Table of Contents

[Basics 3](#_Toc117550062)

[Current Resources with Changes 3](#_Toc117550063)

[New Resources 3](#_Toc117550064)

[Important 3](#_Toc117550065)

[Nuget Package: Microsoft.AspNetCore.Identity.EntityFramewaorkCore 3](#_Toc117550066)

[Change Set #1 : Preparing for Identity Use 3](#_Toc117550067)

[Core/Entities 3](#_Toc117550068)

[AppUser.cs 3](#_Toc117550069)

[AppRole.cs 4](#_Toc117550070)

[AppUserRole.cs 4](#_Toc117550071)

[/Core/DB 4](#_Toc117550072)

[Seed.cs 4](#_Toc117550073)

[DataContext.cs 5](#_Toc117550074)

[/Core/Extensions 7](#_Toc117550075)

[ServiceIdentityExtensions.cs 7](#_Toc117550076)

[/Core/BusinessLogic 8](#_Toc117550077)

[UsersBusinessLogic.cs 8](#_Toc117550078)

[Add Migrations – Update Database 9](#_Toc117550079)

[Add migrations 9](#_Toc117550080)

[Program.cs 9](#_Toc117550081)

[Update database 9](#_Toc117550082)

[Change Set #2 – Database Update 9](#_Toc117550083)

[Program.cs 9](#_Toc117550084)

[Database Handling 9](#_Toc117550085)

[Drop Database 9](#_Toc117550086)

[Run the App 10](#_Toc117550087)

[Change Set #3 – Register and Login Updates 10](#_Toc117550088)

[Business Logic 10](#_Toc117550089)

[UsersBusinessLogic.cs 10](#_Toc117550090)

[Postman Testing 11](#_Toc117550091)

[Change Set #4 – Roles 11](#_Toc117550092)

[/Core/Constants 11](#_Toc117550093)

[SiteIdentityConstants.cs 11](#_Toc117550094)

[/Core/DB/Seed.cs 12](#_Toc117550095)

[Seed Roles and Add Role to USer 12](#_Toc117550096)

[Create an Admin User 12](#_Toc117550097)

[Program.cs 12](#_Toc117550098)

[Change Set #5 – Database Update 12](#_Toc117550099)

[Database Handling 12](#_Toc117550100)

[Drop Database 12](#_Toc117550101)

[Run the App 13](#_Toc117550102)

[Change Set #6 – Policy Based Authorization & Adding Claims to Token 13](#_Toc117550103)

[/Core/Services/TokenService 13](#_Toc117550104)

[ITokenService.cs 13](#_Toc117550105)

[TokenService 13](#_Toc117550106)

[/Core/BusinessLogic/UsersBusinessLogic.cs 14](#_Toc117550107)

[/Core/Dto/UserClaimGetDto.cs 14](#_Toc117550108)

[/Core/Extensions/ClaimsPrincipalExtensions.cs 15](#_Toc117550109)

[Policy Based Authorization 15](#_Toc117550110)

[/Core/Constants/SiteIdentityConstants.cs 15](#_Toc117550111)

[/Core/Extensions/ServiceIdentityExtensions.cs 15](#_Toc117550112)

[Using policy on new Controllers/AdminsController.cs 15](#_Toc117550113)

[Test with Postman 16](#_Toc117550114)

[Change Set #7 – Returning Users Roles and Editing Roles 16](#_Toc117550115)

[/Core/Extensions/StringExtensions.cs 16](#_Toc117550116)

[/Core/BusinessLogic/UsersBusinessLogic 16](#_Toc117550117)

[IUsersBusinessLogic.cs 16](#_Toc117550118)

[UsersBusinessLogic.cs 16](#_Toc117550119)

[/Controllers/AccountController.cs 18](#_Toc117550120)

[Postman Testing 19](#_Toc117550121)

# Basics

|  |  |
| --- | --- |
| WorkingFolder | Copy the content of “Site-04-Message Feature” in “Site-05-Identity Role Management” and issue   * dotnet build |
| Cloudinary Setting | appsetting.json is not checked in so make sure to bring in this from “Site-01-Basics” |

# Current Resources with Changes

1. /Core/Entities/AppUser
2. /Core/BusinessLogic/UsersBusinessLogic
3. /Core/DB/Seed.cs
4. /Core/DB/DataContext
5. /Core/Extensions/ServiceIdentityExtensions
6. /Core/Services/TokenService
7. /Core/Dto/UserClaimGetDto
8. /Core/Dto/StringExtensions

# New Resources

1. Nuget Package: Microsoft.AspNetCore.Identity.EntityFramewaorkCore
2. /Core/Entities/AppRole
3. /Core/Entities/AppUserRole
4. /Core/Constants/SiteIdentityConstants

# Important

1. Email verification and password reset is not part of this refactor
2. The current login system with tokens will work but switching to ASP.Net Identity
3. All the code old code removed and new code in place

# Nuget Package: Microsoft.AspNetCore.Identity.EntityFramewaorkCore

Graphical user interface, text, application

Description automatically generated

# Change Set #1 : Preparing for Identity Use

## Core/Entities

### AppUser.cs

#### Additions

Derive from IdentityUser<int>

public class AppUser: IdentityUser<int>

Add AppUSerRole

    /// <summary>

    /// acting as a join table

    /// </summary>

    public ICollection<AppUserRole> UserRoles { get; set; }

#### Remove Redundant fields

    public string UserName { get; set; }

    public byte[] PasswordHash { get; set; }

    public byte[] PasswordSalt { get; set; }

### AppRole.cs

using System.Collections.Generic;

using Microsoft.AspNetCore.Identity;

namespace MSC.Api.Core.Entities;

public class AppRole : IdentityRole<int>

{

    public ICollection<AppUserRole> UserRoles { get; set; }

}

### AppUserRole.cs

using Microsoft.AspNetCore.Identity;

namespace MSC.Api.Core.Entities;

public class AppUserRole : IdentityUserRole<int>

{

    public AppUser User { get; set; }

    public AppRole Role { get; set; }

}

## /Core/DB

### Seed.cs

* Use the UserManager instead of DataContext to seed users.
* Don’t need to create the token any more manually

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.Json;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.DB;

public class Seed

{

    //UserManager used instead of DataContext

    public static async Task SeedUsers(UserManager<AppUser> userManager)

    {

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

        if (await userManager.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;

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

        foreach (var user in users)

        {

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

            //saves the users to the database as well. no need to do SaveChangesAsync

            await userManager.CreateAsync(user, "A1abcd");

        }

    }

}

### DataContext.cs

* Not doing DbSet for Users any more
* Create UserRoles

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.DB

{

    //Removed use of DataContext : DbContext

    public class DataContext : IdentityDbContext<AppUser, AppRole, int,

                                                IdentityUserClaim<int>, AppUserRole, IdentityUserLogin<int>,

                                                IdentityRoleClaim<int>, IdentityUserToken<int>>

    {

        public DataContext(DbContextOptions options) : base(options)

        {

        }

        //no need to DbSet Users any more with Identity

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

        //UserLike will have a table name of Likes

        public DbSet<UserLike> Likes { get; set; }

        //UserMessage will have a table name of Messages

        public DbSet<Message> Messages { get; set; }

        //give entities some configuration

        protected override void OnModelCreating(ModelBuilder builder)

        {

            base.OnModelCreating(builder);

            //dont foget to add migrations

            //dotnet ef migrations add MessageEntityAdded -o Core/DB/Migrations

            //and then either issue command "dotnet ef database update"

            //or do dontnet run. For this check program.cs "CUSTOM: Seed Data Start" section

            //Due to use of Identity

            CreateUserRole(builder);

            CreateUserLike(builder);

            CreateMessage(builder);

        }

        /// <summary>

        /// User roles due to use of Identity

        /// </summary>

        /// <param name="builder"></param>

        private void CreateUserRole(ModelBuilder builder)

        {

            builder.Entity<AppUser>()

                    .HasMany(ur => ur.UserRoles)

                    .WithOne(u => u.User)

                    .HasForeignKey(ur => ur.UserId)

                    .IsRequired()

            ;

            builder.Entity<AppRole>()

                    .HasMany(ur => ur.UserRoles)

                    .WithOne(u => u.Role)

                    .HasForeignKey(ur => ur.RoleId)

                    .IsRequired()

            ;

        }

        /// <summary>

        /// Configure User Likes

        /// </summary>

        /// <param name="builder"></param>

        private void CreateUserLike(ModelBuilder builder)

        {

            //user like configuration

            //key is combination of sourceUserId and LikedUserId

            builder.Entity<UserLike>()

                    .HasKey(k => new { k.SourceUserId, k.LikedUserId });

            //build relationships between AppUser and UserLike. Here the users liked by the logged in user

            builder.Entity<UserLike>()

                    .HasOne(s => s.SourceUser)

                    .WithMany(l => l.UsersILiked)

                    .HasForeignKey(s => s.SourceUserId)

                    .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities. For sql server use DeleteBehavior.NoAction

            ;

            //build relationships between AppUser and UserLike. Here the logged in user liked by other users

            builder.Entity<UserLike>()

                    .HasOne(s => s.LikedUser)

                    .WithMany(l => l.UsersLikedMe)

                    .HasForeignKey(s => s.LikedUserId)

                    .OnDelete(DeleteBehavior.Cascade) //when the user is deleted then delete the related entities. For sql server use DeleteBehavior.NoAction

            ;

        }

        /// <summary>

        /// Confiure User Messages

        /// </summary>

        /// <param name="builder"></param>

        private void CreateMessage(ModelBuilder builder)

        {

            //user message configuration

            //receiver

            builder.Entity<Message>()

                    .HasOne(r => r.Receipient)

                    .WithMany(m => m.MessagesReceived)

                    .OnDelete(DeleteBehavior.Restrict) //both the parties need to delete the message to be removed from the database

            ;

            //sender

            builder.Entity<Message>()

                    .HasOne(s => s.Sender)

                    .WithMany(m => m.MessagesSent)

                    .OnDelete(DeleteBehavior.Restrict)

            ;

        }

    }

}

## /Core/Extensions

### ServiceIdentityExtensions.cs

using System.Text;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.IdentityModel.Tokens;

using MSC.Api.Core.DB;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Extensions;

public static class IdentityServiceExtensions

{

    public static void AddIdentityServices(this IServiceCollection services, IConfiguration config)

    {

        //for mvc use services.AddIdentity. For the api we can't do that

        services.AddIdentityCore<AppUser>(opt =>

        {

            //there are a lot of options that we can configure here

            //pick per the site password scheme

            opt.Password.RequireNonAlphanumeric = false;

            opt.Password.RequireDigit = false;

            opt.Password.RequireLowercase = false;

            opt.Password.RequireUppercase = false;

        })

        //roles

        .AddRoles<AppRole>()

        //role manager

        .AddRoleManager<RoleManager<AppRole>>()

        //Sign in manager

        .AddSignInManager<SignInManager<AppUser>>()

        //Role validator

        .AddRoleValidator<RoleValidator<AppRole>>()

        //and finally add store to create all the identity related tables

        .AddEntityFrameworkStores<DataContext>();

        ;

        var tokenKey = Encoding.UTF8.GetBytes(config.GetTokenKey());

        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

                .AddJwtBearer(options =>

                {

                    options.TokenValidationParameters = new TokenValidationParameters

                    {

                        ValidateIssuerSigningKey = true,

                        IssuerSigningKey = new SymmetricSecurityKey(tokenKey),

                        ValidateIssuer = false,

                        ValidateAudience = false

                    };

                });

    }

}

Text

Description automatically generated

## /Core/BusinessLogic

### UsersBusinessLogic.cs

Updated the register and login methods to remove the current password hashing

    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)

            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

        //convert to AppUser to register

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

        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;

    }

## Add Migrations – Update Database

### Add migrations

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

Add migrations result. Warnings are ok since we are dropping the columns

Build started...

Build succeeded.

The Entity Framework tools version '6.0.6' is older than that of the runtime '6.0.10'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAc1fbw for more information.

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 initialized 'DataContext' using provider 'Microsoft.EntityFrameworkCore.Sqlite:6.0.6' with options: None

An operation was scaffolded that may result in the loss of data. Please review the migration for accuracy.

Done. To undo this action, use 'ef migrations remove'

### Program.cs

Go to program.cs and comment the following line

    //await Seed.SeedUsers(context);

### Update database

* dotnet ef database update

Make sure that no errors are there and then look at the database

# Change Set #2 – Database Update

## Program.cs

Go to the custom section where seeding data and use UserManager to seed data now

    //due to identity get the userManager and then use it to seed users

    var userManager = services.GetRequiredService<UserManager<AppUser>>();

    //now seed the users

    await Seed.SeedUsers(userManager);

## Database Handling

### Drop Database

* dotnet ef database drop

Build started...

Build succeeded.

The Entity Framework tools version '6.0.6' is older than that of the runtime '6.0.10'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAc1fbw for more information.

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 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 6.0.10 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

* dotnet watch run

This will

* execute the seed method
* run migrations
* and create the database again from UserManager and we’ll get the password hashes etc

# Change Set #3 – Register and Login Updates

## Business Logic

### UsersBusinessLogic.cs

#### Constructor and Properties

Add UserManager and SignInManager to the constructor and initialize the properties

    private readonly UserManager<AppUser> \_userManager;

    private readonly SignInManager<AppUser> \_signInManager;

    private readonly IUsersRepository \_usersRepo;

    private readonly ITokenService \_tokenService;

    private readonly IPhotoService \_photoService;

    private readonly IMapper \_mapper;

    public UsersBusinessLogic(

        UserManager<AppUser> userManager,

        SignInManager<AppUser> signInManager,

        IUsersRepository usersRepo,

        ITokenService tokenService,

        IPhotoService photoService,

        IMapper mapper)

    {

        \_tokenService = tokenService;

        \_userManager = userManager;

        \_signInManager = signInManager;

        \_usersRepo = usersRepo;

        \_photoService = photoService;

        \_mapper = mapper;

    }

#### Register User

Update Private Method RegisterUser

    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

        //convert to AppUser to register

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

        //using userManager register create the user

        var result = await \_userManager.CreateAsync(user, registerUser.Password);

        if (!result.Succeeded)

            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;

    }

#### Login Method Update

    public async Task<UserTokenDto> LoginAsync(LoginDto login)

    {

        if (login == null)

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

        var user = await \_userManager.Users

                                    .Include(p => p.Photos)

                                    .SingleOrDefaultAsync(x => x.UserName == login.UserName.ToLowerInvariant());

        if (user == null)

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

        var result = await \_signInManager.CheckPasswordSignInAsync(user, login.Password, false);

        if (!result.Succeeded)

            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;

    }

### Postman Testing

Using the previously used postman method try creating a new user and login as Lisa. Both should succeed.

# Change Set #4 – Roles

## /Core/Constants

### SiteIdentityConstants.cs

using System.Collections.Generic;

using MSC.Api.Core.Entities;

namespace MSC.Api.Core.Constants;

public class SiteIdentityConstants

{

    public const string Role\_Member = "Member";

    public const string Role\_Admin = "Admin";

    public const string Role\_Moderator = "Moderator";

    public static List<AppRole> SiteRoles = new List<AppRole>

    {

        new AppRole{ Name = SiteIdentityConstants.Role\_Member },

        new AppRole { Name = SiteIdentityConstants.Role\_Admin },

        new AppRole { Name = SiteIdentityConstants.Role\_Moderator }

    };

}

## /Core/DB/Seed.cs

Receives RoleManager as well

    public static async Task SeedUsers(UserManager<AppUser> userManager, RoleManager<AppRole> roleManager)

### Seed Roles and Add Role to USer

Create roles and assign to the user as well

        //Create roles

        foreach (var role in SiteIdentityConstants.SiteRoles)

        {

            await roleManager.CreateAsync(role);

        }

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

        foreach (var user in users)

        {

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

            //saves the users to the database as well. no need to do SaveChangesAsync

            await userManager.CreateAsync(user, "A1abcd");

            //assign a role to the user as well

            await userManager.AddToRoleAsync(user, SiteIdentityConstants.Role\_Member);

        }

### Create an Admin User

        //create a new admin user so that we have atleast one user to begin with

        var admin = new AppUser() { UserName = "admin", DisplayName = "Admin", DateOfBirth = DateTime.Now.AddYears(-19), GuId = System.Guid.NewGuid(), Gender = "male" };

        await userManager.CreateAsync(admin, "A1abcd");

        //add assign Admin and Moderator roles to Admin user

        await userManager.AddToRolesAsync(admin, new[] { SiteIdentityConstants.Role\_Admin, SiteIdentityConstants.Role\_Moderator });

## Program.cs

Pass the role manager to the Seed class as well

    //due to identity get the userManager and Role Manager then use it to seed users and roles

    var userManager = services.GetRequiredService<UserManager<AppUser>>();

    var roleManager = services.GetRequiredService<RoleManager<AppRole>>();

    //now seed the users

    await Seed.SeedUsers(userManager, roleManager);

# Change Set #5 – Database Update

## Database Handling

### Drop Database

* dotnet ef database drop

Build started...

Build succeeded.

The Entity Framework tools version '6.0.6' is older than that of the runtime '6.0.10'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAc1fbw for more information.

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 6.0.10 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 6.0.10 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

* dotnet watch run

This will seed the users and roles.

# Change Set #6 – Policy Based Authorization & Adding Claims to Token

## /Core/Services/TokenService

### ITokenService.cs

    Task<string> CreateToken(AppUser user);

### TokenService

Add Usermanager to the constructor

    private readonly UserManager<AppUser> \_usermanager;

    public TokenService(IConfiguration config, UserManager<AppUser> usermanager)

    {

        \_usermanager = usermanager;

Change to CreateToken to async and add roles to claim

    public async Task<string> CreateToken(AppUser user)

    {

        if (user == null)

            throw new Exception("User info missing");

        //claims

        var claims = new List<Claim>

        {

            new Claim(JwtRegisteredClaimNames.NameId, user.Id.ToString()),

            new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName),

            new Claim("Guid", user.GuId.ToString()),

            new Claim("DisplayName", user.DisplayName),

        };

        //get roles and add to the claims above with a custom name

        var roles = await \_usermanager.GetRolesAsync(user);

        claims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role, role)));

        //credentials with key

        var creds = new SigningCredentials(\_key, SecurityAlgorithms.HmacSha512Signature);

        //describe token

        var tokenDescriptor = new SecurityTokenDescriptor

        {

            Subject = new ClaimsIdentity(claims),

            Expires = DateTime.Now.AddDays(7),

            SigningCredentials = creds

        };

        //token handler

        var tokenHandler = new JwtSecurityTokenHandler();

        //token

        var token = tokenHandler.CreateToken(tokenDescriptor);

        var writeToken = tokenHandler.WriteToken(token);

        return writeToken;

    }

## /Core/BusinessLogic/UsersBusinessLogic.cs

In LoginAsync method update toeknService call

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

In RegisterAsync method update the tokenService call

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

In the private RegisterUser method update add the role for the user as well.

    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

        //convert to AppUser to register

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

        //using userManager register create the user

        var result = await \_userManager.CreateAsync(user, registerUser.Password);

        if (!result.Succeeded)

            throw new DataFailException(result.Errors.ToString());

        //add the user to the member role as well

        var roleResult = await \_userManager.AddToRoleAsync(user, SiteIdentityConstants.Role\_Member);

        if (!roleResult.Succeeded)

            throw new DataFailException(roleResult.Errors.ToString());

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

        if (returnUser == null)

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

        return returnUser;

    }

## /Core/Dto/UserClaimGetDto.cs

Add a new property to populate user roles from claim

using System;

using System.Collections.Generic;

namespace MSC.Api.Core.Dto;

public class UserClaimGetDto

{

    public string UserName { get; set; }

    public int UserId { get; set; }

    public Guid Guid { get; set; }

    public string DisplayName { get; set; }

    public List<string> Roles { get; set; }

    public bool HasUserName => !string.IsNullOrWhiteSpace(UserName);

    public bool HasGuid => Guid != Guid.Empty;

    public bool HasId => UserId > 0;

    public bool HasPrimaryInfo => HasUserName && HasGuid && HasId;

}

## /Core/Extensions/ClaimsPrincipalExtensions.cs

Add and update following to pick the roles from the claims

    public static List<string> GetRoles(this ClaimsPrincipal principal)

    {

        if (principal == null) return null;

        var roleClaims = principal.FindAll(ClaimTypes.Role)?.ToList();

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

        var roles = roleClaims.Select(x => x.Value).ToList();

        return roles;

    }

    public static UserClaimGetDto GetUserClaims(this ClaimsPrincipal principal)

    {

        if (principal == null) return null;

        var claimsDto = new UserClaimGetDto()

        {

            UserId = principal.GetUserId(),

            UserName = principal.GetUserName(),

            Guid = principal.GetUserGuid(),

            DisplayName = principal.GetDisplayName(),

            Roles = principal.GetRoles()

        };

        return claimsDto;

    }

## Policy Based Authorization

### /Core/Constants/SiteIdentityConstants.cs

Add the following two constants to it

    public const string AuthPolicy\_RequireAdminRole = "RequireAdminRole";

    public const string AuthPolicy\_ModeratePhotoRole = "RequireModerateRole";

## /Core/Extensions/ServiceIdentityExtensions.cs

Add the policies to the ServiceIdentityExtensions.AddIdentityServices method

        //add authorization policies

        services.AddAuthorization(opt =>

        {

            opt.AddPolicy(SiteIdentityConstants.AuthPolicy\_RequireAdminRole, policy => policy.RequireRole(SiteIdentityConstants.Role\_Admin));

            opt.AddPolicy(SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole, policy => policy.RequireRole(SiteIdentityConstants.Role\_Admin, SiteIdentityConstants.Role\_Moderator));

        });

## Using policy on new Controllers/AdminsController.cs

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MSC.Api.Core.Constants;

namespace MSC.Api.Controllers

{

    public class AdminController : BaseApiController

    {

        [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_RequireAdminRole)]

        [HttpGet("users-with-roles")]

        public ActionResult GetUsersWithRoles()

        {

            return Ok("Only admins can see this");

        }

        [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole)]

        [HttpGet("photos-to-moderate")]

        public ActionResult GetPhotosForModeration()

        {

            return Ok("Admins or moderators can see this");

        }

    }

}

### Test with Postman

Test the new controller with postman by logging in with different users and checked

# Change Set #7 – Returning Users Roles and Editing Roles

## /Core/Extensions/StringExtensions.cs

Create an extension to split the string to List T

    public static IEnumerable<T> StringSplitToType<T>(this string value, string delimiter = ",")

    {

        var defaultVal = default(IEnumerable<T>);

        if (string.IsNullOrWhiteSpace(value)) return defaultVal;

        var splitResult = value.Split(new[] { delimiter }, StringSplitOptions.RemoveEmptyEntries).Select(p => p.Trim()).ToArray();

        if (splitResult.Length <= 0 || (splitResult.Length == 1 && string.IsNullOrWhiteSpace(splitResult[0])))

            return defaultVal;

        var newO = Activator.CreateInstance<List<T>>();

        foreach (var item in splitResult)

        {

            try

            {

                newO.Add((T)Convert.ChangeType(item, typeof(T)));

            }

            catch { }

        }

        if (!newO.Any()) return defaultVal;

        return newO;

    }

## /Core/BusinessLogic/UsersBusinessLogic

### IUsersBusinessLogic.cs

Add a new method to return users with their roles

    Task<IEnumerable<object>> GetUSersWithRoles();

    Task<BusinessResponse> EditRolesForUser(int adminUSerId, Guid userToUpdate, IEnumerable<string> roles);

### UsersBusinessLogic.cs

Inject role manager

    private readonly UserManager<AppUser> \_userManager;

    private readonly SignInManager<AppUser> \_signInManager;

    private readonly RoleManager<AppRole> \_roleManager;

    private readonly IUsersRepository \_usersRepo;

    private readonly ITokenService \_tokenService;

    private readonly IPhotoService \_photoService;

    private readonly IMapper \_mapper;

    public UsersBusinessLogic(

        UserManager<AppUser> userManager,

        SignInManager<AppUser> signInManager,

        RoleManager<AppRole> roleManager,

        IUsersRepository usersRepo,

        ITokenService tokenService,

        IPhotoService photoService,

        IMapper mapper)

    {

        \_tokenService = tokenService;

        \_userManager = userManager;

        \_signInManager = signInManager;

        \_roleManager = roleManager;

        \_usersRepo = usersRepo;

        \_photoService = photoService;

        \_mapper = mapper;

    }

Add the methods to get the users with roles and edit roles.

    public async Task<IEnumerable<object>> GetUSersWithRoles()

    {

        //get the users, include UserRoles and then Role

        //return an annonamous object

        var users = await \_userManager.Users

                                    .Include(r => r.UserRoles)

                                    .ThenInclude(r => r.Role)

                                    .OrderBy(u => u.UserName)

                                    .Select(u => new

                                    {

                                        u.Id,

                                        UserName = u.UserName,

                                        DisplayName = u.DisplayName,

                                        GuId = u.GuId,

                                        Roles = u.UserRoles.Select(r => r.Role.Name).ToList()

                                    })

                                    .ToListAsync()

                                    ;

        return users;

    }

    public async Task<BusinessResponse> EditRolesForUser(int adminUSerId, Guid userToUpdate, IEnumerable<string> roles)

    {

        //check user

        var user = await \_userManager.Users.SingleOrDefaultAsync(x => x.GuId == userToUpdate);

        if (user == null)

            return new BusinessResponse(HttpStatusCode.NotFound, "User not found to update");

        //check roles to update

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

            return new BusinessResponse(HttpStatusCode.BadRequest, "No roles passed to update");

        //get the siteRoles

        var siteRoles = await \_roleManager.Roles.Select(x => x.Name).ToListAsync();

        //check roles to update are in the site roles

        var updateRolesNotInSiteRoles = roles.Where(x => !siteRoles.Any(y => y == x)).ToList();

        if (updateRolesNotInSiteRoles != null && updateRolesNotInSiteRoles.Any())

            return new BusinessResponse(HttpStatusCode.BadRequest, $"Passed role(s) not in list {string.Join(", ", updateRolesNotInSiteRoles)}");

        //current user roles

        var userRoles = await \_userManager.GetRolesAsync(user);

        //add the new roles that are not in current roles

        var result = await \_userManager.AddToRolesAsync(user, roles.Except(userRoles));

        if (!result.Succeeded)

            return new BusinessResponse(HttpStatusCode.BadRequest, "Failed to add to roles");

        //remove the roles as well since the user may have removes some. Above is only adding new ones

        var removeResult = await \_userManager.RemoveFromRolesAsync(user, userRoles.Except(roles));

        if (!removeResult.Succeeded)

            return new BusinessResponse(HttpStatusCode.BadRequest, "Failed to remove roles");

        //pick new roles

        var currentRoles = await \_userManager.GetRolesAsync(user);

        return new BusinessResponse(HttpStatusCode.OK, "Roles updated successfully", currentRoles);

    }

## /Controllers/AccountController.cs

* update action “users-with-roles” to fetch the users and roles
* create a new action to edit the user roles
* Don’t forget to inject userBusinessLogic

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MSC.Api.Core.BusinessLogic;

using MSC.Api.Core.Constants;

using MSC.Api.Core.Extensions;

namespace MSC.Api.Controllers;

public class AdminController : BaseApiController

{

    private readonly IUsersBusinessLogic \_userBl;

    public AdminController(IUsersBusinessLogic userBl)

    {

        \_userBl = userBl;

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_RequireAdminRole)]

    [HttpGet("users-with-roles")]

    public async Task<ActionResult<IEnumerable<object>>> GetUsersWithRoles()

    {

       var users = await \_userBl.GetUSersWithRoles();

        return Ok(users);

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_ModeratePhotoRole)]

    [HttpGet("photos-to-moderate")]

    public ActionResult GetPhotosForModeration()

    {

        return Ok("Admins or moderators can see this");

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_RequireAdminRole)]

    [HttpPost("edit-roles/{guid:Guid}")]

    public async Task<ActionResult<IEnumerable<string>>> EditRoles([FromRoute] Guid guid, [FromQuery] string roles)

    {

        if (string.IsNullOrWhiteSpace(roles))

            return BadRequest("No roles provided to update");

        var rolesList = roles.StringSplitToType<string>();

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

            return BadRequest("Unable to parse the roles provided");

        //edit the roles

        var result = await \_userBl.EditRolesForUser(User.GetUserId(), guid, rolesList);

        ActionResult actionResult = BadRequest("Unable to edit roles");

        if (result != null)

        {

            switch (result.HttpStatusCode)

            {

                case HttpStatusCode.OK:

                    actionResult = Ok(result.ConvertDataToType<IEnumerable<string>>());

                    break;

                case HttpStatusCode.BadRequest:

                    actionResult = BadRequest(result.Message);

                    break;

                case HttpStatusCode.NotFound:

                    actionResult = NotFound(result.Message);

                    break;

                default:

                    actionResult = BadRequest("Unable to edit roles");

                    break;

            }

        }

        return actionResult;

    }

}

### Postman Testing

Test the users-with-roles action again and you should see a result like following

Text

Description automatically generated