;

    }

    public void RemovePhoto(Photo photo)

    {

        \_context.Photos.Remove(photo);

    }

}

### Add Repository to UnitOfWork

#### /Core/DB/UnitOfWork

##### IUnitOfWork.cs

    IPhotoRepository PhotoRepo { get; }

##### UnitOfWork.cs

    public IPhotoRepository PhotoRepo => new PhotoRepository(\_context);

# /Core/BusinessLogic

## UsersBusinessLogic

Repo is ignoring Query filter for the current user. Pass the appropriate flag.

### IUsersBusinessLogic.cs

    Task<UserDto> GetUserByGuidAsync(Guid id, UserClaimGetDto claims);

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

    Task<UserDto> GetUserAsync(string name, UserClaimGetDto claims);

### UsersBusinessLogic.cs

    public async Task<UserDto> GetUserByGuidAsync(Guid id, UserClaimGetDto claims)

    {

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

        var user = await \_uow.UsersRepo.GetUserByGuidAsync(id, isCurrentUser: isCurrent);

        if (user == null) return null;

        return user;

    }

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

    {

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

        var user = await \_uow.UsersRepo.GetUserAsync(id, isCurrentUser: isCurrent);

        if (user == null) return null;

        return user;

    }

    public async Task<UserDto> GetUserAsync(string name, UserClaimGetDto claims)

    {

        var isCurrent = claims != null && claims.UserName == name;

        var user = await \_uow.UsersRepo.GetUserAsync(name, isCurrentUser: isCurrent);

        if (user == null) return null;

        return user;

    }

Remove the logic in the AddPhoto when adding a photo to automatically set a photo

to main if they do not have a main photo (no unapproved photos should be a users main

photo).

        //success, build photo entity and save

        var photo = new Photo

        {

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

            PublicId = result.PublicId //set public id

            //cannot make unapproved photo a main photo so disabling

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

        };

## PhotoBusinessLogic

### IPhotoBusinessLogic.cs

    Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotos();

    Task<Photo> GetPhotoById(int id);

    Task<BusinessResponse> ApprovePhoto(int photoId);

    Task<BusinessResponse> RemovePhoto(int photoId);

### PhotoBusinessLogic.cs

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

using MSC.Api.Core.DB.UnitOfWork;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Services;

namespace MSC.Api.Core.BusinessLogic;

public class PhotoBusinessLogic : IPhotoBusinessLogic

{

    private readonly IUnitOfWork \_uow;

    private readonly IPhotoService \_photoService;

    public PhotoBusinessLogic(IUnitOfWork uow, IPhotoService photoService)

    {

        \_uow = uow;

        \_photoService = photoService;

    }

    public async Task<Photo> GetPhotoById(int id)

    {

        return await \_uow.PhotoRepo.GetPhotoById(id);

    }

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

    {

        return await \_uow.PhotoRepo.GetUnapprovedPhotos();

    }

    public async Task<BusinessResponse> ApprovePhoto(int photoId)

    {

        var photo = await GetPhotoById(photoId);

        if (photo == null) return new BusinessResponse(HttpStatusCode.NotFound, "Photo not found");

        photo.IsApproved = true;

        //get the user by the photo and if the user has no main photo applied then make the photo being approved as main

        var user = await \_uow.UsersRepo.GetUserByPhotoId(photoId);

        if (user != null && !user.Photos.Any(x => x.IsMain))

            photo.IsMain = true;

        if (await \_uow.Complete())

            return new BusinessResponse(HttpStatusCode.OK, "Photo Approved");

        return new BusinessResponse(HttpStatusCode.BadRequest, "Something went bad and could not approve photo");

    }

    public async Task<BusinessResponse> RemovePhoto(int photoId)

    {

        var photo = await GetPhotoById(photoId);

        if (photo == null) return new BusinessResponse(HttpStatusCode.NotFound, "Photo not found");

        if (photo.PublicId != null)

        {

            //cloudinary

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

            if (result.Result == "ok")

                \_uow.PhotoRepo.RemovePhoto(photo);

        }

        else

        {

            \_uow.PhotoRepo.RemovePhoto(photo);

        }

        if (await \_uow.Complete())

            return new BusinessResponse(HttpStatusCode.OK, "Photo Removed");

        return new BusinessResponse(HttpStatusCode.BadRequest, "Something went bad and could not remove photo");

    }

}

# Controllers

## BuggyController.cs

Update all actions to pass null for to the UsersBusinessLogic methods for the user claims.

## UsersController.cs

Put the following code in the actions that have an issue and then pass the user claims to the UsersBusinessLogic methods where needed.

        var userClaims = User.GetUserClaims();

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

        {

            return BadRequest("User issue");

        }

## PhotoBusinessLogic

### IPhotoBusinessLogic.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.BusinessLogic;

public interface IPhotoBusinessLogic

{

    Task<IEnumerable<PhotoForApprovalDto>> GetUnapprovedPhotos();

    Task<Photo> GetPhotoById(int id);

    Task ApprovePhoto(int photoId);

    Task RemovePhoto(int photoId);

}

### PhotoBusinessLogic.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Api.Core.DB.UnitOfWork;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Services;

namespace MSC.Api.Core.BusinessLogic;

public class PhotoBusinessLogic : IPhotoBusinessLogic

{

    private readonly IUnitOfWork \_uow;

    private readonly IPhotoService \_photoService;

    public PhotoBusinessLogic(IUnitOfWork uow, IPhotoService photoService)

    {

        \_uow = uow;

        \_photoService = photoService;

    }

    public async Task<Photo> GetPhotoById(int id)

    {

        return await \_uow.PhotoRepo.GetPhotoById(id);

    }

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

    {

        return await \_uow.PhotoRepo.GetUnapprovedPhotos();

    }

    public async Task ApprovePhoto(int photoId)

    {

        var photo = await GetPhotoById(photoId);

        photo.IsApproved = true;

        await \_uow.Complete();

    }

    public async Task RemovePhoto(int photoId)

    {

        var photo = await GetPhotoById(photoId);

        if (photo.PublicId != null)

        {

            //cloudinary

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

            if (result.Result == "ok")

                \_uow.PhotoRepo.RemovePhoto(photo);

        }

        else

        {

            \_uow.PhotoRepo.RemovePhoto(photo);

        }

        await \_uow.Complete();

    }

}

## AdminController.cs

Implement the GetPhotosForModeration action

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole)]

    [HttpGet("photos-to-moderate")]

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

    {

        var photos = await \_photoBl.GetUnapprovedPhotos();

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

            return BadRequest("No photoes to moderate");

        return Ok(photos);

    }

Add two new actions to Approve and Reject a photo

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole)]

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

    public async Task<ActionResult> ApprovePhoto(int photoId)

    {

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

        ActionResult actionResult = BadRequest("Unable to approve photo");

        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 approve photo");

                    break;

            }

        }

        return actionResult;

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole)]

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

    public async Task<ActionResult> RejectPhoto(int photoId)

    {

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

        ActionResult actionResult = BadRequest("Unable to reject photo");

        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 reject photo");

                    break;

            }

        }

        return actionResult;

    }