Table of Contents

[Current Resources with Changes 2](#_Toc115736407)

[/Core/Dto 2](#_Toc115736408)

[UserTokenDto 2](#_Toc115736409)

[UserRegisterDto 2](#_Toc115736410)

[/Core/Dto/AutoMapper/AutoMapperProfile.cs 3](#_Toc115736411)

[Business Logic 3](#_Toc115736412)

[UserBusinessLigic.cs 3](#_Toc115736413)

# Current Resources with Changes

1. /Core/Dto/UserToken
2. /Core/Dto/UserRegisterDto
3. /Core/Dto/AutoMapper/AutoMapperProfiles
4. /Core/BusinessLogic/UsersBusinessLogic

# /Core/Dto

## UserTokenDto

Add a new field DisplayName to it

using System;

namespace MSC.Api.Core.Dto;

public class UserTokenDto

{

    public string UserName { get; set; }

    public string DisplayName { get; set; }

    public Guid GuId { get; set; }

    public string Token { get; set; }

    public string MainPhotoUrl { get; set; }

}

## UserRegisterDto

Add the additional properties for the registration.

using System;

using System.ComponentModel.DataAnnotations;

namespace MSC.Api.Core.Dto;

public class UserRegisterDto

{

    [Required(ErrorMessage = "Gender is empty")]

    public string Gender { get; set; }

    [Required(ErrorMessage = "DisplayName is empty")]

    [StringLength(5)]

    public string DisplayName { get; set; }

    [Required(ErrorMessage = "DateOfBirth is empty")]

    public DateTime DateOfBirth { get; set; }

    [Required(ErrorMessage = "City is empty")]

    [RegularExpression("^[a-zA-z]\*$")]

    public string City { get; set; }

    [Required(ErrorMessage = "Country is empty")]

    [RegularExpression("^[a-zA-z]\*$")]

    public string Country { get; set; }

    [Required(ErrorMessage = "UserName is empty")]

    [MinLength(5, ErrorMessage = "UserName length must be atleast 5 chars")]

    public string UserName { get; set; }

    [Required(ErrorMessage = "Password is empty")]

    [StringLength(8, MinimumLength = 4)]

    public string Password { get; set; }

}

# /Core/Dto/AutoMapper/AutoMapperProfile.cs

Create two new methods

    private void Map\_UserRegister\_To\_AppUser()

    {

        CreateMap<UserRegisterDto, AppUser>()

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

        .ForMember(dest => dest.GuId, opt => opt.MapFrom(src => Guid.NewGuid()))

        ;

    }

    private void Map\_AppUser\_To\_UserTokenDto()

    {

        CreateMap<AppUser, UserTokenDto>()

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

        ;

    }

Also add the both the new methods

    public AutoMapperProfiles()

    {

        Map\_AppUser\_To\_UserDto();

        Map\_Photo\_To\_PhotoDto();

        Map\_UserUpdate\_To\_AppUser();

        Map\_UserRegister\_To\_AppUser();

        Map\_AppUser\_To\_UserTokenDto();

    }

# Business Logic

## UserBusinessLigic.cs

Login and register methods have been updated

    public async Task<UserTokenDto> RegisterAsync(UserRegisterDto registerUser)

    {

        if (registerUser == null)

            throw new ValidationException("Invalid user"); //exception middleware

        var user = await RegisterUser(registerUser);

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

            throw new ValidationException("Unable to create registration"); //exception middleware

        var userToken = \_mapper.Map<UserTokenDto>(user);

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

        return userToken;

    }

    public async Task<UserTokenDto> LoginAsync(LoginDto login)

    {

        if (login == null)

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

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

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

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

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

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

        if (hashKeyLogin == null)

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

        //both are byte[]

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

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

        //build and return user token

        var userToken = \_mapper.Map<UserTokenDto>(user);

        userToken.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"); //exception middleware

        //check user not already taken

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

        if (isUser)

            throw new ValidationException("Username already taken"); //exception middleware

        //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"); //exception middleware

        //convert to AppUser to register

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

        user.PasswordHash = hashKey.Hash;

        user.PasswordSalt = hashKey.Salt;

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

        if (!isRegister)

            throw new DataFailException("User not registerd");

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

        if (returnUser == null)

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

        return returnUser;

    }