# Project

* Site-15-Api-Ng-identity-role-management
  + Copied from: Site-14-Api-Ng-messaging-queryparams-routeresolvers
* 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/AppRole.cs | MSC.WebApi/Constrollers/AdminController.cs |
| MSC.core/DB/Entities/AppUserRole.cs |  |
| MSC.Core/Constants/SiteIdentityConstants.cs |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/DB/Entities/AppUser.cs | MSC.WebApi/Program.cs |
| MSC.Core/BusinessLogic/UserBusinessLogic |  |
| MSC.Core/DB/Data/SeedData.cs |  |
| MSC.Core/DB/Data/DataContext.cs |  |
| MSC.core/Extensions/AppServiceExtensions.cs |  |
| MSC.Core/Services/TokenService |  |
| MSC.Core/Extensions/StringExtensions.cs |  |

# Clear Database

* New migrations created
* Clear the users
* dotnet ef database drop
* dotnet ef database update

# Intro

Refactor the code to use ASP.NET identity and gain an understanding of the following

* using .NET identity
* Role management
* Policy based authorization
* UserManager<T>
* SignInManager<T>
* RoleManager<T>

Why?

* Battle hardned, written and tested by Microsoft
* Comes with a password hasher with 10,000 salt iterations
  + Our custom solution is only doing 1 pass
* Full framework for managing members and roles
* Provide an entity framework schema to create the needed tables
* Highly customizable

## Important

Check for comment IR\_REFATCOR to find all the code that has been changed.

## User and Role

* A user can have many roles
* A role can have may users
* So these are in many to many relationship
* AppUserRole will get the User and Role properties
* AppUser and AppRole will both use AppAppUserRole as a collection to later define Many to Many relationship

# Step 1 - Setup

## MSC.Core/DB

### Entities

#### Updated AppUser.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

namespace MSC.Core.DB.Entities;

//IR\_REFATCOR : derive from IdentityUser,

//remove Id, userName, passwordHash, PasswordSalt since these are coming from IdentityUser

//remove the Column(Order attribute

[Index(nameof(Guid))]

[Index(nameof(UserName))]

public class AppUser : IdentityUser<int> /\*int here wil make the key int, default is string\*/

{

    /\*

    //[Column(Order = 1)]

    public int Id { get; set; }

    \*/

    //[Column(Order = 2)]

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    [Required]

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

    /\*

    //[Column(Order = 3)]

    [Required]

    public string UserName { get; set; }

    \*/

    /\*

    //[Column(Order = 4)]

    [Required]

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

    \*/

    /\*

    //[Column(Order = 5)]

    [Required]

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

    \*/

    //[Column(Order = 6)]

    public DateOnly DateOfBirth { get; set; }

    //[Column(Order = 7)]

    public string DisplayName { get; set; }

    //[Column(Order = 8)]

    public string Gender { get; set; }

    //[Column(Order = 9)]

    public string Introduction { get; set; }

    //[Column(Order = 10)]

    public string LookingFor { get; set; }

    //[Column(Order = 11)]

    public string Interests { get; set; }

    //[Column(Order = 12)]

    public string City { get; set; }

    //[Column(Order = 13)]

    public string Country { get; set; }

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

    //[Column(Order = 14)]

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

    //[Column(Order = 15)]

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

    //[Column(Order = 16)]

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

    /\*

    public int GetAge(){

        var age = DateOfBirth.CalculateAge();

        return age;

    }

    \*/

    /// <summary>

    /// Other users that liked the logged in User. CheckDB Context for relationships

    /// </summary>

    public List<UserLike> LikedByUsers { get; set; }

    /// <summary>

    /// Users that the logged in user liked. CheckDB Context for relationships

    /// </summary>

    public List<UserLike> LikedUsers { get; set; }

    //for messages, check DB Context

    public List<UserMessage> MessagesSent { get; set; }

    public List<UserMessage> MessagesReceived { get; set; }

    //navigation property to the join table

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

}

#### New AppRole.cs

using System.Collections.Generic;

using Microsoft.AspNetCore.Identity;

namespace MSC.Core.DB.Entities;

//IR\_REFATCOR

//Many to Many relationship with AppUser where

//A user can have may roles

//and a role can have many users

public class AppRole : IdentityRole<int>

{

    //navigation property to the join table

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

}

#### New AppUserRole.cs

using Microsoft.AspNetCore.Identity;

namespace MSC.Core.DB.Entities;

//IR\_REFATCOR

//A joining table between AppUser and AppRole

//Used in both AppRole and and AppUser

//User <=> Role have Many to Many relationship

//A user can have many roles

//and a role can have many users

//so put iCollection of AppUSerRole in both AppRole and AppUser

public class AppUserRole : IdentityUserRole<int>

{

    public AppUser User { get; set; }

    public AppRole Role { get; set; }

}

#### Update UserBusinessLogic.cs

Since the properties have been remove so remove the use of the same

##### RegisterUserAsync Method

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

        ////IR\_REFATCOR: removed these properties

        //appUser.PasswordHash = hashSalt.Hash;

        //appUser.PasswordSalt = hashSalt.Salt;

##### LoginAsync.cs

        /\*

        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");

        \*/

#### Update SeedData.cs

            ////IR\_REFATCOR: removed these properties

            /\*

            user.PasswordHash = hashkey.Hash;

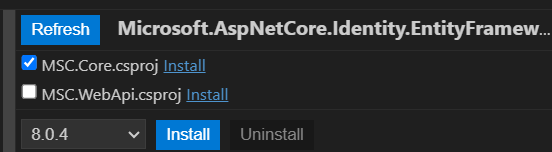
            user.PasswordSalt = hashkey.Salt;

            \*/

## Packages

### Microsoft.AspNetCore.Identity.EntityFrameworkCore

By Microsoft



## Re-Configuring DataContext

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

#### Derive from IdentityDbcontext

Order is important

//IR\_REFATCOR

//public class DataContext : DbContext

public class DataContext : IdentityDbContext<AppUser, //class we created

                                            AppRole, //class we created

                                            int, //type int as that is what we defined AppUser, AppRole, AppUserRole

                                            IdentityUserClaim<int>,

                                            AppUserRole, //class we created

                                            IdentityUserLogin<int>,

                                            IdentityRoleClaim<int>,

                                            IdentityUserToken<int>

                                            >

#### AppUSer all remove

    //IR\_REFATCOR

    /\*

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

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

    \*/

#### Create relationship

    protected override void OnModelCreating(ModelBuilder builder)

    {

        base.OnModelCreating(builder);

        //IR\_REFATCOR Due to use of Identity

        CreateUserRole(builder);

And the method

    //IR\_REFATCOR Many to Many relationship

    private void CreateUserRole(ModelBuilder builder)

    {

        builder.Entity<AppUser>()

                    .HasMany(ur => ur.UserRoles)

                    .WithOne(u => u.User) //Property in AppUserRole

                    .HasForeignKey(ur => ur.UserId)

                    .IsRequired()

            ;

            builder.Entity<AppRole>()

                    .HasMany(ur => ur.UserRoles)

                    .WithOne(u => u.Role) //Property in AppUserRole

                    .HasForeignKey(ur => ur.RoleId)

                    .IsRequired()

            ;

    }

## Configuring Identity - Startup Class

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

#### AddAuthenticationService

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

    {

        //IR\_REFACOR - Identity, out it before services.AddAuthentication

        //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>>()

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

        .AddEntityFrameworkStores<DataContext>()

        ;

        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

        .AddJwtBearer(options => {

            options.TokenValidationParameters = new TokenValidationParameters

            {

                ValidateIssuerSigningKey = true,

                IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config.GetTokenKey())),

                ValidateIssuer = false,

                ValidateAudience = false

            };

        });

        return services;

    }

# Step 2 – Drop DB / Add Migaration / SeedData

## Add Migrations – Update Database

> dotnet ef migrations add IdentityAdded

This will give a warning about loss of data. This is ok since we have removed the columns from the Users table.

### Comment Seed Data - Temporarily

Before running the app or issuing update database, go to programs.cs and comment out the seed data

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);

}

> dotnet ef database update

|  |  |
| --- | --- |
| Before | After |
|  |  |

## MSC.Core/DB/Data

### SeedData.cs

Mark the current method as obsolete

//IR\_REFACTOR

public class SeedData

{

    /// <summary>

    /// Broken after Identity use, Before IR\_REFACTOR. Use new method SeedUsers that takes in UserManager

    /// </summary>

    /// <param name="context"></param>

    /// <returns></returns>

    [Obsolete]

    public static async Task SeedUsers(DataContext context)

Create the method

    /// <summary>

    /// New method after Identity implementation

    /// </summary>

    /// <returns></returns>

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

    {

        //if users then do not seed new data

        if(await userManager.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;

        //default password

        var defaultPassword = "Password@1";

        // convert the username to lower case

        foreach(var user in users){

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

            await userManager.CreateAsync(user, defaultPassword); //will save as well.

        }

    }

## MSC.Webapi/Program.cs

Use the new method to seed data.

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

//IR\_REFACTOR

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

var services = scope.ServiceProvider;

try{

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

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

    //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);

    await SeedData.SeedUsers(userManager);

}

catch(Exception ex)

{

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

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

}

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

## 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'.

## Update Database

> dotnet ef database update

Or run app

## Run App to Seed Data

> dotnet watch --no-hot-reload

Check AspNetUsers table and all 10 users specified in MSC.Core/DB/Data/UserSeedData.json should be created again.

# Step 3 – UserExists / Login / Register Updates

Update Register and Login

## MSC.Core/BusinessLogic/UserBusinessLogic.cs

### Constructor

Add userManager

    private readonly UserManager<AppUser> \_userManager;

    private readonly IUserRepository \_userRepo;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

    private readonly IPhotoService \_photoService;

    public UserBusinessLogic(UserManager<AppUser> userManager, IUserRepository userRepo, ITokenService tokenService, IMapper mapper, IPhotoService photoService)

    {

        \_userManager = userManager;

        \_userRepo = userRepo;

        \_tokenService = tokenService;

        \_mapper = mapper;

        \_photoService = photoService;

    }

### UserExists Method

    public async Task<bool> UserExists(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("User name missing");

        //IR\_REFACTOR : use the user manager

        //return await \_userRepo.UserExists(userName);

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

    }

### RegisterUserAsync Method

    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");

        //IR\_REFACTOR

        /\*

        //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();

        \*/

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

        ////IR\_REFATCOR: removed these properties

        //appUser.PasswordHash = hashSalt.Hash;

        //appUser.PasswordSalt = hashSalt.Salt;

        //IR\_REFACTOR

        /\*

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

        if(!isRegister)

            throw new DataFailException("User not registerd");

        \*/

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

        if(!result.Succeeded)

            throw new DataFailException(string.Join(", ", result.Errors));

        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);;

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

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

        return loggedInUser;

    }

### LoginAsync Method

    public async Task<LoggedInUserDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing");

        //IR\_REFATCOR:

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

        var user = await \_userManager.Users

                                    .Include(p => p.Photos)

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

        ////IR\_REFATCOR: removed these properties

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

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

        if(user == null)

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

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

        ////IR\_REFATCOR: removed these properties

        /\*

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

        if (hashKeyLogin == null)

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

        //both are byte[]

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

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

        \*/

        var result = await \_userManager.CheckPasswordAsync(user, login.Password);

        if (!result)

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

        //mapping via manual user mapper

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

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

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

        return loggedInUser;

    }

# Step 4 – Add Roles and Assign to Users

## Constants

### MSC.Core/Constants/SiteIdentityConstants.cs

using System.Collections.Generic;

using MSC.Core.DB.Entities;

namespace MSC.Core.Constants;

//IR\_REFACTOR

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}

    };

    public static List<AppRole> SiteAdminModeratorRoles = new List<AppRole>{

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

        new AppRole{ Name = SiteIdentityConstants.Role\_Moderator}

    };

    public const string AuthPolicy\_Admin = "AuthPolicy\_Admin";

    public const string AuthPolicy\_Moderator\_Photos = "AuthPolicy\_Moderator\_Photos";

}

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

* Update the new SeedUsers method, [created above](#_SeedData.cs)
* It will now receive roleManager
* Add “Member” role to current users being read from json
* Add new new admin user with Admin and Moderator roles

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

    {

        //if users then do not seed new data

        if(await userManager.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;

        //Roles

        List<AppRole> roles = SiteIdentityConstants.SiteRoles; //Helper property to get the roles as List<AppRole>

        foreach(var role in roles){

            await roleManager.CreateAsync(role);

        }

        //default password

        var defaultPassword = "Password@1";

        //convert the username to lower case

        foreach(var user in users){

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

            await userManager.CreateAsync(user, defaultPassword); //will save as well.

            //add the user to Member Role

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

        }

        //create Admin User and add Admin and Moderator roles

        var adminUser = new AppUser{

            UserName = "admin",

            DisplayName = "Admin",

            Gender = "male"

        };

        await userManager.CreateAsync(adminUser, defaultPassword);

        var adminModeratorRoles = SiteIdentityConstants.SiteAdminModeratorRoles.Select(x => x.Name).ToList();

        await userManager.AddToRolesAsync(adminUser, adminModeratorRoles);

    }

## MSC.WebApi/Program.cs

Pass the roleManager to SeedData

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

//IR\_REFACTOR

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

var services = scope.ServiceProvider;

try{

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

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

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

    //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);

    await SeedData.SeedUsers(userManager, roleManager);

}

catch(Exception ex)

{

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

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

}

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

## MSC.Core/BusinessLogic/BusinessLogic.cs

### RegisterUserAsync Method

Add the user to member role after successful registration.

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

        if(!result.Succeeded)

            throw new DataFailException(string.Join(", ", result.Errors));

        //add the user to the member role

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

        if(!roleResult.Succeeded)

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

## Drop Database

> dotnet ef database drop

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'.

## Run the App again

> dotnet watch --no-hot-reload

## Check Tables

See “[UserIds and Password.txt](../UserIds%20And%20Passwords.txt)” in the root, Site 15 section

## Add Roles to Claims – TokenService

### MSC.Core/Services/ITokenService.cs

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

namespace MSC.Core.Services;

public interface ITokenService

{

    Task<string> CreateToken(AppUser user);

}

### MSC.Core/Services/TokenService.cs

Add the user roles to claims

#### Constructor

Inject UserManager

    private readonly SymmetricSecurityKey \_key;

    private readonly UserManager<AppUser> \_userManager;

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

    {

        var tokenKey = config.GetTokenKey(); //from app settings config using the extension created

        if(string.IsNullOrWhiteSpace(tokenKey))

            throw new Exception("TokenKey missing");

        \_key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(tokenKey));

        \_userManager = userManager;

    }

#### CreateToken Method

Change the signature.

Add roles after adding user claims

    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);

        if(roles != null)

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

        else

            claims.Add(new Claim(ClaimTypes.Role, SiteIdentityConstants.Role\_Member)); //default it to Member when no roles found

## MSC.Core/BusinessLogic/UserBusinessLogic.s

### RegisterUserAsync

Await the Create token call

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

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

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

        return loggedInUser;

### LoginAsync

Await the create token call

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

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

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

        return loggedInUser;

## MSC.Core/Mappers/ManualMapperExtensions.cs

This is not being used any more so remove comment out the CreateTokenCall

### ManualMapToLoggedInUSerDto

        //IR\_REFACTOR :  CreateToken is async call now so commented it. Not used any more

        //loggedInUser.Token = tokenService.CreateToken(user);

## Check the Token

### Login as a Blanca

{

    "id": 5,

    "userName": "blanca",

    "guid": "092a9b9e-9fe3-4fc6-8b9d-488d33b33059",

    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiI1IiwidW5pcXVlX25hbWUiOiJibGFuY2EiLCJHdWlkIjoiMDkyYTliOWUtOWZlMy00ZmM2LThiOWQtNDg4ZDMzYjMzMDU5IiwiRGlzcGxheU5hbWUiOiJCbGFuY2EiLCJyb2xlIjoiTWVtYmVyIiwibmJmIjoxNzE1NjYxMzA5LCJleHAiOjE3MTYyNjYxMDksImlhdCI6MTcxNTY2MTMwOX0.barVKk4E72LYWgXVlPPYvo31zPlFyqUWkBx3lZj95OKTdXE8RIrlpqCHMsNTKaG8SfObrsxIep\_zaKoLqGLl3Q",

    "displayName": "Blanca",

    "mainPhotoUrl": "https://randomuser.me/api/portraits/women/27.jpg",

    "gender": "female"

}

Go to <https://jwt.io> and decode the token

{

"nameid": "5",

"unique\_name": "blanca",

"Guid": "092a9b9e-9fe3-4fc6-8b9d-488d33b33059",

"DisplayName": "Blanca",

"role": "Member",

"nbf": 1715661309,

"exp": 1716266109,

"iat": 1715661309

}

### Login as Admin

{

    "id": 11,

    "userName": "admin",

    "guid": "4ce8e019-d3d1-4744-8e49-9e97737867b7",

    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiIxMSIsInVuaXF1ZV9uYW1lIjoiYWRtaW4iLCJHdWlkIjoiNGNlOGUwMTktZDNkMS00NzQ0LThlNDktOWU5NzczNzg2N2I3IiwiRGlzcGxheU5hbWUiOiJBZG1pbiIsInJvbGUiOlsiQWRtaW4iLCJNb2RlcmF0b3IiXSwibmJmIjoxNzE1NjYxNTE1LCJleHAiOjE3MTYyNjYzMTUsImlhdCI6MTcxNTY2MTUxNX0.b2rEO9pUC5X5Bw99ZhiWcnOWeIvvPIgdzixEZIXbAQX5YGanuPvSqlDJfmlZUHMqCEG4QQUeFBKuInldTNNckQ",

    "displayName": "Admin",

    "mainPhotoUrl": "",

    "gender": "male"

}

Go to <https://jwt.io> and decode the token

{

"nameid": "11",

"unique\_name": "admin",

"Guid": "4ce8e019-d3d1-4744-8e49-9e97737867b7",

"DisplayName": "Admin",

"role": [

"Admin",

"Moderator"

],

"nbf": 1715661515,

"exp": 1716266315,

"iat": 1715661515

}

# Step 5 – CopyToOutputDirectory [Not Needed]

Currently the file is being picked in SeedData.cs by

* getting the base directory and then
* moving upto MSC.Core and
* finally moving down to the file

## MSC.Core/MSC.Core.csproj

Open the project file and manually add the seed data file to it as new itemgroup

CopytToOutputDirectory could be either

* Always or
* PreserveNewest

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <TargetFramework>net8.0</TargetFramework>

    <!--

    <ImplicitUsings>disable</ImplicitUsings>

    <Nullable>disable</Nullable>

    -->

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.1" />

    <PackageReference Include="CloudinaryDotNet" Version="1.26.2" />

    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.2" />

    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="8.0.4" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="8.0.2">

      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

      <PrivateAssets>all</PrivateAssets>

    </PackageReference>

    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="8.0.2" />

    <PackageReference Include="Microsoft.Extensions.Configuration.Binder" Version="8.0.1" />

    <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="7.3.1" />

  </ItemGroup>

  <ItemGroup>

    <None Update="DB\Data\UserSeedData.json">

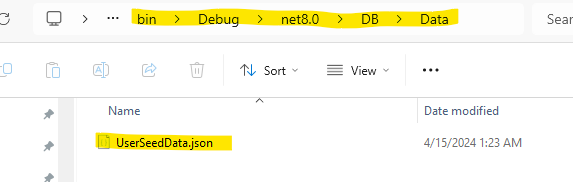
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>

    </None>

  </ItemGroup>

</Project>

## Build Solution or individual Projects



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

Change the path pick from the executing dir

        //seed data file location

        //var x = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

        //C:\\...\\MySocialConnect-API\\MSC.WebApi\\bin\\Debug\\net8.0

        var location = System.AppDomain.CurrentDomain.BaseDirectory;

        //C:\\...\\MySocialConnect-API\\MSC.WebApi\\bin\\Debug\\net8.0\\

        var filePath = @"DB\Data\UserSeedData.json";

        //old way of getting to the file

        /\*

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

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

        var fileFullPath = @$"{file}\MSC.Core\{filePath}";

        \*/

        //Since file has been setup with CopyToOutputDirectory=Always, check MSC.Core.csproj for details.

        var fileFullPath = Path.Combine(location, filePath);

# Step 6 – Adding Policy Based Authorization

## Roles Based Authorization

### UserController.cs

We can do like

    [Authorize(Roles = SiteIdentityConstants.Role\_Admin)]

    [HttpGet]

    public async Task<ActionResult<PagedList<UserDto>>> GetUsers([FromQuery]UsersSearchParamDto userParams)

When the user is not in role then 403 response will be sent back.

## Policy Based Authorization

### /MSC.Core/Constants/SiteIdentityConstants.cs

Two constants have already been added

    public const string AuthPolicy\_Admin = "AuthPolicy\_Admin";

    public const string AuthPolicy\_Moderator\_Photos = "AuthPolicy\_Moderator\_Photos";

## Add Policy to AppServiceExtensions

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

#### AddAuthenticationServie Method

1. The first one added is AddIdentityCore, [above](#_AddAuthenticationService) when configured identity
2. The second one is AddAuthentication, which was [already there](002%20Project3%20-WebApi-Auth-JWT-Reg-Login-Extensions.docx), Add Authentication Service section.
3. The third item added is AddAuthorization.
4. Order is important for these.

        services.AddAuthorization(opt => {

            opt.AddPolicy(SiteIdentityConstants.AuthPolicy\_Admin,

                            policy => policy.RequireRole(SiteIdentityConstants.Role\_Admin));

            opt.AddPolicy(SiteIdentityConstants.AuthPolicy\_Moderator\_Photos,

                            policy => policy.RequireRole(SiteIdentityConstants.Role\_Admin, SiteIdentityConstants.Role\_Moderator));

        });

## MSC.WebApi/Controllers/AdminController.cs

* New controller, simple setup. Will be overwritten [below](#_Step_7_–).
* Policy base authorization applied to the two actions specified.

namespace MSC.WebApi.Controller;

public class AdminController : BaseApiController

{

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

    [HttpGet("users-with-roles")]

    public ActionResult GetUsersWithRoles()

    {

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

    }

    [Authorize(Policy = SiteIdentityConstants.AuthPolicy\_Moderator\_Photos)]

    [HttpGet("photos-to-moderate")]

    public ActionResult GetPhotosForModeration()

    {

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

    }

}

# Step 7 – Get Users/Roles and Update User Roles

## MSC.Core/Extensions/StringExtensions.cs

### StringSplitToTypes

Add a new method to split the string per the type specified and return the same.

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

    {

        var defaultValue = default(IEnumerable<T>);

        if(string.IsNullOrWhiteSpace(value))

            return defaultValue;

        var result = value

                        .Split(new[] {delimiter}, StringSplitOptions.RemoveEmptyEntries)

                        .Select(x => x.Trim())

                        .ToArray();

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

            return defaultValue;

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

        foreach(var item in result)

        {

            try{

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

            }

            catch{}

        }

        if(!newO.Any())

            return defaultValue;

        return newO;

    }

## MSC.Core/BusinessLogic/UserBusinessLogic.cs

### IUserBusinessLogic.cs

    //from admin controller after IR\_REFACTOR

    Task<IEnumerable<object>> GetUSersWithRoles();

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

### UserBusinessLogic.cs

#### Constructor

Add roleManager to constructor

    private readonly UserManager<AppUser> \_userManager;

    private readonly RoleManager<AppRole> \_roleManager;

    private readonly IUserRepository \_userRepo;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

    private readonly IPhotoService \_photoService;

    public UserBusinessLogic(UserManager<AppUser> userManager, RoleManager<AppRole> roleManager,

                            IUserRepository userRepo, ITokenService tokenService, IMapper mapper,

                            IPhotoService photoService)

#### GetUsersWithRoles

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

    {

        //get the users, include UserRoles and then Role

        //return an annonamous object

        //exclude admin user

        var users = await \_userManager.Users

                        //.Include(r => r.UserRoles)

                        //.ThenInclude(r => r.Role)

                        //.Where(u => u.UserName.ToLower() != "admin")

                        .OrderBy(u => u.DisplayName)

                        .Select(u => new {

                            u.Id,

                            UserName = u.UserName,

                            DisplayName = u.DisplayName,

                            GuId = u.Guid,

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

                        })

                        .ToListAsync();

        return users;

    }

#### EditRolesForUSer

    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 site roles

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

        //check roles to update are in siteRoles

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

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

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

        //current user roles

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

        //add the new roles only that do not below to user currently

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

        if(!resultAdd.Succeeded)

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

        //remove the roles as since the user may have removed some. Above is oly 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 updates successfully!", currentRoles);

    }

## AdminController.cs

### Constructor

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.Core.BusinessLogic;

using MSC.Core.Constants;

using MSC.Core.Extensions;

namespace MSC.WebApi.Controller;

public class AdminController : BaseApiController

{

    private readonly IUserBusinessLogic \_userBl;

    public AdminController(IUserBusinessLogic \_userBl)

    {

        this.\_userBl = \_userBl;

    }

### GetUserswithRoles

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

    [HttpGet("users-with-roles")]

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

    {

        var users = await \_userBl.GetUSersWithRoles();

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

            return NotFound("No users found");

        return Ok(users);

    }

And here is the response from this method

[

    {

        "id": 8,

        "userName": "atkinson",

        "displayName": "Atkinson",

        "guid": "6dfc56e9-a86e-480c-ac23-7f802462d6a1",

        "roles": [

            "Member"

        ]

    },

    {

        "id": 5,

        "userName": "blanca",

        "displayName": "Blanca",

        "guid": "092a9b9e-9fe3-4fc6-8b9d-488d33b33059",

        "roles": [

            "Member"

        ]

    }

]

### EditRoles

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

    [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");

        //roles are comma seperated list so split

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

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

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

        //edit the roles

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

        ActionResult actionResult = result.HttpStatusCode switch{

            HttpStatusCode.OK => Ok(result.ConvertDataToType<IEnumerable<string>>()),

            HttpStatusCode.BadRequest => BadRequest(result.Message),

            HttpStatusCode.NotFound => NotFound(result.Message),

            \_ => BadRequest("Unable to edit roles")

        };

        return actionResult;

    }

Response from above

[

    "Member"

]

# Postman

In the root “Postman\_collection.json”

Check Site-15