# Project

* Site-10-Api-Ng-upload-photos-cloudinary
  + Copied from: Site-09-Api-Ng-templateform-candeactivate-spinner-intercepter-caching
* 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/Helper/CloudinaryConfig.cs |  |
| MSC.Core/Dtos/BusinessResponse.cs |  |
| MSC.Core/Services/IPhotoService.cs |  |
| MSC.Core/Services/PhotoService.cs |  |
|  |  |
|  |  |
|  |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/Constants/ConfigKeyConstants.cs | Appsettings.json |
| MSC.Core/Extensions/AppServiceExtensions.cs | Controllers/UsersController.cs |
| MSC.Core/BusinessLogic/UserBusinessLogic |  |
|  |  |
|  |  |
|  |  |

# Clear Database

Clear the users

dotnet ef database drop

dotnet ef database update

# Cloudinary

## Account Setup

<https://cloudinary.com/>

Setup the account and this will give the info

Go to dashboard:

Cloud name: dj…

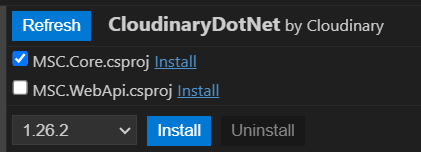
API key: 59…12

API secret: m…0

API environment variable: CLOUDINARY\_URL=cloudinary://59…12:m…0@dj…

## NuGet Gallery

CloudinaryDotNet by Cloudinary



# appsettings.json

Ignoring the tracked file

* remove the file from the project
* add it to .gitignore
* and then commit and pushed the changes
* after that add the appsettings.json to project again.

Add the cloudinary keys to the appsettings.json

"CloudinarySettings":{

    "CloudName": "dj…",

    "ApiKey": "59…12",

    "ApiSecret": "m…0" ,

    "Folder": "MySocialConnect2024"

  }

# MSC.Core/Dtos/Helper

## CloudinaryConfig.cs

using System;

namespace MSC.Core.Helper;

public class CloudinaryConfig

{

    public CloudinaryConfig()

    {

    }

    public CloudinaryConfig(string cloudName, string apiKey, string apiSecret, string folder)

    {

        CloudName = cloudName;

        ApiKey = apiKey;

        ApiSecret = apiSecret;

        Folder = folder;

    }

    public string CloudName { get; set; }

    public string ApiKey { get; set; }

    public string ApiSecret { get; set; }

    public string Folder {get; set;}

    //consolidated property

    public string ApiEnvironmentVariable => $"CLOUDINARY\_URL=cloudinary://{ApiKey}:{ApiSecret}@{CloudName}";

}

## BusinessResponse.cs

namespace MSC.Core.Dtos;

public class BusinessResponse

{

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

    public string Message { get; set; }

}

# /MSC.Coe/Service/PhotoService

## IPhotosService.cs

using System.Threading.Tasks;

using CloudinaryDotNet.Actions;

using Microsoft.AspNetCore.Http;

namespace MSC.Core.Services;

public interface IPhotoService

{

    Task<ImageUploadResult> AddPhotoAsync(IFormFile file);

    Task<DeletionResult> DeletePhotoAync(string publicId);

}

## PhotoService.cs

using System;

using System.Threading.Tasks;

using CloudinaryDotNet;

using CloudinaryDotNet.Actions;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Options;

using MSC.Core.Helper;

namespace MSC.Core.Services;

public class PhotoService : IPhotoService

{

    private readonly Cloudinary \_cloudinary;

    private readonly string \_cloudinaryFolder;

    public PhotoService(IOptions<CloudinaryConfig> cloudinaryConfig)

    {

        //cloudinary account object

        var account = new Account(

            cloudinaryConfig.Value.CloudName,

            cloudinaryConfig.Value.ApiKey,

            cloudinaryConfig.Value.ApiSecret

        );

        \_cloudinary = new Cloudinary(account);

        \_cloudinaryFolder = cloudinaryConfig.Value.Folder;

    }

    public async Task<ImageUploadResult> AddPhotoAsync(IFormFile file)

    {

        var uploadResult = new ImageUploadResult();

        if(file.Length > 0)

        {

            using var stream = file.OpenReadStream();

            var uploadParams = new ImageUploadParams

            {

                File = new FileDescription(file.FileName, stream),

                //transformation for square image

                Transformation = new Transformation().Height(500).Width(500).Crop("fill").Gravity("face"),

                Overwrite = true,

                //uploads to this specified folder and will overwrite due to above

                Folder = \_cloudinaryFolder

            };

            uploadResult = await \_cloudinary.UploadAsync(uploadParams);

        }

        return uploadResult;

    }

    public async Task<DeletionResult> DeletePhotoAync(string publicId)

    {

        var deleteParams = new DeletionParams(publicId);

        var result = await \_cloudinary.DestroyAsync(deleteParams);

        return result;

    }

}

# MSC.Core/Constants/

## ConfigKeyConstants.cs

    public const string CloudinarySettingsKey = "CloudinarySettings";

# MSC.Core/Extensions/

## AppServiceExtensions.cs

Register photo service and configure cloudinary settings

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

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        services.AddScoped<IPhotoService, PhotoService>();

        //services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()); //when have single project/assembly

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

        //adding the Cloudinary to read data from

        //check programs.cs for ref: builder.Services.Configure<EnvConfig>(configuration);

        services.Configure<CloudinaryConfig>(config.GetSection(ConfigKeyConstants.CloudinarySettingsKey));

        return services;

    }

# MSC.Core/BusinessLogic

## IUserBusinessLogic.cs

Add three new methods to handle photos

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

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

    Task<bool> SetPhotoMainAsync(int photoId, UserClaimGetDto claims);

## UserBusinessLogic.cs

Inject the photos service

    private readonly IUserRepository \_userRepo;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

    private readonly IPhotoService \_photoService;

    public UserBusinessLogic(IUserRepository userRepo, ITokenService tokenService, IMapper mapper, IPhotoService photoService)

    {

        \_userRepo = userRepo;

        \_tokenService = tokenService;

        \_mapper = mapper;

        \_photoService = photoService;

    }

Implement the three new methods to handle the photos

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

    {

        var appUser = await \_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

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

        };

        //add the photo

        appUser.Photos.Add(photo);

        //\_userRepo.Update(appUser);

        if(await \_userRepo.SaveAllAsync())

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

        return null;

    }

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

    {

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

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found");

        var response = new BusinessResponse();

        var photo = appUser.Photos?.FirstOrDefault(x => x.Id == photoId);

        if(photo == null)

        {

            response.HttpStatusCode = HttpStatusCode.NotFound;

            response.Message = "Photo not found";

            return response;

        }

        if(photo.IsMain)

        {

            response.HttpStatusCode = HttpStatusCode.BadRequest;

            response.Message = "You cannot delete your main photo";

            return response;

        }

        //delete from cludinary

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

        {

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

            if(result.Error != null)

            {

                response.HttpStatusCode = HttpStatusCode.BadRequest;

                response.Message = result.Error?.Message ?? "Unable to delete photo from service";

                return response;

            }

        }

        //remove from data base

        appUser.Photos.Remove(photo);

        if(await \_userRepo.SaveAllAsync())

        {

            response.HttpStatusCode = HttpStatusCode.OK;

            return response;

        }

        //here it is an error

        response.HttpStatusCode = HttpStatusCode.BadRequest;

        response.Message = "Unable to delete photo";

        return response;

    }

    public async Task<bool> SetPhotoMainAsync(int photoId, UserClaimGetDto claims)

    {

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

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found");

        var photo = appUser.Photos?.FirstOrDefault(x => x.Id == photoId);

        if(photo == null)

            return false;

        if(photo.IsMain)

            throw new DataFailException("This is already your main photo");

        var currentMain = appUser.Photos?.FirstOrDefault( x=> x.IsMain == true);

        if(currentMain != null)

            currentMain.IsMain = false;

        photo.IsMain = true;

        if(await \_userRepo.SaveAllAsync())

            return true;

        return false;

    }

# MSC.WebApi/Controllers

## UsersController.cs

Add the three new end points

    [HttpPost("add/photo")]

    public async Task<ActionResult<PhotoDto>> AddPhoto(IFormFile file)

    {

        //get the user claims

        var userClaims = User.GetUserClaims();

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

            return BadRequest("User Issue");

        var photoDto = await \_userBusinessLogic.AddPhotoAsync(file, userClaims);

        if(photoDto == null)

            return BadRequest("Problem adding photo");

        //this is to tell from where to pick the info. Point to GetUserByGuid and pass the guid to it

        //this will return 401

        //look at the location headers for the for url to the action that gets the user by guid

        return CreatedAtRoute("GetUserByGuid", new {guid = userClaims.Guid.ToString() }, photoDto);

    }

    [HttpDelete("delete/{photoId}/photo")]

    public async Task<ActionResult> DeletePhoto(int photoId)

    {

        //get the user claims

        var userClaims = User.GetUserClaims();

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

            return BadRequest("User Issue");

        var result = await \_userBusinessLogic.DeletePhotoAsync(photoId, userClaims);

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

        if (result != null)

        {

            switch (result.HttpStatusCode)

            {

                case HttpStatusCode.OK:

                    actionResult = Ok();

                    break;

                case HttpStatusCode.BadRequest:

                    actionResult = BadRequest(result.Message ?? "Unable to delete photo");

                    break;

                case HttpStatusCode.NotFound:

                    actionResult = NotFound(result.Message ?? "Photo not found");

                    break;

                default:

                    actionResult = BadRequest("Unable to delete photo");

                    break;

            }

        }

        return actionResult;

    }

    [HttpPut("set/photo/{photoId}/main")]

    public async Task<ActionResult> SetMainPhoto(int photoId)

    {

        //get claims

        var userClaims = User.GetUserClaims();

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

            return BadRequest("User issue");

        var result = await \_userBusinessLogic.SetPhotoMainAsync(photoId, userClaims);

        if (result)

            return NoContent();

        return BadRequest("Unable to set photo to main");

    }

# Postman

Check Site-10 of the post man collection in root Postman\_collection.json.