# Project

* Site-07-Api-Ng-userentity-datetimeext-relationships-seeddata-automapper
  + Copied from: Site-06-Api-Ng-error-handling-interceptors
* 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/DB/Entities/Photo.cs |  |
| /MSC.Core/Extensions/DateTimeExtensions.cs |  |
| /Msc.Core/Dtos/UserDto.cs |  |
| /Msc.Core/Dtos/PhotoDto.cs |  |
| /Msc.Core/Mappers/ManualMapperExtensions.cs |  |
| /MSC.Core/Mappers/AutoMapperProfiles.cs |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| / MSC.Core/DB/Entities/AppUser.cs |  |
| /MSC.Core/Dtos/LoggedInUserDto.cs |  |
| /MSC.Core/BusinessLogic/UserBusinessLogic.cs |  |
| /MSC.Core/Repositories/IUserRepository.cs |  |
| /MSC.Core/Repositories/UserRepository.cs |  |
| /MSC.Core/Extensions/AppServiceExtensons.cs |  |

# Clear Database

Clear the users

dotnet ef database drop

dotnet ef database update

# /MSC.Core/Extensions

## DateTimeExtensions.cs

using System;

namespace MSC.Core.Extensions;

public static class DateTimeExtensions

{

    public static int CalculateAge(this DateOnly dob)

    {

        //todays date

        var today = DateOnly.FromDateTime(DateTime.UtcNow);

        //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 > today.AddYears(-age))

            age--;

        return age;

    }

}

# MSC.Core/DB

## Entities

### Photo.cs

For a given user we can have many photos.

* fully defining the relationship between AppUser and Photos
* <https://learn.microsoft.com/en-us/ef/core/modeling/relationships/one-to-many>
* alternatively can also do OnModelCreating in Data context

using System.ComponentModel.DataAnnotations.Schema;

namespace MSC.Core.DB.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; }

}

### AppUser.cs

Extending the app user

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.EntityFrameworkCore;

namespace MSC.Core.DB.Entities;

[Index(nameof(Guid))]

[Index(nameof(UserName))]

public class AppUser

{

    public int Id { get; set; }

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    [Required]

    public Guid Guid { get; set; }  = Guid.NewGuid();

    [Required]

    public string UserName { get; set; }

    [Required]

    public byte[] PasswordHash {get; set;} //actual password

    [Required]

    public byte[] PasswordSalt { get; set; } //the salt to hash the password

    public DateOnly 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 List<Photo> Photos { get; set; } = new(); //new List<Photo>();

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

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

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

    public int GetAge(){

        var age = DateOfBirth.CalculateAge();

        return age;

    }

}

## Data

### DataContext.cs

Check the comment for the photos with Users

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Entities;

namespace MSC.Core.DB.Data;

public class DataContext : DbContext

{

    public DataContext(DbContextOptions options) : base(options)

    {

    }

    //Photos will be pulled with the user so no need to put here. Check Photo entity for details

    public DbSet<AppUser> Users { get; set; }

}

# Reset Database

|  |  |
| --- | --- |
| Drop database > dotnet ef database drop  Build started...  Build succeeded.  Are you sure you want to drop the database 'main' on server 'DbFile/MySocialConnect.db'? (y/N)  y  Dropping database 'main' on server 'DbFile/MySocialConnect.db'.  Successfully dropped database 'main'. | Create Migration > dotnet ef migrations add ExtendedUserEntity  Build started...  Build succeeded.  Done. To undo this action, use 'ef migrations remove'  Update Database |
| Update Database > dotnet ef database update |  |

# SeedData

## MSC.Core/DB/Data/UserSeedData.json

Json file added to this location to load the users.

Seed data json created via <https://json-generator.com/>

Following two files are used to generate the random data

* jsongenerator\_female.txt
* jsongenerator\_male.txt

## MSC.Core/DB/Data/SeedData.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Reflection;

using System.Text.Json;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Entities;

using MSC.Core.Extensions;

namespace MSC.Core.DB.Data;

public class SeedData

{

    public static async Task SeedUsers(DataContext context)

    {

        //if users then do not seed new data

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

        //seed data file location

        var location = System.AppDomain.CurrentDomain.BaseDirectory;

        var index = location.IndexOf("MSC.WebApi");

        var file = location.Substring(0, index-1);

        var fileFullPath = @$"{file}\MSC.Core\DB\Data\UserSeedData.json";

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

        if(!isFile) return;

        //read file

        var userData = await File.ReadAllTextAsync(fileFullPath);

        if(string.IsNullOrWhiteSpace(userData)) return;

        //Deserialize

        var options = new JsonSerializerOptions{ PropertyNameCaseInsensitive = true };

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

        if(users == null) return;

        //all the users will get the same password

        var hashkey = "Password@1".ComputeHashHmacSha512();

        if(hashkey == null) return;

        //add the passwors hash, password salt and also convert the username to lower case

        foreach(var user in users){

            user.PasswordHash = hashkey.Hash;

            user.PasswordSalt = hashkey.Salt;

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

            context.Users.Add(user);

        }

        //save to database

        await context.SaveChangesAsync();

    }

}

## Program.cs

Call the seed data routine

app.MapControllers();

/\*\*\*Custom Section Seed Data Start\*\*\*/

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

var services = scope.ServiceProvider;

try{

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

    //Asynchronously applies any pending migrations for the context to the database. Will create the database if it does not already exist.

    await context.Database.MigrateAsync();

    await SeedData.SeedUsers(context);

}

catch(Exception ex)

{

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

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

}

/\*\*\*Custom Section Seed Data End\*\*\*/

app.Run();

# Run the App

This will create DB if dropped and then if no users found will create the default 10 users.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Guid** | **Gender** | **UserName** | **Password** |
| 1 | C0F9087B-5130-4BE3-91DE-854E97404647 | female | mitzi | Password@1 |
| 2 | 9320E9B7-5D36-498E-BCF1-04A41ED29CB7 | female | krista | Password@1 |
| 3 | 77713C6C-19E6-43DD-8064-A0DB4E9A7165 | female | kasey | Password@1 |
| 4 | FDCA8B3A-8164-407F-BC81-657690D5ED92 | female | hollie | Password@1 |
| 5 | CB7961A4-7CD8-4A4B-86E9-34CAC6A6C6F8 | female | blanca | Password@1 |
| 6 | 6A69410A-A688-4CBE-8F2A-A33EBBF5D7E0 | male | mcbride | Password@1 |
| 7 | FC0F5E32-833C-4863-8F1D-07D486621B57 | male | sherman | Password@1 |
| 8 | A8C4C032-7BB5-42C9-AEEB-66559C6E0950 | male | atkinson | Password@1 |
| 9 | 95A5E5D1-AC8E-45A3-9C08-66390EDA2177 | male | holder | Password@1 |
| 10 | 5F90F05A-8305-46B2-8B47-AC2C084CB501 | male | greer | Password@1 |

# MSC.Core/Dtos

## LoggedInUserDto.cs

Added Id and DisplayName properties to it

using System;

namespace MSC.Core.Dtos;

public class LoggedInUserDto

{

    public int Id { get; set; }

    public string UserName { get; set; }

    public Guid Guid { get; set; }

    public string Token { get; set; }

    public string DisplayName {get; set;}

    public string MainPhotoUrl {get; set;}

    public string Gender { get; set; }

}

## PhotoDto

namespace MSC.Core.Dtos;

public class PhotoDto

{

    public int Id { get; set; }

    public string Url { get; set; }

    public bool IsMain { get; set; }

    public string PublicId { get; set; }

}

## UserDto

using System;

using System.Collections.Generic;

namespace MSC.Core.Dtos;

public class UserDto

{

    public int Id { get; set; }

    public Guid GuId { 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 IEnumerable<PhotoDto> Photos { get; set; }

    public DateTime LastActive { get; set; }

    public DateTime CreatedOn { get; set; }

    public DateTime UpdatedOn { get; set; }

}

# Manual Mappers

Created an extension inside Mappers folder for manually mapping AppUser and Photo to dtos

## ManualMapperExtension.cs

using System.Collections.Generic;

using System.Linq;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

using MSC.Core.Services;

namespace MSC.Core.Mappers;

public static class ManualMapperExtensions

{

    public static LoggedInUserDto ManualMapToLoggedInUserDto(this AppUser user, ITokenService tokenService)

    {

        if(user == null)

            return null;

        var loggedInUser = new LoggedInUserDto();

        loggedInUser.Id = user.Id;

        loggedInUser.UserName = user.UserName;

        loggedInUser.Guid = user.Guid;

        loggedInUser.DisplayName = user.DisplayName;

        loggedInUser.Gender = user.Gender;

        loggedInUser.MainPhotoUrl = user.Photos.ManualGetMainPhotoUrl();

        loggedInUser.Token = tokenService.CreateToken(user);

        return loggedInUser;

    }

    public static string ManualGetMainPhotoUrl(this IEnumerable<Photo> photos)

    {

        var mainPhotoUrl = "";

        if(photos != null && photos.Any(x => x.IsMain == true)){

            var main = photos.FirstOrDefault(x => x.IsMain == true);

            if(main != null)

                mainPhotoUrl = main.Url;

        }

        return mainPhotoUrl;

    }

    public static IEnumerable<UserDto> ManualMapUsers(this IEnumerable<AppUser> users)

    {

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

            return null;

        List<UserDto> usersDto = new List<UserDto>();

        foreach(var user in users)

        {

            var userDto = user.ManualMapUser();

            if(userDto != null);

                usersDto.Add(userDto);

        }

        if(!usersDto.Any())

            return null;

        return usersDto;

    }

    public static UserDto ManualMapUser(this AppUser user)

    {

        if(user == null)

            return null;

        var userDto = new UserDto()

        {

            Id = user.Id,

            GuId = user.Guid,

            UserName = user.UserName,

            PhotoUrl = user.Photos.ManualGetMainPhotoUrl(),

            Age = user.GetAge(),

            DisplayName = user.DisplayName,

            Gender = user.Gender,

            Introduction = user.Introduction,

            LookingFor = user.LookingFor,

            Interests = user.Interests,

            City = user.City,

            Country = user.Country,

            LastActive = user.LastActive,

            CreatedOn = user.CreatedOn,

            UpdatedOn = user.UpdatedOn,

            Photos = user.Photos.ManualMapPhotos()

        };

        return userDto;

    }

    public static IEnumerable<PhotoDto> ManualMapPhotos(this IEnumerable<Photo> photos)

    {

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

            return null;

        var photoDtos = new List<PhotoDto>();

        foreach(var photo in photos)

        {

            var photoDto = photo.ManualMapPhoto();

            if(photoDto != null)

                photoDtos.Add(photoDto);

        }

        if(!photoDtos.Any())

            return null;

        return photoDtos;

    }

    public static PhotoDto ManualMapPhoto(this Photo photo)

    {

        if(photo == null)

            return null;

        var photoDto = new PhotoDto()

        {

            Id = photo.Id,

            Url = photo.Url,

            IsMain = photo.IsMain,

            PublicId = photo.PublicId

        };

        return photoDto;

    }

}

# MSC.Core/Repositories

## IUserRepository.cs

Updated signatures

public interface IUserRepository

{

    void Update(AppUser user);

    Task<bool> SaveAllAsync();

    Task<IEnumerable<AppUser>> GetUsersAsync();

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

    Task<AppUser> GetUserAsync(int id);

    Task<AppUser> GetUserAsync(string userName);

    Task<AppUser> GetUserAsync(Guid guid);

    Task<bool> UserExists(string userName);

    Task<bool> RegisterUserAsync(AppUser user);

}

## UserRepository.cs

* Updated method signature above
* adding the Include Photos
* adding the update method

    #region Update and SaveAll

    public void Update(AppUser user)

    {

        \_context.Entry(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;

    }

    #endregion Update and SaveAll

    #region Get Users

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

    {

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

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

        return users;

    }

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

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid user name");

        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;

    }

    public async Task<AppUser> GetUserAsync(int id)

    {

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

        var user = await \_context.Users

                                .Include(p => p.Photos)

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

        return user;

    }

    public async Task<AppUser> GetUserAsync(string userName)

    {

        //var user = await \_context.Users.FirstOrDefaultAsync(x => x.UserName.Equals(userName));

        var user = await \_context.Users

                                .Include(p => p.Photos)

                                .FirstOrDefaultAsync(x => x.UserName.ToLower() == userName.ToLower());

        return user;

    }

    public async Task<AppUser> GetUserAsync(Guid guid)

    {

        //var user = await \_context.Users.FirstOrDefaultAsync(x => x.Guid.ToString() == guid.ToString());

        //var user = await \_context.Users.FirstOrDefaultAsync(x => x.Guid == guid);

        var user = await \_context.Users

                                .Include(p => p.Photos)

                                .FirstOrDefaultAsync(x => x.Guid == guid);

        return user;

    }

    #endregion Get Users

    #region Register

    public async Task<bool> UserExists(string userName)

    {

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

    }

    public async Task<bool> RegisterUserAsync(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

        var isSave = await SaveAllAsync();

        return isSave;

    }

    #endregion Register - Update - SaveAll

# MSC.Core/BusinessLogic

## IUserBusiessLogic.cs

public interface IUserBusinessLogic

{

    Task<IEnumerable<UserDto>> GetUsersAsync();

    Task<AppUser> GetUserRawAsync(string userName);

    Task<UserDto> GetUserAsync(int id);

    Task<UserDto> GetUserAsync(string userName);

    Task<UserDto> GetUserAsync(Guid guid);

    Task<bool> UserExists(string userName);

    Task<LoggedInUserDto> LoginAsync(LoginDto login);

    Task<LoggedInUserDto> RegisterUserAsync(UserRegisterDto registerUser);

}

## UserBusinessLogic.cs

Passing back now DTOs along with using manual mapper

public class UserBusinessLogic : IUserBusinessLogic

{

    private readonly IUserRepository \_userRepo;

    private readonly ITokenService \_tokenService;

    public UserBusinessLogic(IUserRepository userRepo, ITokenService tokenService)

    {

        \_userRepo = userRepo;

        \_tokenService = tokenService;

    }

    #region Get Users

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

    {

        //return IEnumerable AppUser

        var users = await \_userRepo.GetUsersAsync();

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

            return null;

        //return users;

        //tie AppUser to the UserDto

        var usersDto = users.ManualMapUsers();

        return usersDto;

    }

    public async Task<AppUser> GetUserRawAsync(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("User name missing");

        var user = await \_userRepo.GetUserRawAsync(userName);

        return user;

    }

    public async Task<UserDto> GetUserAsync(int id)

    {

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

        //return user;

        if(user == null)

            return null;

        var userDto = user.ManualMapUser();

        return userDto;

    }

    public async Task<UserDto> GetUserAsync(string userName)

    {

        var user = await \_userRepo.GetUserAsync(userName);

        //return user;

        if(user == null)

            return null;

        var userDto = user.ManualMapUser();

        return userDto;

    }

    public async Task<UserDto> GetUserAsync(Guid guid)

    {

        var user = await \_userRepo.GetUserAsync(guid);

        //return user;

        if(user == null)

            return null;

        var userDto = user.ManualMapUser();

        return userDto;

    }

    #endregion Get Users

    #region  Register - Login - UserExists

    public async Task<bool> UserExists(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("User name missing");

        return await \_userRepo.UserExists(userName);

    }

    public async Task<LoggedInUserDto> RegisterUserAsync(UserRegisterDto registerUser)

    {

        if (registerUser == null)

            throw new ValidationException("Invalid user");

        if(string.IsNullOrWhiteSpace(registerUser.UserName))

            throw new ValidationException("User name missing");

        if(string.IsNullOrWhiteSpace(registerUser.Password))

            throw new ValidationException("Password is missing");

        if(await UserExists(registerUser.UserName))

            throw new ValidationException("Username already taken");

        //hash the password, it will give back hash and salt key

        var hashSalt = registerUser.Password.ComputeHashHmacSha512();

        if(hashSalt == null)

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

        //create app user to save

        var appUser = new AppUser();

        appUser.UserName = registerUser.UserName.ToLower();

        appUser.PasswordHash = hashSalt.Hash;

        appUser.PasswordSalt = hashSalt.Salt;

        var isRegister = await \_userRepo.RegisterUserAsync(appUser);

        if(!isRegister)

            throw new DataFailException("User not registerd");

        var returnUser = await \_userRepo.GetUserRawAsync(registerUser.UserName, includePhotos: true);

        if(returnUser == null)

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

        var loggedInUser = returnUser.ManualMapToLoggedInUserDto(\_tokenService);;

        return loggedInUser;

    }

    public async Task<LoggedInUserDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing");

        var user = await \_userRepo.GetUserRawAsync(login.UserName, includePhotos: true);

        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");

        //mapping via manual user mapper

        var loggedInUser = user.ManualMapToLoggedInUserDto(\_tokenService);

        return loggedInUser;

    }

    #endregion Register

}

# Run the App and Test with PostMan

Check potman folder Site-07 for the tests

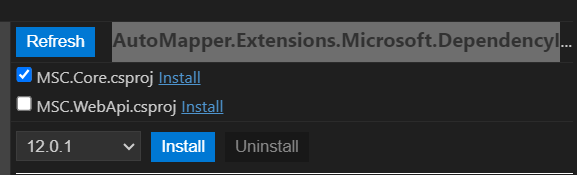
# AutoMapper

## Add AutoMapper

Open nuget and search for automapper

Open Nuget and search for automapper

Select AutoMapper.Extensions.Microsoft.DependencyInjection by Jimmy Bogard : AutoMapper extensions for ASP.NET Core



## Create AutoMapper Profile

### MSC.Core/Mappers/AutoMapperProfiles.cs

using System.Collections.Generic;

using System.Linq;

using AutoMapper;

using MSC.Core.DB.Entities;

using MSC.Core.Dtos;

namespace MSC.Core.Mappers;

public class AutoMapperProfiles : Profile

{

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

        Map\_AppUser\_To\_LoggedInUserDto();

        Map\_UserRegisterDto\_To\_AppUser();

    }

    #region Mappers start

    private void Map\_AppUser\_To\_UserDto()

    {

        //same name propertirs will be automatically mapped

        //Age will also get automatically mapped since source has GetAge, the keyword 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(src.Photos)))

        .ForMember(dest => dest.Age, opt => opt.MapFrom(src => src.GetAge()))

        ;

    }

    private void Map\_Photo\_To\_PhotoDto()

    {

        CreateMap<Photo, PhotoDto>();

    }

    private void Map\_AppUser\_To\_LoggedInUserDto()

    {

        CreateMap<AppUser, LoggedInUserDto>()

        .ForMember(dest => dest.MainPhotoUrl, opt => opt.MapFrom(src => PickMainUrl(src.Photos)))

        ;

    }

    private void Map\_UserRegisterDto\_To\_AppUser()

    {

        CreateMap<UserRegisterDto, AppUser>()

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

        ;

    }

    #endregion Mappers end

    #region Helper functions

    private static string PickMainUrl(IEnumerable<Photo> photos)

    {

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

            return string.Empty;

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

        return url;

    }

    #endregion

}

## Add autoMapperProfile via services

### MSC.Core/Extensions/AppServiceExtensions.cs

#### Method AddServices

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

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

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

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

        return services;

    }

## Use AutoMapper – UserBusinessLogic

### Method GetUsersAsync()

        //tie AppUser to the UserDto

        //var usersDto = users.ManualMapUsers();

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

### Method GetUserAsync(int id)

        //var userDto = user.ManualMapUser();

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

### Method GetUserAsync(string username)

//var userDto = user.ManualMapUser();

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

### Method GetUserAsync(Guid guid)

        //var userDto = user.ManualMapUser();

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

### Method RegisterUserAsync

#### Change #1

        //create app user to save

        /\*

        var appUser = new AppUser();

        appUser.UserName = registerUser.UserName.ToLower();

        \*/

        var appUser = \_mapper.Map<AppUser>(registerUser);

#### Change #2

        //var loggedInUser = returnUser.ManualMapToLoggedInUserDto(\_tokenService);;

        var loggedInUser = \_mapper.Map<LoggedInUserDto>(returnUser);

        loggedInUser.Token = \_tokenService.CreateToken(returnUser);

### Method LoginAsync

        //mapping via manual user mapper

        //var loggedInUser = user.ManualMapToLoggedInUserDto(\_tokenService);

        var loggedInUser = \_mapper.Map<LoggedInUserDto>(user);

        loggedInUser.Token = \_tokenService.CreateToken(user);

## Test Changes via Postman

Test changes via postman Site-07 tests

# AutoMapper Queryable Extensions

## MSC.Core/Dtos/

### UserDto.cs

Chane the photos to List

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

## MSC.Core/DB/Entities

### AppUser.cs

Change the photos to List and comment out the GetAge function.

Update every where that is broken due to this change

## /MSC.Core/Repositories

### IUserRepository.cs

Added four new methods to for using auto mapper queryable extensions

    Task<IEnumerable<UserDto>> GetUsersAMQEAsync();

    Task<UserDto> GetUserAMQEAsync(int id);

    Task<UserDto> GetUserAMQEAsync(string userName);

    Task<UserDto> GetUserAMQEAsync(Guid guid);

### UserRepository.cs

Implement the same

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

    {

         var users = await \_context.Users

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .ToListAsync();

        return users;

    }

public async Task<UserDto> GetUserAMQEAsync(int id)

    {

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();

        return user;

    }

    public async Task<UserDto> GetUserAMQEAsync(string userName)

    {

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();

        return user;

    }

    public async Task<UserDto> GetUserAMQEAsync(Guid guid)

    {

        var user = await \_context.Users

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

                    .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)

                    .FirstOrDefaultAsync();

        return user;

    }

## /MSC.Core/BusinessLogic

### IUserBusinessLogic.cs

Create the new 4 methods to call the new 4 queryable extensions methods

    Task<IEnumerable<UserDto>> GetUsersAMQEAsync();

    Task<UserDto> GetUserAMQEAsync(int id);

    Task<UserDto> GetUserAMQEAsync(string userName);

    Task<UserDto> GetUserAMQEAsync(Guid guid);

### UserBusinessLogic.cs

Implement the new 4 methods

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

    {

        var users = await \_userRepo.GetUsersAMQEAsync();

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

        return users;

    }

    public async Task<UserDto> GetUserAMQEAsync(int id)

    {

        var user = await \_userRepo.GetUserAMQEAsync(id);

        if(user == null) return null;

        return user;

    }

    public async Task<UserDto> GetUserAMQEAsync(string userName)

    {

        var user = await \_userRepo.GetUserAMQEAsync(userName);

        if(user == null) return null;

        return user;

    }

    public async Task<UserDto> GetUserAMQEAsync(Guid guid)

    {

        var user = await \_userRepo.GetUserAMQEAsync(guid);

        if(user == null) return null;

        return user;

    }

## MSC.Web/controller

### UserController.cs

Open users controller and start using the new method auto mapper queryable extensions.

    [HttpGet]

    public async Task<ActionResult<IEnumerable<UserDto>>> GetUsers()

    {

        //var users = await \_userBusinessLogic.GetUsersAsync();

        var users = await \_userBusinessLogic.GetUsersAMQEAsync();

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

        {

            return NotFound("No users found!");

        }

        return Ok(users);

    }

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

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

    {

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

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

        if (user == null)

        {

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

        }

        return Ok(user);

    }

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

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

    {

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

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

        if (user == null)

        {

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

        }

        return Ok(user);

    }

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

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

    {

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

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

        if (user == null)

        {

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

        }

        return Ok(user);

    }

## Test Changes via Postman

Test changes via postman Site-07 tests