# Core >> Extensions >> DateTimeExtensions.cs

Create a new Extension Class DataTimeExtensions.cs in the Core/Extensions folder

## CalculateAge

namespace OAuth2.WebApi.Core.Extensions;

public static class DateTimeExtensions

{

    /// <summary>

    /// Calculate Age

    /// </summary>

    /// <param name="dob"></param>

    /// <returns>Age</returns>

    public static int CalculateAge(this DateOnly dob)

    {

        //for more accurate calculation

        //https://stackoverflow.com/questions/3054715/c-sharp-calculate-accurate-age

        //todays date

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

        //calcuate the age

        var age = today.Year - dob.Year;

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

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

            age--;

        return age;

    }

}

# Extending User Entity with Photos

## Core >> Entities >> Photos.cs

* This class will not get a DBSet in the Core/DB/DataContext.cs
* The relationship will be convention based.
* However, will make sure that AppUserId is not null

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace OAuth2.WebApi.Core.Entities;

[Table("Photos")]

public class Photo

{

    public int Id { get; set; }

    [Required]

    public string URL { get; set; }

    public bool IsMain { get; set; } = false;

    public string PublicId { get; set; }

    //fully defining the relationship between AppUser and Photos

    //this way an orphan entry will not be created.

    //if null are allowed then we dont need to do the following

    public int AppUserId { get; set; }

    public AppUser AppUser { get; set; }

}

## Core >> Entities >> AppUser.cs

* Add the new properties to the AppUser.cs
* Also create a function to calculate age
* Photo will be List<Photo> - 1 to many relationship

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using OAuth2.WebApi.Core.Extensions;

namespace OAuth2.WebApi.Core.Entities;

public class AppUser

{

    /// <summary>

    /// Due to conventions don't need to put [Key] on it since the property name is convention based.

    /// </summary>

    [Key]

    public int Id { get; set; }

    /// <summary>

    /// Auto generation like this is not happening. So check the Core/DB/DataContext.cs for more details

    /// </summary>

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    [Required]

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

    [Required]

    public string UserName { get; set; }

    [Required]

    public byte[] PasswordHash { get; set; }

    [Required]

    public byte[] PasswordSalt { get; set; }

    [Required]

    public DateOnly DateOfBirth { get; set; }

    [Required]

    public string DisplayName { get; set; }

    [Required]

    public string Gender { get; set; }

    public string Introduction { get; set; }

    public string LookingFor { get; set; }

    public string Interests { get; set; }

    [Required]

    public string City { get; set; }

    [Required]

    public string Country { get; set; }

    public List<Photo> Photos { get; set; } = new(); //don't need to do new List<Photo>()

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

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

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

    //Calculate users age from DateOfBirth using the extension method

    public int GetAge()

    {

        return DateOfBirth.CalculateAge();

    }

}

## Core >> DB >> DataContext.cs

* Make sure that DisplayName is unique

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.DB;

/// <summary>

/// DataContext Class, add as a service to program.cs

/// </summary>

public class DataContext : DbContext

{

    public DataContext(DbContextOptions options) : base(options)

    {

    }

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

    //we are not putting the DBSet for Photos

    //Photos will be pulled via Users

    //setup a relationships in photos and apply the Table attriute to it

    protected override void OnModelCreating(ModelBuilder modelBuilder)

    {

        //AppUser has guid that needs to be autogenerated for insert

        //This isn't working, in AppUser create the guid

        /\*

        modelBuilder.Entity<AppUser>()

        .Property(x => x.Guid)

        .ValueGeneratedOnAdd();

        \*/

        //make DisplayName Unique

        modelBuilder.Entity<AppUser>(entity =>

        {

            entity.HasIndex(e => e.DisplayName).IsUnique();

        });

    }

}

## Create Migrations / Update Database

|  |  |
| --- | --- |
| >dotnet ef migrations add ExtendedUserEntity  Build started...  Build succeeded.  Done. To undo this action, use 'ef migrations remove'  Delete current data from table  >dotnet ef database update |  |

# Seeding Data

## Seed Data

Check Core >> DB >> UserSeedData.json

## Seed Class

* File has been created in location Core >> DB
* Name of the file has Seed.cs
* It has only one static method that reads the UserSeedData.json and writes to the database.

using System.Text.Json;

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.Entities;

using OAuth2.WebApi.Core.Extensions;

namespace OAuth2.WebApi.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;

        //seed 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;

        //casing issues in seed data

        var options = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };

        //get object from json

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

        //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 = "Pa$$w0rd1".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();

    }

}

## Call the Seed Class on Application Start

### Program.cs

Add the following piece of code in between app.MapControllers() and app.run().

app.MapControllers();

//CUSTOM: Seed Data Start

//this will give access to all th services

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

var services = scope.ServiceProvider;

try

{

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

    //applies any pending migrations for the context to the the database

    //will create the database if it does not already exist

    //just restarting the application will apply our migrations

    await context.Database.MigrateAsync();

    //seed users

    await Seed.SeedUsers(context);

}

catch (Exception ex)

{

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

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

}

//CUSTOM: Seed Data End

app.Run();

## UsersController.cs

Open users control and apply [Authorize] attribute to all end points.

## Drop Database

Drop the database using the following command. The DB will be recreated with seed data when the next time app will run.

>dotnet ef database drop

Build started...

Build succeeded.

Are you sure you want to drop the database 'main' on server 'Core/DB/OAuth2Sample.db'? (y/N)

y

Dropping database 'main' on server 'Core/DB/OAuth2Sample.db'.

Successfully dropped database 'main'.

## Running the App – Build DB and Seed Data

Use either

>dotnet watch --no-hot-reload

Or

>dotnet run

It will create the DB and will seed data.

## Testing With Postman

|  |  |  |
| --- | --- | --- |
| Login as Lisa and pick up the token | Get all users with Lisa’s token | Get user with d 4 with Lisa’s token |
|  |  |  |

# Updates to include Photos

## UserRepository

### GetAppUsersAsync

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

    {

        //var users = \_context.Users.ToList();

        var users = await \_context.Users

                                    .Include(p => p.Photos)

                                    .ToListAsync();

        return users;

    }

### GetAppUserAsync(int id)

    public async Task<AppUser> GetAppUserAsync(int id)

    {

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

        var user = await \_context.Users

                                    .Include(p => p.Photos)

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

        return user;

    }

### GetAppUserAsync(string name)

   public async Task<AppUser> GetAppUserAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

        var user = await \_context.Users

                                    .Include(p => p.Photos)

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

        return user;

    }

### GetAppUserAsync(Guid guid)

    public async Task<AppUser> GetAppUserAsync(Guid guid)

    {

        var user = await \_context.Users

                                    .Include(p => p.Photos)

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

        return user;

    }

## DTO

### Problem

Above updates will throw an error where we will get into looping due to AppUser and Photo entity.

We will solve this with DTOs that UserBusinessLogic will use to return data to the controller.

### Core >> DTO >> PhotoDto

namespace OAuth2.WebApi.Core.Dto;

public class PhotoDto

{

    public int Id { get; set; }

    public string URL { get; set; }

    public bool IsMain { get; set; } = false;

}

### Core >> DTO >> UserDto

namespace OAuth2.WebApi.Core.Dto;

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 ICollection<PhotoDto> Photos { get; set; }

    public DateTime LastActive { get; set; }

    public DateTime CreatedOn { get; set; }

    public DateTime UpdatedOn { get; set; }

}

## AutoMaper

### Install AutoMapper.Extensions.Microsoft.DependencyInjection

Install from NuGet Gallery AutoMapper.Extensions.Microsoft.DependencyInjection by Jimmy Bogard

Check 0001 Project Setup.docx >> Adding Packages sections for more details.

### AutoMapperProfiles.cs

* Create a new folder AutoMapper in the Core folder.
* Create a helper AutoMapperProfiles.cs file in there.

using AutoMapper;

using OAuth2.WebApi.Core.Dto;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.AutoMapper;

//derive from a class provided by AutoMapper called Profile

public class AutoMapperProfiles : Profile

{

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

    }

    private void Map\_AppUser\_To\_UserDto()

    {

        //same name 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.UserName, opt => opt.MapFrom(src => src.UserName.ToTitleCase()))

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

        //No need to do following, AutoMapper will do this by convention since the Source function name is starting with "Get"

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

        ;

    }

    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;

    }

    private void Map\_Photo\_To\_PhotoDto()

    {

        CreateMap<Photo, PhotoDto>();

    }

}

### Register AutoMapper – ServiceExtensions.cs

* Register AutoMapper in the services.
* Go to Core >> Extensions >> ServiceExtensions.cs
* Update method RegisterServices

    public static void RegisterServices(this IServiceCollection services, IConfiguration configuration)

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        //AutoMapper

        //old way

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

        //new way

        services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

    }

## Updates to IUserBusinessLogic To Return UserDto

Updated IUserBusinessLogic.cs

using OAuth2.WebApi.Core.Dto;

namespace OAuth2.WebApi.Core.Data.BusinessLogic;

public interface IUserBusinessLogic

{

    Task<IEnumerable<UserDto>> GetAppUsersAsync();

    Task<UserDto> GetAppUserAsync(int id);

    Task<UserDto> GetAppUserAsync(string userName);

    Task<UserDto> GetAppUserAsync(Guid guid);

    Task<LoginResponseDto> RegisterAsync(UserRegisterDto registerUser);

    Task<LoginResponseDto> LoginAsync(LoginDto loginInfo);

}

## Updates to UserBusiessLogic to Return UserDto

### Inject IMapper

Using statement

using AutoMapper;

Constructor updates

    private readonly IUserRepository \_userRepository;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

    public UserBusinessLogic(IUserRepository userRepository, ITokenService tokenService, IMapper mapper)

    {

        \_userRepository = userRepository;

        \_tokenService = tokenService;

        \_mapper = mapper;

    }

### GetAppUsersAsync

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

    {

        var appUsers = await \_userRepository.GetAppUsersAsync();

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

            return null;

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

        return users;

    }

### GetAppUserAsyncI(int id)

    public async Task<UserDto> GetAppUserAsync(int id)

    {

        var appUser = await \_userRepository.GetAppUserAsync(id);

        if (appUser == null)

            return null;

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

        return user;

    }

### GetAppUserAsyncI(string userName)

    public async Task<UserDto> GetAppUserAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

        var appUser = await \_userRepository.GetAppUserAsync(userName);

        if (appUser == null)

            return null;

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

        return user;

    }

### GetAppUserAsyncI(Guid guid)

    public async Task<UserDto> GetAppUserAsync(Guid guid)

    {

        var appUser = await \_userRepository.GetAppUserAsync(guid);

        if (appUser == null)

            return null;

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

        return user;

    }

## Updates to UserController

### GetUsers

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

    {

        var users = await \_userBL.GetAppUsersAsync();

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

            return NotFound("No users found!");

        return Ok(users);

    }

### GetUser(int id)

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

    {

        var user = await \_userBL.GetAppUserAsync(id);

        if (user == null)

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

        return Ok(user);

    }

### GetUser(string name)

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

    {

        var user = await \_userBL.GetAppUserAsync(name);

        if (user == null)

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

        return Ok(user);

    }

### GetUser(Guid guid)

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

    {

        var user = await \_userBL.GetAppUserAsync(guid);

        if (user == null)

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

        return Ok(user);

    }

## Testing with Postman

Follow [above](#_Testing_With_Postman) to test again with post man

# AutoMapper Queryable Extensions

* What ever implementation has been done so far is good enough to pick data, pass to business logic. Business logic user AutoMapper to convert data to UserDto and returning it to the controller.
* For better optimization, will user AutoMapper Queryable Extensions.
* Repository will return the UserDto to business logic.

## Core >> Entities >> AppUser.cs

Comment the function for CalculatingAge

    //removed with AutoMapperQueryable Extensions and moved to AutoMapperProfiles

    /\*

    //Calculate users age from DateOfBirth using the extension method

    public int GetAge()

    {

        return DateOfBirth.CalculateAge();

    }

    \*/

## Core >> AutoMapper >> AutoMapperProfiles.cs

### Map\_AppUser\_to\_UserDto()

Map age to use the extension since we removed it from the AppUser.cs above

   private void Map\_AppUser\_To\_UserDto()

    {

        //same name 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.UserName, opt => opt.MapFrom(src => src.UserName.ToTitleCase()))

        .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()))

        ;

    }

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

Make this method static

    //made the method static after change to 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;

    }

## IUserRepository

|  |  |
| --- | --- |
| From | To |
| using OAuth2.WebApi.Core.Entities;  namespace OAuth2.WebApi.Core.Data.Repositories;  public interface IUserRepository  {      Task<bool> SaveAllAsync();      void Update(AppUser appUser);      Task<bool> UserExistsAsync(string userName);      Task<bool> RegisterAsync(AppUser appUser);      Task<IEnumerable<AppUser>> GetAppUsersAsync();      Task<AppUser> GetAppUserAsync(int id);      Task<AppUser> GetAppUserAsync(string userName);      Task<AppUser> GetAppUserAsync(Guid guid);  } | using OAuth2.WebApi.Core.Dto;  using OAuth2.WebApi.Core.Entities;  namespace OAuth2.WebApi.Core.Data.Repositories;  public interface IUserRepository  {      Task<bool> SaveAllAsync();      void Update(AppUser appUser);      Task<bool> UserExistsAsync(string userName);      Task<bool> RegisterAsync(AppUser appUser);      Task<AppUser> GetAppUserAsync(string userName, bool includePhotos = false);      Task<IEnumerable<UserDto>> GetUsersAsync();      Task<UserDto> GetUserAsync(int id);      Task<UserDto> GetUserAsync(string userName);      Task<UserDto> GetUserAsync(Guid guid);  } |

## UserRepository

* As from above now returning UserDto for the following methods.
* Original code is commented, and new code is replacing it

### Inject IMapper

using System.ComponentModel.DataAnnotations;

using AutoMapper;

using AutoMapper.QueryableExtensions;

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.DB;

using OAuth2.WebApi.Core.Dto;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Data.Repositories;

public class UserRepository : IUserRepository

{

    private readonly DataContext \_context;

    private readonly IMapper \_mapper;

    public UserRepository(DataContext context, IMapper mapper)

    {

        \_context = context;

        \_mapper = mapper;

    }

### GetAppUserAsync(string username, bool includePhotos = false)

New method

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

    {

        if (userName == null)

            throw new ValidationException("Invalid userName");

        AppUser appUser = null;

        if (!includePhotos)

        {

            appUser = await \_context.Users

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

        }

        else

        {

            appUser = await \_context.Users

                        .Include(p => p.Photos)

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

        }

        return appUser;

    }

### GetUserAsync(int id)

Rename GetAppUserAsync(int id) to GetUserAsync(int id)

|  |  |
| --- | --- |
| From | To |
| public async Task<AppUser> GetAppUserAsync(int id)      {          //var user = \_context.Users.Find(id);          var user = await \_context.Users               .Include(p => p.Photos)               .SingleOrDefaultAsync(x => x.Id == id);          return user;      } | public async Task<UserDto> GetUserAsync(int id)      {          var user = await \_context.Users                 .Where(x => x.Id == id)                 .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)                 .AsSplitQuery()                 .AsNoTracking()                 .SingleOrDefaultAsync();          return user;      } |

### GetUserAsync(string userName)

Rename GetAppUserAsync(string userName) to GetUserAsync(string userName)

|  |  |
| --- | --- |
| From | To |
| public async Task<AppUser> GetAppUserAsync(string userName)      {          if (string.IsNullOrWhiteSpace(userName))              throw new ValidationException("Invalid userName");          var user = await \_context.Users                .Include(p => p.Photos)                .SingleOrDefaultAsync(x => x.UserName == userName.ToLower());          return user;      } | public async Task<UserDto> GetUserAsync(string userName)      {          if (string.IsNullOrWhiteSpace(userName))              throw new ValidationException("Invalid userName");          var user = await \_context.Users                      .Where(x => x.UserName == userName.ToLower())                      .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)                      .AsSplitQuery()                      .AsNoTracking()                      .SingleOrDefaultAsync();          return user;      } |

### GetUserAsync(Guid guid)

Rename GetAppUserAsync(Guid guid) to GetUserAsync(Guid guid)

|  |  |
| --- | --- |
| From | To |
| public async Task<AppUser> GetAppUserAsync(Guid guid)      {          var user = await \_context.Users                      .Include(p => p.Photos)                      .SingleOrDefaultAsync(x => x.Guid == guid);          return user;      } | public async Task<UserDto> GetUserAsync(Guid guid)      {          var user = await \_context.Users                      .Where(x => x.Guid == guid)                      .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)                      .AsSplitQuery()                      .AsNoTracking()                      .SingleOrDefaultAsync();          return user;      } |

### GetUsersAsync()

|  |  |
| --- | --- |
| From | To |
| public async Task<IEnumerable<AppUser>> GetAppUsersAsync()      {          //var users = \_context.Users.ToList();          var users = await \_context.Users                      .Include(p => p.Photos)                      .ToListAsync();          return users;      } | public async Task<IEnumerable<UserDto>> GetUsersAsync()      {          var users = await \_context.Users                              .ProjectTo<UserDto>(\_mapper.ConfigurationProvider)                              .AsSplitQuery()                              .AsNoTracking()                              .ToListAsync();          return users;      } |

## IUserBusinessLogic

|  |  |
| --- | --- |
| From | To |
| using OAuth2.WebApi.Core.Dto;  namespace OAuth2.WebApi.Core.Data.BusinessLogic;  public interface IUserBusinessLogic  {      Task<IEnumerable<UserDto>> GetAppUsersAsync();      Task<UserDto> GetAppUserAsync(int id);      Task<UserDto> GetAppUserAsync(string userName);      Task<UserDto> GetAppUserAsync(Guid guid);      Task<LoginResponseDto> RegisterAsync(UserRegisterDto registerUser);      Task<LoginResponseDto> LoginAsync(LoginDto loginInfo);  } | using OAuth2.WebApi.Core.Dto;  using OAuth2.WebApi.Core.Entities;  namespace OAuth2.WebApi.Core.Data.BusinessLogic;  public interface IUserBusinessLogic  {      Task<IEnumerable<UserDto>> GetUsersAsync();      Task<UserDto> GetUserAsync(int id);      Task<UserDto> GetUserAsync(string userName);      Task<UserDto> GetUserAsync(Guid guid);      Task<LoginResponseDto> RegisterAsync(UserRegisterDto registerUser);      Task<LoginResponseDto> LoginAsync(LoginDto loginInfo);      Task<AppUser> GetAppUserAsync(string userName, bool includePhotos = false);  } |

## UserBusinessLogic

Sine the UsersRepository and IUsersBusinessLogic has changed, adjust the UsersBusinessLogic accordingly

### GetAppUserAsync(string username, bool includePhotos = false)

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

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

        var appUser = await \_userRepository.GetAppUserAsync(userName, includePhotos);

        return appUser;

    }

### GetUsersAsync()

Rename GetAppUsersAsync() to GetUsersAsync()

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

    {

        var appUsers = await \_userRepository.GetUsersAsync();

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

            return null;

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

        return appUsers;

    }

### GetUserAsync(int id)

Rename GetAppUserAsync(int id) to GetUserAsync(int id)

    public async Task<UserDto> GetUserAsync(int id)

    {

        var appUser = await \_userRepository.GetUserAsync(id);

        if (appUser == null)

            return null;

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

        return appUser;

    }

### GetUserAsync(string userName)

Rename GetAppUserAsync(string userName) to GetUserAsync(string userName)

    public async Task<UserDto> GetUserAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

        var appUser = await \_userRepository.GetUserAsync(userName);

        if (appUser == null)

            return null;

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

        return appUser;

    }

### GetUserAsync(Guid guid)

Rename GetAppUserAsync(Guid guid) to GetUserAsync(Guid guid)

    public async Task<UserDto> GetUserAsync(Guid guid)

    {

        var appUser = await \_userRepository.GetUserAsync(guid);

        if (appUser == null)

            return null;

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

        return appUser;

    }

### LoginAsync(LoginDto loginDto)

Use the new repository method GetAppUserAsync(string username, bool includePhotos = false)

    public async Task<LoginResponseDto> LoginAsync(LoginDto loginInfo)

    {

        if (loginInfo == null)

            throw new ValidationException("Login info missing");

        var appUser = await \_userRepository.GetAppUserAsync(loginInfo.UserName, includePhotos: true);

## UsersController

Since the UserBusinessLogic has changed so change the calls in the UsersController accordingly.

### GetUsers()

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

    {

        var users = await \_userBL.GetUsersAsync();

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

            return NotFound("No users found!");

        return Ok(users);

    }

### GetUser(int id)

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

    {

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

        if (user == null)

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

        return Ok(user);

    }

### GetUser(string name)

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

    {

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

        if (user == null)

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

        return Ok(user);

    }

### GetUser(Guid guid)

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

    {

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

        if (user == null)

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

        return Ok(user);

    }