Table of Contents

[Current Resources with Changes 3](#_Toc113815739)

[New Resources Added 3](#_Toc113815740)

[AutoMapper 3](#_Toc113815741)

[Extensions 3](#_Toc113815742)

[/Core/Extensions/DateTimeExtensions 3](#_Toc113815743)

[Entities 3](#_Toc113815744)

[/Core/Entities/Photo 3](#_Toc113815745)

[/Core/Entities/AppUser 4](#_Toc113815746)

[Entity Framework Update - Relationships 4](#_Toc113815747)

[AppUser / Photo Update - Relationship 4](#_Toc113815748)

[Viewing the Database 4](#_Toc113815749)

[Seeding Data 5](#_Toc113815750)

[Using Json File 5](#_Toc113815751)

[UserSeedData.json 5](#_Toc113815752)

[Seed Class 5](#_Toc113815753)

[Program.cs 6](#_Toc113815754)

[Drop the current database 6](#_Toc113815755)

[Run the App to test Seeding Data 7](#_Toc113815756)

[Controllers/UsersController 7](#_Toc113815757)

[Testing With Postman 7](#_Toc113815758)

[Getting Token by Logging In 7](#_Toc113815759)

[Getting User(s) 7](#_Toc113815760)

[Getting All Users 8](#_Toc113815761)

[Getting User Lisa 8](#_Toc113815762)

[Dtos 8](#_Toc113815763)

[Core/Dto/PhotoDto 8](#_Toc113815764)

[Core/Dto/UserDto 8](#_Toc113815765)

[AutoMapperProfiles 9](#_Toc113815766)

[/Core/Dto/AutoMapper/AutoMapperProfile.cs 9](#_Toc113815767)

[Add to ServiceExtensions 9](#_Toc113815768)

[Method RegisterRepos 9](#_Toc113815769)

[Repository / Business Logic Updates 10](#_Toc113815770)

[Users 10](#_Toc113815771)

[Core/Resositories/IUsersRespository.cs 10](#_Toc113815772)

[Core/UsersRepository.cs 10](#_Toc113815773)

[Core/BusinessLogic/IUserBusninessLogic 11](#_Toc113815774)

[Core/BusinessLogic/UserBusinessLogic 11](#_Toc113815775)

[Controllers/UserController 13](#_Toc113815776)

[Repository / Business Logic Updates with AutoMapper Queryable Extensions 14](#_Toc113815777)

[/Core/Entities/AppUser.cs 14](#_Toc113815778)

[/Core/Dto/AutoMapper/AutoMapperProfile.cs 14](#_Toc113815779)

[/Core/Dto/AutoMapper/AutoMapperProfiles.cs 15](#_Toc113815780)

[Core/Repositories 16](#_Toc113815781)

[IUsersReporitory.cs 16](#_Toc113815782)

[UserRepository.cs 16](#_Toc113815783)

[Core/BusinessLogic 18](#_Toc113815784)

[IUsersBusinessLogic.cs 18](#_Toc113815785)

[UsersBusinessLogic.cs 18](#_Toc113815786)

# Current Resources with Changes

1. /Core/Entities/AppUser
2. Program.cs
3. Controllers/UsersController.cs
4. Core/Resositories/IUsersRespository.cs
5. Core/UsersRepository.cs
6. Core/Dto/UserDto
7. Controllers/UsersController.cs

# New Resources Added

1. AutoMapper
2. /Core/Extensions/DateTimeExtensions
3. /Core/Entities/Photo
4. /Core/DB/UserSeedData.json
5. /Core/DB/Seed.cs
6. /Core/Dto/PhotoDto
7. /Core/Dto/AutoMapper

# AutoMapper

Open the new nuget gallery by CTRL+SHIFT+P and then finding and clicking NuGet Gallery

Install AutoMapper.Extensions.Microsoft.DependencyInjection v11.0.0 by Jimmy Bogard [AutoMapper extensions for ASP.NET Core]

Then create a new folder inside root and call it “DtoMappers”. This is where we will create the mappers to map entities to Dto.

# Extensions

## /Core/Extensions/DateTimeExtensions

using System;

namespace MSC.Api.Core.Extensions;

public static class DateTimeExtensions

{

    public static int CalculateAge(this DateTime dob)

    {

        //todays date

        var today = DateTime.Now;

        //calculate the age

        var age = today.Year - dob.Year;

        //go back to the year in which the person was in case of a leap year

        if(dob.Date > today.AddYears(-age))

            age--;

        return age;

    }

}

# Entities

## /Core/Entities/Photo

using System.ComponentModel.DataAnnotations.Schema;

namespace MSC.Api.Core.Entities;

//Database table will be called Photos

[Table("Photos")]

public class Photo

{

    public int Id { get; set; }

    public string Url { get; set; }

    public bool IsMain { get; set; }

    public string PublicId { get; set; }

    //fully defining the relationship between AppUser and Photos

    public AppUser AppUser { get; set; }

    public int AppUserId { get; set; }

}

## /Core/Entities/AppUser

This entity is already available so add the following properties to it

    public DateTime DateOfBirth { get; set; }

    public string DisplayName { get; set; }

    public string Gender { get; set; }

    public string Introduction { get; set; }

    public string LookingFor { get; set; }

    public string Interests { get; set; }

    public string City { get; set; }

    public string Country { get; set; }

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

    public DateTime LastActive { get; set; } = DateTime.Now;

    public DateTime CreatedOn { get; set; } = DateTime.Now;

    public DateTime UpdatedOn { get; set; } = DateTime.Now;

Also add a function which will calculate the age using the extension method with DOB.

    public int GetAge()

    {

        return this.DateOfBirth.CalculateAge();

    }

# Entity Framework Update - Relationships

## AppUser / Photo Update - Relationship

One user can have multiple photos so this will be using One-To-Many relationship.

Photos will be only added when getting the user. We will not individually pull photos.

Since Photo and AppUser classes have the required changes, run migrations to update the table and create the relationship between the two tables.

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

Check the file and then either issue update or remove command

* dotnet ef database update
* dotnet ef migrations remove

Note:

Follow the “[0004 EntityFrameWork Setup Code First - DBContext - Sqlite.docx](0004%20EntityFrameWork%20Setup%20Code%20First%20-%20DBContext%20-%20Sqlite.docx)”, section “Migrations and Database Update”. In command prompt you must be in MSC.Api folder. This is where the project is.

### Viewing the Database

SQLite extension is already installed. Click CTRL+SHIFT+P to open sqlite explorer. Expand it under the Explorer to view

# Seeding Data

## Using Json File

### UserSeedData.json

Move the json file to /Core/DB folder from documents folder – “[0013 UserSeedDataRaw.json](0013%20UserSeedDataRaw.json)”

Graphical user interface, application, chat or text message

Description automatically generated

### Seed Class

In /Core/DB create new class Seed.cs which will work with the above UserSeedData.json.

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.Json;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Core.DB;

public class Seed

{

    public static async Task SeedUsers(DataContext context)

    {

        //if we have users in the table then do not do any thing

        if(await context.Users.AnyAsync()) return;

        //File location

        var file = "Core/DB/UserSeedData.json";

        //check file exists

        var isFile = await Task.Run(() => File.Exists(file));

        if(!isFile) return;

        //read file

        var userData = await File.ReadAllTextAsync(file);

        //make sure that we have user data

        if(string.IsNullOrWhiteSpace(userData)) return;

        //get object from json

        var users = JsonSerializer.Deserialize<List<AppUser>>(userData);

        //check users

        if(users == null || !users.Any()) return;

        //all the users will get the same password so get it here outside the loop

        var hashKey = "password".ComputeHashHmacSha512();

        if(hashKey == null) return;

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

        foreach(var user in users)

        {

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

            user.PasswordHash = hashKey.Hash;

            user.PasswordSalt = hashKey.Salt;

            //we are only adding tracking to the user, save changes will happen outside of the loop

            context.Users.Add(user);

        }

        //add to the database

        await context.SaveChangesAsync();

    }

}

### Program.cs

Add the following using

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using MSC.Api.Core.DB;

And then after line

var app = builder.Build();

Place the following custom section

//CUSTOM: Seed Data Start

using var scope = app.Services.CreateScope();

var services = scope.ServiceProvider;

try

{

    //get the required service

    var context = services.GetRequiredService<DataContext>();

    //asynchronously applies an pending migrations for the context to the database. Will create the database if it doesn't exist already

    //just restarting the application will apply our migrations

    await context.Database.MigrateAsync();

    //now seed the users

    await Seed.SeedUsers(context);

}

catch(Exception ex)

{

    var logger = services.GetRequiredService<ILogger<Program>>();

    logger.LogError(ex, "An error occured during migration");

}

//CUSTOM: Seed Data End

### Drop the current database

Use the following command to drop the current database.

Important do not drop the migrations created so far.

* dotnet ef database drop

|  |
| --- |
| > dotnet ef database drop  Build started...  Build succeeded.  info: Microsoft.EntityFrameworkCore.Infrastructure[10403]  Entity Framework Core 6.0.6 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 6.0.6 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'. |

### Run the App to test Seeding Data

Put the break point in Program.cs on the new code line

Debug and make sure that there is no error.

Go to SQLLite Explorer tab and refresh it

Check the users and the photos tables. Following is the json that got exported when viewing the tables

**Users**: Check the file – “[0013 SeedResult\_Users.json](0013%20SeedResult_Users.json)”

**Photos**: Check the file = “[0013 SeedResult\_Photos.json](0013%20SeedResult_Photos.json)”

Then run the app normal way, duplicate data will not go in

* dotnet watch run

# Controllers/UsersController

Remove the AllowAnonymous attribute from GetUsers method

Remove the Authorize attribute from GetUser

Apply the Authorize attribute to the controller since all methods need login

[Authorize]

public class UsersController : BaseApiController

# Testing With Postman

## Getting Token by Logging In

Method: POST

URL: <https://localhost:5000/api/account/login>

Body as Json:

{

    "username": "lisa",

    "password": "password"

}

Click SEND and look at the result

{

    "userName": "lisa",

    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJsaXNhIiwibmJmIjoxNjYyNzc3Nzg1LCJleHAiOjE2NjMzODI1ODUsImlhdCI6MTY2Mjc3Nzc4NX0.5-uppZJy4zKcQPmYLyuwdb3Bab8SaL-fzbpKjR3\_FTGc7HAl5dJVLElF6vSZPUzCJ89ao-58BMT9VYmQSnJpRw"

}

Copy the token

## Getting User(s)

Go to Authorization tab and select Bearer Token. Then paste the token in the text box.

Graphical user interface, text, application, email

Description automatically generated

### Getting All Users

Method: GET

URL: <https://localhost:5000/api/users>

Click send and you should be able to see all the users

### Getting User Lisa

Method: GET

URL: <https://localhost:5000/api/users/1>

Click send and you should be able to see all the users

# Dtos

## Core/Dto/PhotoDto

Add new dto for the photo entity

namespace MSC.Api.Core.Dto;

public class PhotoDto

{

    public int Id { get; set; }

    public string Url { get; set; }

    public bool IsMain { get; set; }

}

## Core/Dto/UserDto

Add additional fields to the UserDto

public class UserDto

{

    public int Id { 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 Gender { get; set; }

    public string Introduction { get; set; }

    public string LookingFor { get; set; }

    public string Interests { get; set; }

    public string City { get; set; }

    public string Country { get; set; }

    public ICollection<PhotoDto> Photos { get; set; }

    public DateTime LastActive { get; set; }

    public DateTime CreatedOn { get; set; }

    public DateTime UpdatedOn { get; set; }

}

# AutoMapperProfiles

## /Core/Dto/AutoMapper/AutoMapperProfile.cs

using System.Collections.Generic;

using System.Linq;

using AutoMapper;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Dto.AutoMapper;

public class AutoMapperProfiles : Profile

{

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

    }

#region Mappers

    private void Map\_AppUser\_To\_UserDto()

    {

        //same nme propertirs will be automatically mapped

        //Age will also get automatically mapped since source has GetAge, the keywor Age are the same

        //PhotoUrl we'll need to map manually. will pick the url where isMain is true. Do check for null.

        //  \*\*\*Hint: An expression tree lambda may not contain a null propagating operator.

        //  so use a function intead

        CreateMap<AppUser, UserDto>()

        .ForMember(dest => dest.PhotoUrl, opt => opt.MapFrom(src => PickMainUrl\_AppUser\_To\_UserDto(src.Photos)));

    }

    private void Map\_Photo\_To\_PhotoDto()

    {

        CreateMap<Photo, PhotoDto>();

    }

#endregion Mappers

#region Helper Functions

    private string PickMainUrl\_AppUser\_To\_UserDto(ICollection<Photo> photos)

    {

        if (photos == null || !photos.Any()) return string.Empty;

        var url = photos.FirstOrDefault(x => x.IsMain)?.Url ?? string.Empty;

        return url;

    }

#endregion Helper Functions

}

## Add to ServiceExtensions

### Method RegisterRepos

Add the auto mapper to it

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

# Repository / Business Logic Updates

## Users

### Core/Resositories/IUsersRespository.cs

Rename the following methods by right clicking and then selecting Rename Symbol. This will change the name wherever it is being used.

    Task<IEnumerable<AppUser>> GetUsersAsync();

    Task<AppUser> GetUserAsync(int id);

    Task<AppUser> GetUserAsync(string userName);

    Task<AppUser> RegisterAsync(AppUser user);

    Task<bool> UserExistsAsync(string userName);

Also, add following two new

   void Update(AppUser user);

    Task<bool> SaveAllAsync();

### Core/UsersRepository.cs

Rewrite the current methods and create the new.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.DB;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Repositories;

public class UsersRepository : IUsersRepository

{

    private readonly DataContext \_context;

    public UsersRepository(DataContext context)

    {

        \_context = context;

    }

    public async Task<IEnumerable<AppUser>> GetUsersAsync()

    {

        //var users = await \_context.Users.ToListAsync();

        //add photos as eager loading

        var users = await \_context.Users.Include(p => p.Photos).ToListAsync();

        return users;

    }

    public async Task<AppUser> GetUserAsync(int id)

    {

        //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;

    }

    public async Task<AppUser> GetUserAsync(string userName)

    {

        //add photos as eager loading

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

        return user;

    }

    public async Task<AppUser> RegisterAsync(AppUser user)

    {

        if (user == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(user);

        var isSave = await SaveAllAsync();

        if(!isSave)

            throw new ValidationException("User not registerd");

        return user;

    }

    public async Task<bool> UserExistsAsync(string userName)

    {

        if (userName == null)

            throw new ValidationException("Invalid userName");

        var isUser = await \_context.Users.AnyAsync(x => x.UserName.ToLower() == userName.ToLower());

        return isUser;

    }

    //marking the entity only that it has been modified

    public void Update(AppUser user)

    {

        if (user == null)

            throw new ValidationException("Invalid user");

        //ef adds a flag to the entity that it has been modified

        \_context.Entry<AppUser>(user).State = EntityState.Modified;

    }

    public async Task<bool> SaveAllAsync()

    {

        //make sure that the changes have been saved

        var isSave = await \_context.SaveChangesAsync() > 0;

        return isSave;

    }

}

### Core/BusinessLogic/IUserBusninessLogic

Rename the following methods by right clicking and then selecting Rename Symbol. This will change the name ever where it is being used.

   Task<IEnumerable<UserDto>> GetUsersAsync();

    Task<UserDto> GetUserAsync(int id);

    Task<UserDto> GetUserAsync(string name);

    Task<UserTokenDto> RegisterAsync(UserRegisterDto registerUser);

    Task<UserTokenDto> LoginAsync(LoginDto login);

### Core/BusinessLogic/UserBusinessLogic

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using AutoMapper;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Extensions;

using MSC.Api.Core.Repositories;

using MSC.Api.Core.Services;

namespace MSC.Api.Core.BusinessLogic;

public class UsersBusinessLogic : IUsersBusinessLogic

{

    private readonly IUsersRepository \_usersRepo;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

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

    {

        \_tokenService = tokenService;

        \_usersRepo = usersRepo;

        \_mapper = mapper;

    }

    public async Task<IEnumerable<UserDto>> GetUsersAsync()

    {

        var users = await \_usersRepo.GetUsersAsync();

        if (users == null || !users.Any()) return null;

        //var userDto = users.Select(x => new UserDto { Id = x.Id, UserName = x.UserName }).ToList();

        var userDto = \_mapper.Map<IEnumerable<UserDto>>(users);

        return userDto;

    }

    public async Task<UserDto> GetUserAsync(int id)

    {

        var user = await \_usersRepo.GetUserAsync(id);

        if (user == null) return null;

        //var userDto = new UserDto { Id = user.Id, UserName = user.UserName };

        var userDto = \_mapper.Map<UserDto>(user);

        return userDto;

    }

    public async Task<UserDto> GetUserAsync(string name)

    {

        var user = await \_usersRepo.GetUserAsync(name);

        if (user == null) return null;

        //var userDto = new UserDto { Id = user.Id, UserName = user.UserName };

        var userDto = \_mapper.Map<UserDto>(user);

        return userDto;

    }

    public async Task<UserTokenDto> RegisterAsync(UserRegisterDto registerUser)

    {

        if (registerUser == null)

            throw new ValidationException("Invalid user");

        var user = await RegisterUser(registerUser);

        if (user == null || user.Id <= 0)

            throw new ValidationException("Unable to create registration");

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            Token = \_tokenService.CreateToken(user)

        };

        return userToken;

    }

    public async Task<UserTokenDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing");

        var user = await \_usersRepo.GetUserAsync(login.UserName);

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

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

        //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");

        //both are byte[]

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

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

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            Token = \_tokenService.CreateToken(user)

        };

        return userToken;

    }

    private async Task<AppUser> RegisterUser(UserRegisterDto registerUser)

    {

        if (registerUser == null || string.IsNullOrWhiteSpace(registerUser.UserName) || string.IsNullOrWhiteSpace(registerUser.Password))

            throw new ValidationException("User info missing");

        //check user not already taken

        var isUser = await \_usersRepo.UserExistsAsync(registerUser.UserName);

        if (isUser)

            throw new ValidationException("Username already taken");

        //hash the password. it will give back hash and the salt

        var hashKey = registerUser.Password.ComputeHashHmacSha512();

        if (hashKey == null)

            throw new ValidationException("Unable to handle provided password");

        //convert to AppUser to register

        var user = new AppUser { UserName = registerUser.UserName, PasswordHash = hashKey.Hash, PasswordSalt = hashKey.Salt };

        user = await \_usersRepo.RegisterAsync(user);

        return user;

    }

}

# Controllers/UserController

Routes for the following changed

    [HttpGet("{id}/id")]

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

    {

        var user = await \_usersBl.GetUserAsync(id);

        if(user == null)

        {

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

        }

        return Ok(user);

    }

    [HttpGet("{name}/name")]

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

    {

        var user = await \_usersBl.GetUserAsync(name);

        if(user == null)

        {

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

        }

        return Ok(user);

    }

# Repository / Business Logic Updates with AutoMapper Queryable Extensions

Above is vary valid and works.

What we will do is to

* return the DTO from the Repository
* use automapper queryable extensions

## /Core/Entities/AppUser.cs

Open the AppUser entity and delete the GetAge function. With this the query is selecting all the fields. Will move the calculateAge to the AutoMapper profile.

using System;

using System.Collections.Generic;

namespace MSC.Api.Core.Entities;

public class AppUser

{

    public int Id { get; set; }

    public string UserName { get; set; }

    public byte[] PasswordHash { get; set; }

    public byte[] PasswordSalt { get; set; }

    public DateTime DateOfBirth { get; set; }

    public string DisplayName { get; set; }

    public string Gender { get; set; }

    public string Introduction { get; set; }

    public string LookingFor { get; set; }

    public string Interests { get; set; }

    public string City { get; set; }

    public string Country { get; set; }

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

    public DateTime LastActive { get; set; } = DateTime.Now;

    public DateTime CreatedOn { get; set; } = DateTime.Now;

    public DateTime UpdatedOn { get; set; } = DateTime.Now;

}

## /Core/Dto/AutoMapper/AutoMapperProfile.cs

using System.Collections.Generic;

using System.Linq;

using AutoMapper;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Core.Dto.AutoMapper;

public class AutoMapperProfiles : Profile

{

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

    }

#region Mappers

    private void Map\_AppUser\_To\_UserDto()

    {

        //same nme propertirs will be automatically mapped

        //Age will also get automatically mapped since source has GetAge, the keywor Age are the same

        //PhotoUrl we'll need to map manually. will pick the url where isMain is true. Do check for null.

        //  \*\*\*Hint: An expression tree lambda may not contain a null propagating operator.

        //  so use a function intead

        CreateMap<AppUser, UserDto>()

        .ForMember(dest => dest.PhotoUrl, opt => opt.MapFrom(src => PickMainUrl\_AppUser\_To\_UserDto(src.Photos)))

        .ForMember(dest => dest.Age, opt => opt.MapFrom(src => src.DateOfBirth.CalculateAge()));

    }

    private void Map\_Photo\_To\_PhotoDto()

    {

        CreateMap<Photo, PhotoDto>();

    }

#endregion Mappers

#region Helper Functions

    //converted to static method after conversion to using automapper queryable extensions

    private static string PickMainUrl\_AppUser\_To\_UserDto(ICollection<Photo> photos)

    {

        if (photos == null || !photos.Any()) return string.Empty;

        var url = photos.FirstOrDefault(x => x.IsMain)?.Url ?? string.Empty;

        return url;

    }

#endregion Helper Functions

}

## /Core/Dto/AutoMapper/AutoMapperProfiles.cs

Also map the age

using System.Collections.Generic;

using System.Linq;

using AutoMapper;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Core.Dto.AutoMapper;

public class AutoMapperProfiles : Profile

{

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

    }

#region Mappers

    private void Map\_AppUser\_To\_UserDto()

    {

        //same nme propertirs will be automatically mapped

        //Age will also get automatically mapped since source has GetAge, the keywor Age are the same

        //PhotoUrl we'll need to map manually. will pick the url where isMain is true. Do check for null.

        //  \*\*\*Hint: An expression tree lambda may not contain a null propagating operator.

        //  so use a function intead

        CreateMap<AppUser, UserDto>()

        .ForMember(dest => dest.PhotoUrl, opt => opt.MapFrom(src => PickMainUrl\_AppUser\_To\_UserDto(src.Photos)))

        .ForMember(dest => dest.Age, opt => opt.MapFrom(src => src.DateOfBirth.CalculateAge()));

    }

    private void Map\_Photo\_To\_PhotoDto()

    {

        CreateMap<Photo, PhotoDto>();

    }

#endregion Mappers

#region Helper Functions

    private string PickMainUrl\_AppUser\_To\_UserDto(ICollection<Photo> photos)

    {

        if (photos == null || !photos.Any()) return string.Empty;

        var url = photos.FirstOrDefault(x => x.IsMain)?.Url ?? string.Empty;

        return url;

    }

#endregion Helper Functions

}

## Core/Repositories

### IUsersReporitory.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 IUsersRepository

{

    void Update(AppUser user);

    Task<bool> SaveAllAsync();

    Task<IEnumerable<UserDto>> GetUsersAsync();

    Task<UserDto> GetUserAsync(int id);

    Task<UserDto> GetUserAsync(string userName);

    Task<AppUser> GetAppUserAsync(string userName);

    Task<bool> RegisterAsync(AppUser user);

    Task<bool> UserExistsAsync(string userName);

}

### UserRepository.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using AutoMapper;

using AutoMapper.QueryableExtensions;

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 UsersRepository : IUsersRepository

{

    private readonly DataContext \_context;

    private readonly IMapper \_mapper;

    public UsersRepository(DataContext context, IMapper mapper)

    {

        \_context = context;

        \_mapper = mapper;

    }

    public async Task<IEnumerable<UserDto>> GetUsersAsync()

    {

        //var users = await \_context.Users.ToListAsync();

        //add photos as eager loading

        //var users = await \_context.Users.Include(p => p.Photos).ToListAsync();

        //return users;

        //using automapper queryable extensions

        var users = await \_context.Users

                            .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                            .AsSplitQuery()

                            .AsNoTracking()

                            .ToListAsync();

        return users;

    }

    public async Task<UserDto> GetUserAsync(int id)

    {

        //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;

    }

    public async Task<UserDto> GetUserAsync(string userName)

    {

        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;

    }

    public async Task<AppUser> GetAppUserAsync(string userName)

    {

        if (userName == null)

            throw new ValidationException("Invalid userName");

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

        return user;

    }

    public async Task<bool> RegisterAsync(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

        var isSave = await SaveAllAsync();

        return isSave;

    }

    public async Task<bool> UserExistsAsync(string userName)

    {

        var user = await GetAppUserAsync(userName);

        return user != null;

    }

    //marking the entity only that it has been modified

    public void Update(AppUser user)

    {

        if (user == null)

            throw new ValidationException("Invalid user");

        //ef adds a flag to the entity that it has been modified

        \_context.Entry<AppUser>(user).State = EntityState.Modified;

    }

    public async Task<bool> SaveAllAsync()

    {

        //make sure that the changes have been saved

        var isSave = await \_context.SaveChangesAsync() > 0;

        return isSave;

    }

}

## Core/BusinessLogic

### IUsersBusinessLogic.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 IUsersBusinessLogic

{

    Task<IEnumerable<UserDto>> GetUsersAsync();

    Task<UserDto> GetUserAsync(int id);

    Task<UserDto> GetUserAsync(string name);

    Task<UserTokenDto> RegisterAsync(UserRegisterDto registerUser);

    Task<UserTokenDto> LoginAsync(LoginDto login);

}

### UsersBusinessLogic.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

using MSC.Api.Core.Dto;

using MSC.Api.Core.Entities;

using MSC.Api.Core.Extensions;

using MSC.Api.Core.Repositories;

using MSC.Api.Core.Services;

namespace MSC.Api.Core.BusinessLogic;

public class UsersBusinessLogic : IUsersBusinessLogic

{

    private readonly IUsersRepository \_usersRepo;

    private readonly ITokenService \_tokenService;

    public UsersBusinessLogic(IUsersRepository usersRepo, ITokenService tokenService)

    {

        \_tokenService = tokenService;

        \_usersRepo = usersRepo;

    }

    public async Task<IEnumerable<UserDto>> GetUsersAsync()

    {

        var users = await \_usersRepo.GetUsersAsync();

        if (users == null || !users.Any()) return null;

        //var userDto = users.Select(x => new UserDto { Id = x.Id, UserName = x.UserName }).ToList();

        //var userDto = \_mapper.Map<IEnumerable<UserDto>>(users);

        //return userDto;

        return users;

    }

    public async Task<UserDto> GetUserAsync(int id)

    {

        var user = await \_usersRepo.GetUserAsync(id);

        if (user == null) return null;

        //var userDto = new UserDto { Id = user.Id, UserName = user.UserName };

        //var userDto = \_mapper.Map<UserDto>(user);

        //return userDto;

        return user;

    }

    public async Task<UserDto> GetUserAsync(string name)

    {

        var user = await \_usersRepo.GetUserAsync(name);

        if (user == null) return null;

        //var userDto = new UserDto { Id = user.Id, UserName = user.UserName };

        //var userDto = \_mapper.Map<UserDto>(user);

        //return userDto;

        return user;

    }

    public async Task<UserTokenDto> RegisterAsync(UserRegisterDto registerUser)

    {

        if (registerUser == null)

            throw new ValidationException("Invalid user");

        var user = await RegisterUser(registerUser);

        if (user == null || user.Id <= 0)

            throw new ValidationException("Unable to create registration");

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            Token = \_tokenService.CreateToken(user)

        };

        return userToken;

    }

    public async Task<UserTokenDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing");

        var user = await \_usersRepo.GetAppUserAsync(login.UserName);

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

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

        //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");

        //both are byte[]

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

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

        var userToken = new UserTokenDto

        {

            UserName = user.UserName,

            Token = \_tokenService.CreateToken(user)

        };

        return userToken;

    }

    private async Task<AppUser> RegisterUser(UserRegisterDto registerUser)

    {

        if (registerUser == null || string.IsNullOrWhiteSpace(registerUser.UserName) || string.IsNullOrWhiteSpace(registerUser.Password))

            throw new ValidationException("User info missing");

        //check user not already taken

        var isUser = await \_usersRepo.UserExistsAsync(registerUser.UserName);

        if (isUser)

            throw new ValidationException("Username already taken");

        //hash the password. it will give back hash and the salt

        var hashKey = registerUser.Password.ComputeHashHmacSha512();

        if (hashKey == null)

            throw new ValidationException("Unable to handle provided password");

        //convert to AppUser to register

        var user = new AppUser { UserName = registerUser.UserName, PasswordHash = hashKey.Hash, PasswordSalt = hashKey.Salt };

        var isRegister = await \_usersRepo.RegisterAsync(user);

        if(!isRegister)

            throw new ValidationException("User not registerd");

        var returnUser = await \_usersRepo.GetAppUserAsync(user.UserName);

        if(returnUser == null)

            throw new ValidationException("Something went wrong. No user found!");

        return returnUser;

    }

}