Table of Contents

[Current Resources with Changes 3](#_Toc115033156)

[New Resources 3](#_Toc115033157)

[/Core/ExceptionsCustom/DataFailException.cs 3](#_Toc115033158)

[/Core/MiddleWare/ExceptionMiddleware.cs 3](#_Toc115033159)

[ExceptionMiddlewware 3](#_Toc115033160)

[/Controller/AccountController.cs 5](#_Toc115033161)

[Respositories 5](#_Toc115033162)

[Users 5](#_Toc115033163)

[IUsersRepository 5](#_Toc115033164)

[UsersRepository 5](#_Toc115033165)

[Photo Storage Options 5](#_Toc115033166)

[Upload Actions 6](#_Toc115033167)

[Cloudinary 6](#_Toc115033168)

[Account Creation 6](#_Toc115033169)

[Configuring the SDK 6](#_Toc115033170)

[Configuring Cloudinary in WebApi 6](#_Toc115033171)

[Installing Package 6](#_Toc115033172)

[Adding keys to AppSettings.json 7](#_Toc115033173)

[Current Keys 7](#_Toc115033174)

[New Keys Added 7](#_Toc115033175)

[/Core/Constants/ConfigKeyConstants 7](#_Toc115033176)

[DTO 7](#_Toc115033177)

[Helpers 7](#_Toc115033178)

[/Core/Dto/Helpers/CloudinaryConfig 7](#_Toc115033179)

[Reading the config 7](#_Toc115033180)

[/Core/Extensions/ConfigExtensions 8](#_Toc115033181)

[StronglyTyped > /Core/Extensions/ServiceExtensions 8](#_Toc115033182)

[RegisterRepos 8](#_Toc115033183)

[Programs.cs 8](#_Toc115033184)

[PhotoService 8](#_Toc115033185)

[/Core/Services/IPhotoService 8](#_Toc115033186)

[/Core/Services/PhotoService 8](#_Toc115033187)

[Register PhotService with IoC 9](#_Toc115033188)

[/Core/Extensions/ServiceExtensions > RegisterRepos 9](#_Toc115033189)

[BusinessLogic 9](#_Toc115033190)

[Users 9](#_Toc115033191)

[IUsersBusinessLogic 9](#_Toc115033192)

[UsersBusinessLogic 9](#_Toc115033193)

[UsersController 10](#_Toc115033194)

[Add Name to Current Get Routes 10](#_Toc115033195)

[By Guid 10](#_Toc115033196)

[By Id 10](#_Toc115033197)

[By Name 10](#_Toc115033198)

[AddPhoto Action 10](#_Toc115033199)

[Testing with Postman 11](#_Toc115033200)

[Result 11](#_Toc115033201)

# Current Resources with Changes

1. Appsettings.json
2. /Core/Constant/ConfigKeyConstants
3. /Core/Extensions/ConfigExtensions
4. /Core/Extensions/ServiceExtensions
5. Program.cs
6. /Core/MiddleWare/ExceptionMiddleware.cs
7. /Core/Repositories/UsersRepository
8. /Core/BusinessLogic/UsersBusinessLogic
9. /Controllers/UsersController

# New Resources

1. Cloudinary
2. /Core/Dto/Helpers/CloudinaryConfig
3. /core/Services/PhotoService
4. /Core/ExceptionsCustom/DataFailException.cs

# /Core/ExceptionsCustom/DataFailException.cs

Create a custom data fail exception which will be used where ever ValidationException could not be used due to name.

using System;

namespace MSC.Api.Core.ExceptionsCustom;

public class DataFailException : Exception

{

    public DataFailException()

    {

    }

    public DataFailException(string message) : base(message)

    {

    }

    public DataFailException(string message, Exception innerException) : base(message, innerException)

    {

    }

}

# /Core/MiddleWare/ExceptionMiddleware.cs

## ExceptionMiddlewware

Update exception middle ware to handle the Validation and unauthorized exceptions. Here is the updated middleware

using System;

using System.ComponentModel.DataAnnotations;

using System.Net;

using System.Text.Json;

using System.Threading.Tasks;

//using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using MSC.Api.Core.Constants;

using MSC.Api.Core.Dto;

using MSC.Api.Core.ExceptionsCustom;

namespace MSC.Api.Core.Middleware;

public class ExceptionMiddleware

{

    private readonly RequestDelegate \_next;

    private readonly ILogger<ExceptionMiddleware> \_logger;

    private readonly IHostEnvironment \_env;

    /// <summary>

    /// Receives RequestDelegate which is whats next in the middle ware pipeline

    /// </summary>

    /// <param name="next">What is next in the pipeline</param>

    /// <param name="logger">So to log the exception</param>

    /// <param name="env">The environment development/production</param>

    public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger, IHostEnvironment env)

    {

        \_next = next;

        \_logger = logger;

        \_env = env;

    }

    /// <summary>

    /// The required method to invoke the middleware

    /// </summary>

    /// <param name="context">The http context</param>

    /// <returns></returns>

    public async Task InvokeAsync(HttpContext context)

    {

        try

        {

            //pass the context to the next piece of middleware

            await \_next(context);

        }

catch (DataFailException dfe)

        {

            \_logger.LogError(dfe, dfe.Message);

            await WriteError(context, dfe.Message, dfe.StackTrace?.ToString(), HttpStatusCode.BadRequest);

        }

        catch (ValidationException vex)

        {

            \_logger.LogError(vex, vex.Message);

            await WriteError(context, vex.Message, vex.StackTrace?.ToString(), HttpStatusCode.BadRequest);

        }

        catch (UnauthorizedAccessException uex)

        {

            \_logger.LogError(uex, uex.Message);

            await WriteError(context, uex.Message, uex.StackTrace?.ToString(), HttpStatusCode.Unauthorized);

        }

        catch (Exception ex)

        {

            \_logger.LogError(ex, ex.Message);

            await WriteError(context, ex, HttpStatusCode.InternalServerError);

        }

    }

    private async Task WriteError(HttpContext context, Exception ex, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //create the response model

        ApiExceptionDto response = null;

        if (\_env.IsDevelopment())

        {

            //development put out the exact message and stack trace

            response = new ApiExceptionDto(context.Response.StatusCode, ex.Message, ex.StackTrace?.ToString());

        }

        else

        {

            //production do not put out the exact message and stack trace

            response = new ApiExceptionDto(context.Response.StatusCode, "Internal Server Error");

        }

        //want the json responses to go as camel case

        var jsonOptions = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

        //serialize the response

        var json = JsonSerializer.Serialize(response, jsonOptions);

        //write

        await context.Response.WriteAsync(json);

    }

    private async Task WriteError(HttpContext context, string exMessage, string exStackTrace, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //write

        await context.Response.WriteAsync(exMessage);

    }

}

## /Controller/AccountController.cs

Remove try/catch from all the actions. Only keep the code.

# Respositories

## Users

### IUsersRepository

#### GetAppUserAsync

Update GetAppUserAsync to receive param to pull photos as well by performing eager loading. By default it should be false

Task<AppUser> GetAppUserAsync(string userName, bool includePhotos = false);

### UsersRepository

#### GetAppUserAsync

   public async Task<AppUser> GetAppUserAsync(string userName, bool includePhotos = false)

    {

        if (userName == null)

            throw new ValidationException("Invalid userName");

        AppUser user = null;

        if (!includePhotos)

            user = await \_context.Users.SingleOrDefaultAsync(x => x.UserName.ToLower() == userName.ToLower());

        else

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

        return user;

    }

# Photo Storage Options

1. In the database
2. On file system – better than database
3. Cloudservice – cost is involved
   1. Cloudinary – a free service is available, no credit card needed.

# Upload Actions

1. Logged in client uploads the photo to API
2. Server uploads the photo to Cloudinary
3. Cloudinary stores photo, sends response
4. API saves photo URL and public ID to DB
5. Saved in DB and given auto generated ID
6. 201 created response sent to client with location header

# Cloudinary

## Account Creation

Setup a free account @ <https://cloudinary.com/users/register_free>

## Configuring the SDK

Get the .net info and also pick the

* Cloudname
* ApiKey
* ApiSecret
* ApiEnvironment

In the webapi add cloudinary by using package

Install-Package CloudinaryDotNet

Check following documentation for .Net

<https://cloudinary.com/documentation/dotnet_integration>

# Configuring Cloudinary in WebApi

## Installing Package

CTRL+SHIFT+P and then open nuget gallery

Select “CloudinaryDotNet by Cloudinary” and click install

Graphical user interface, application

Description automatically generated

## Adding keys to AppSettings.json

### Current Keys

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "AllowedHosts": "\*",

  "AllowSpecificOrigins": ["https://localhost:4200","http://localhost:4200"],

  "TokenKey": "7NCyQkWBsqV3bZsT4qShUN6qzpWUjmRs"

}

### New Keys Added

I am putting in my cloudinarykeys so have excluded it from the source control.

Add following section to it. Make sure to change add in your values for the following 3

  "CloudinarySettings":{

    "CloudName": "dj7i1ncqg",

    "ApiKey": "588574281751962",

    "ApiSecret": "XXXXXXXXXXXXXXXXXXXXXXXXX"

  }

### /Core/Constants/ConfigKeyConstants

Add the name as constant so that it can be later used to retrieve data

    public const string CloudinarySettingsKey = "CloudinarySettings";

# DTO

## Helpers

Create a new helper’s folder and then create

### /Core/Dto/Helpers/CloudinaryConfig

namespace MSC.Api.Core.Dto.Helpers;

public class CloudinaryConfig

{

    public CloudinaryConfig() { }

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

    {

        CloudName = cloudName;

        ApiKey = apiKey;

        ApiSecret = apiSecret;

    }

    public string CloudName { get; set; }

    public string ApiKey { get; set; }

    public string ApiSecret { get; set; }

    //consolidated property

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

}

## /Core/Dto/UserTokenDto

Add a property to return the Main Photo Url

    public string MainPhotoUrl { get; set; }

## Core/Dto/BusinessResponse

This model is used to return httpstatuscode from the business logic

namespace MSC.Api.Core.Dto;

public class BusinessResponse

{

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

    public string Message { get; set; }

}

# Reading the config

We can use the extensions or strongly type it. Preferred is strongly typed

## /Core/Extensions/ConfigExtensions

Add the new method to retrieve the Cloudinary config from AppSettings.json

    public static CloudinaryConfig GetCloudinaryConfig(this IConfiguration config)

    {

        var cloudinary = config.GetSectionValue<CloudinaryConfig>(ConfigKeyConstants.CloudinarySettingsKey, null);

        return cloudinary;

    }

## StronglyTyped > /Core/Extensions/ServiceExtensions

### RegisterRepos

Add towards the top

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

### Programs.cs

Update the following to pass in the configuration as well

builder.Services.RegisterRepos(configuration);

# PhotoService

## /Core/Services/IPhotoService

using System.Threading.Tasks;

using CloudinaryDotNet.Actions;

using Microsoft.AspNetCore.Http;

namespace MSC.Api.Core.Services;

public interface IPhotoService

{

    Task<ImageUploadResult> AddPhotoAsync(IFormFile file);

    Task<DeletionResult> DeletePhotoAync(string publicId);

}

## /Core/Services/PhotoService

using System.Threading.Tasks;

using CloudinaryDotNet;

using CloudinaryDotNet.Actions;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Options;

using MSC.Api.Core.Dto.Helpers;

namespace MSC.Api.Core.Services;

public class PhotoService : IPhotoService

{

    private readonly Cloudinary \_cloudinary;

    //get the CloudinaryConfig that has been strongly typed. check ServiceExtensions

    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);

    }

    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),

                //following transformation will create square image

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

            };

            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;

    }

}

## Register PhotService with IoC

### /Core/Extensions/ServiceExtensions > RegisterRepos

services.AddScoped<IPhotoService, PhotoService>();

# BusinessLogic

## Users

### IUsersBusinessLogic

Add a new definition

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

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

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

### UsersBusinessLogic

#### RegisterAsync Method

Return empty string for the MainPhotoUrl

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            GuId = user.GuId,

            Token = \_tokenService.CreateToken(user),

            MainPhotoUrl = string.Empty

        };

#### LoginAsync Method

Return the MainPhotoUrl

    public async Task<UserTokenDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing"); //exception middleware

        var user = await \_usersRepo.GetAppUserAsync(login.UserName, includePhotos: true);

        if (user == null || user.PasswordSalt == null || user.PasswordHash == null)

            throw new UnauthorizedAccessException("Either username or password is wrong"); //exception middleware

        //password is hashed in db. Hash login password and check against the DB one

        var hashKeyLogin = login.Password.ComputeHashHmacSha512(user.PasswordSalt);

        if (hashKeyLogin == null)

            throw new UnauthorizedAccessException("Either username or password is wrong"); //exception middleware

        //both are byte[]

        if (!hashKeyLogin.Hash.AreEqual(user.PasswordHash))

            throw new UnauthorizedAccessException("Either username or password is wrong"); //exception middleware

        //get the main photo url

        var mainPhotoUrl = user.Photos?.FirstOrDefault(x => x.IsMain)?.Url ?? "";

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            GuId = user.GuId,

            Token = \_tokenService.CreateToken(user),

            MainPhotoUrl = mainPhotoUrl

        };

        return userToken;

    }

#### Add Photo Method

Implement the new method by first injecting the photo service

    private readonly IPhotoService \_photoService;

    public UsersBusinessLogic(IUsersRepository usersRepo, ITokenService tokenService, IPhotoService photoService, IMapper mapper)

    {

        \_tokenService = tokenService;

        \_usersRepo = usersRepo;

        \_photoService = photoService;

        \_mapper = mapper;

    }

And then create the new method.

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

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserName, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

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

        //error

        if (result.Error != null)

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

        //success, build photo entity and save

        var photo = new Photo

        {

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

            PublicId = result.PublicId, //set public id

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

        };

        //add the photo. Photos is an abstract method so cannot be null

        appUser.Photos.Add(photo);

        if (await \_usersRepo.SaveAllAsync())

        {

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

        }

        return null;

    }

#### Set Photo Main Method

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

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserName, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

        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"); //exception middleware

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

        if (currentMain != null)

            currentMain.IsMain = false;

        photo.IsMain = true;

        if (await \_usersRepo.SaveAllAsync())

            return true;

        return false;

    }

#### Delete Photo Method

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

    {

        //get app user with photos

        var appUser = await \_usersRepo.GetAppUserAsync(claims.UserName, includePhotos: true);

        if (appUser == null)

            throw new UnauthorizedAccessException("User not found"); //exception middleware

        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 cloudinary

        if (photo.PublicId != null)

        {

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

            if (result.Error != null)

            {

                response.HttpStatusCode = HttpStatusCode.BadRequest;

                response.Message = result.Error.Message;

                return response;

            }

        }

        //remove from the database as well

        appUser.Photos.Remove(photo);

        if (await \_usersRepo.SaveAllAsync())

        {

            response.HttpStatusCode = HttpStatusCode.OK;

            return response;

        }

        //it is an error then

        response.HttpStatusCode = HttpStatusCode.BadRequest;

        response.Message = "Unable to delete photo";

        return response;

    }

# UsersController

## Add Name to Current Get Routes

Update the by guid, id and name actions and add the name to the route.

### By Guid

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

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

### By Id

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

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

### By Name

    [HttpGet("{name}/name", Name = "GetUserByName")]

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

## AddPhoto Action

1. Add photo will add the photo and then it will return the CreatedAtRoute result.
2. In the CreatedAtRoute it will point to GetUserByGuid route and will pass the guid to it.
3. This will get return as the location header, tells where to view the info
4. AddPhoto will return 201 response back

    [HttpPost("add/photo")]

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

    {

        //get claims

        var userClaims = User.GetUserClaims();

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

        {

            return BadRequest("User issue");

        }

        var photoDto = await \_usersBl.AddPhoto(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

        //lok 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);

    }

### Testing with Postman

|  |  |
| --- | --- |
| Post: <https://localhost:5000/api/users/add/photo>  Authorization Tab: put the Bearer token, login from the front end to get the token  Body tab: select form-data, under the Key enter “File” and under value select a user image. Check Documents folder for some free images picked from internet  Click Send | Graphical user interface, text  Description automatically generated with medium confidence |

#### Result

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Mark Photo Active Action

    [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 \_usersBl.SetPhotoMain(photoId, userClaims);

        if (result)

            return NoContent();

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

    }

## Delete Photo Action

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

    public async Task<ActionResult> DeletePhoto(int photoId)

    {

        //get claims

        var userClaims = User.GetUserClaims();

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

        {

            return BadRequest("User issue");

        }

        var result = await \_usersBl.DeletePhoto(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;

    }