# Project

* Site-02-Entity-Cors
  + Copied from: Site-01-Basics
* For the “MySocialConnect-API”
  + dotnet restore
  + dotnet build
  + Go to project : MSC.WebApi
    - dotnet build : to build
    - dotnet run : to run the api

## MySocialConnect-API Folder Structure

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
|  |  |

# New Resources

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| DB/Entities/AppUser.cs | Controller/BaseApiController.cs |
| DB/Data/DbContext.cs | Controller/UsersController |
| Constants/ConfigKeyConstants.cs |  |
| Repositories/IUserRepository.cs |  |
| Repositories/UserRepository.cs |  |
| BusinessLogic/IUserBusinessLogic |  |
| BusinessLogic/UserBusinessLogic |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core.csproj | MSC.WebApi.csproj |
|  | Program.cs |
|  | WeatherForecast.cs |
|  | Controllers/WeatherForecastController.cs |
|  | appsettings.Development.json |

# Turnoff Nullable and ImplicitUsings

For both MSC.Core and MSC.WebApi go to the csproj file and turn off the nullable and implicitUsings

## MSC.Core

### MSC.Core.csproj

|  |  |
| --- | --- |
| From | To |
| <PropertyGroup>      <TargetFramework>net8.0</TargetFramework>      <ImplicitUsings>enable</ImplicitUsings>      <Nullable>enable</Nullable>    </PropertyGroup> | <PropertyGroup>      <TargetFramework>net8.0</TargetFramework>      <ImplicitUsings>disable</ImplicitUsings>      <Nullable>disable</Nullable>    </PropertyGroup> |

## MSC.WebApi

### MSC.WebApi.csproj

|  |  |
| --- | --- |
| From | To |
| <PropertyGroup>      <TargetFramework>net8.0</TargetFramework>      <ImplicitUsings>enable</ImplicitUsings>      <Nullable>enable</Nullable>    </PropertyGroup> | <PropertyGroup>      <TargetFramework>net8.0</TargetFramework>      <ImplicitUsings>disable</ImplicitUsings>      <Nullable>disable</Nullable>    </PropertyGroup> |

### Program.cs

Add the missing using statements

### WeatherForecast.cs

Add the missing using statements

Remove ? rom string Summary property

### Controllers

#### WeatherForecastController.cs

Add the missing using statements

## Compile and Run

* Build the solution
* Run MSC.WebAPI
* Check swagger to make sure that WeatherForecast is working
  + <https://localhost:5001/swagger>

# Adding Packages

## EntityFramework

Open [000 Project1 - Basic Info - Setup.docx](000%20Project1%20-%20Basic%20Info%20-%20Setup.docx) and follow

C# Packages and then “Add Entity Framework”

## Dotnet – ef

Open [000 Project1 - Basic Info - Setup.docx](000%20Project1%20-%20Basic%20Info%20-%20Setup.docx) and follow

C# Packages and then “Add dotnet-ef”

# MSC.Core/Constants

## ConfigKeyConstants.cs

Add a new constant file. This will have keys for the items specified in App

# MSC.Core/DB

## Entities

### AppUser.cs

Create an entity with three properties Id (int), Guid (Guid) and UserName (string)

using System;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.EntityFrameworkCore;

namespace MSC.Core.DB.Entities;

[Index(nameof(Guid))]

[Index(nameof(UserName))]

public class AppUser

{

    public int Id { get; set; }

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    public Guid Guid { get; set; }

    public string UserName { get; set; }

}

# MSC.Core/Data

## DataContext.cs

Create a data constant class

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Entities;

namespace MSC.Core.DB.Data;

public class DataContext : DbContext

{

    public DataContext(DbContextOptions options) : base(options)

    {

    }

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

}

# MSC.WebApi – Setting Data

## DbFile

Create an empty folder. This will be where the SQLITE db will be created.

## appsettings.development.json

Add the connection string to it

  "ConnectionStrings": {

    "DefaultConnection": "Data source=DbFile/MySocialConnect.db"

  }

## Program.cs

Add DBContext to program.cs

/\*\*\*Custom Section Start\*\*\*/

//DBContext and connection string

//Migration assembly is needed since DBContext is in MSC.Core where as the Migrations are getting created in MSC.WebApi via

//dotnet ef migrations add InitialCreate -o DbFile/Migrations

//if every thing is in MSC.WebApi then b is not needed.

builder.Services.AddDbContext<DataContext>(opt => {

    opt.UseSqlite(builder.Configuration.GetConnectionString(ConfigKeyConstants.DefaultConnection),

                    b => b.MigrationsAssembly("MSC.WebApi")

                );

});

/\*\*\*Custom Section End\*\*\*/

builder.Services.AddControllers();

# Migrations & Database

dotnet ef migrations -h

* add
* bundle
* list
* remove
* script

## Create Migrations

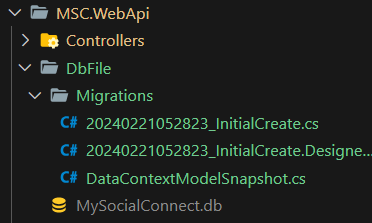
Create the migrations in MSC.WebApi/DbFile

>dotnet ef migrations add InitialCreate -o DbFile/Migrations

|  |  |
| --- | --- |
|  |  |

## Update Database

>dotnet ef database update



## Commands

### Migrations

* dotnet ef migrations add InitialCreate -o DbFile/Migrations
  + any time a change a made is to the entities run the above command, just change the InitialCreate part and then run update database command from below.
* dotnet ef migrations remove

### Database Update

After migrations update the database

* dotnet ef database update

### Completely remove all migrations and start all over again

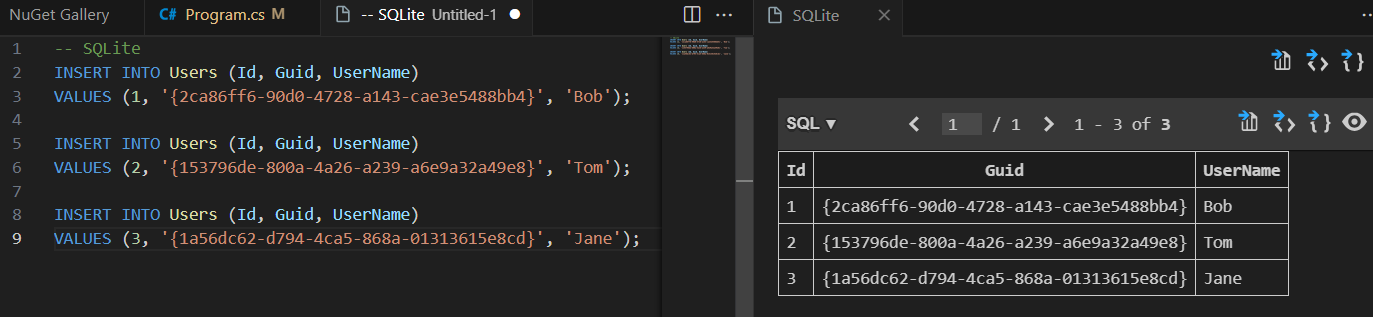
* dotnet ef database update 0
* dotnet ef migrations remove

### Drop database and recreate

* dotnet ef database drop
* dotnet ef database update

## Create Data Manually

This is temporary step till seed data is put in place. SQLITE extension added, check the [000 Project1 - Basic Info – Setup](000%20Project1%20-%20Basic%20Info%20-%20Setup.docx) C# Related Extensions for more help.



# Getting Data

## MSC.Core/Repositories

### IUserRepository.cs

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

namespace MSC.Core.Repositories;

public interface IUserRepository

{

    Task<IEnumerable<AppUser>> GetUsersAsync();

    Task<AppUser> GetUserAsync(int id);

    Task<AppUser> GetUserAsync(string userName);

    Task<AppUser> GetUserAsync(Guid guid);

}

### UserRepository.cs

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using MSC.Core.DB.Data;

using MSC.Core.DB.Entities;

namespace MSC.Core.Repositories;

public class UserRepository : IUserRepository

{

    private readonly DataContext \_context;

    public UserRepository(DataContext context)

    {

        \_context = context;

    }

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

    {

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

        return users;

    }

    public async Task<AppUser> GetUserAsync(int id)

    {

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

        return user;

    }

    public async Task<AppUser> GetUserAsync(string userName)

    {

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

        return user;

    }

    public async Task<AppUser> GetUserAsync(Guid guid)

    {

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

        return user;

    }

}

## MSC.Core/BusinessLogic

### IUserBusinessLogic

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

namespace MSC.Core.BusinessLogic;

public interface IUserBusinessLogic

{

    Task<IEnumerable<AppUser>> GetUsersAsync();

    Task<AppUser> GetUserAsync(int id);

    Task<AppUser> GetUserAsync(string userName);

    Task<AppUser> GetUserAsync(Guid guid);

}

### UserBusinessLogic

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using MSC.Core.DB.Entities;

using MSC.Core.Repositories;

namespace MSC.Core.BusinessLogic;

public class UserBusinessLogic : IUserBusinessLogic

{

    private readonly IUserRepository \_userRepo;

    public UserBusinessLogic(IUserRepository userRepo)

    {

        \_userRepo = userRepo;

    }

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

    {

        var users = await \_userRepo.GetUsersAsync();

        return users;

    }

    public async Task<AppUser> GetUserAsync(int id)

    {

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

        return user;

    }

    public async Task<AppUser> GetUserAsync(string userName)

    {

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

        return user;

    }

    public async Task<AppUser> GetUserAsync(Guid guid)

    {

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

        return user;

    }

}

## MSC.WebApi/Programs.cs

Add the repository and business logic for DI

// Add services to the container.

/\*\*\*Custom Section Start\*\*\*/

//DBContext and connection string

//Migration assembly is needed since DBContext is in MSC.Core where as the Migrations are getting created in MSC.WebApi via

//dotnet ef migrations add InitialCreate -o DbFile/Migrations

//if every thing is in MSC.WebApi then b is not needed.

builder.Services.AddDbContext<DataContext>(opt => {

    opt.UseSqlite(builder.Configuration.GetConnectionString(ConfigKeyConstants.DefaultConnection),

                    b => b.MigrationsAssembly("MSC.WebApi")

                );

});

//add resources for DI

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

/\*\*\*Custom Section End\*\*\*/

builder.Services.AddControllers();

## MSC.WebApi/Controllers

### BaseApiController.cs

All of the controller will derive from this base controller

using Microsoft.AspNetCore.Mvc;

namespace MSC.WebApi.Controller;

[ApiController]

[Route("api/[controller]")]

public class BaseApiController : ControllerBase

{

}

### UserController.cs

Create a new UserController and derive from the BaseApiController

It will have the methods to get users data from the business logic

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MSC.Core.BusinessLogic;

using MSC.Core.DB.Entities;

namespace MSC.WebApi.Controller;

// /api/users

public class UsersController : BaseApiController

{

    private readonly IUserBusinessLogic \_userBusinessLogic;

    public UsersController(IUserBusinessLogic userBusinessLogic)

    {

        \_userBusinessLogic = userBusinessLogic;

    }

    [HttpGet]

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

    {

        var users = await \_userBusinessLogic.GetUsersAsync();

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

        {

            return NotFound("No users found!");

        }

        return users.ToList();

    }

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

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

    {

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

        if (user == null)

        {

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

        }

        return user;

    }

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

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

    {

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

        if (user == null)

        {

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

        }

        return user;

    }

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

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

    {

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

        if (user == null)

        {

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

        }

        return user;

    }

}

# Adding CORS Support

## MSC.WebApi

### Program.cs

The angular app will be connections from <https://localhost:4200/> or <http://localhost:4200/>.

Create a CORS policy to allow the angular app to connect.

#### Change 1

/\*\*\*Custom Section Start\*\*\*/

//DBContext and connection string

//Migration assembly is needed since DBContext is in MSC.Core where as the Migrations are getting created in MSC.WebApi via

//dotnet ef migrations add InitialCreate -o DbFile/Migrations

//if every thing is in MSC.WebApi then b is not needed.

builder.Services.AddDbContext<DataContext>(opt => {

    opt.UseSqlite(builder.Configuration.GetConnectionString(ConfigKeyConstants.DefaultConnection),

                    b => b.MigrationsAssembly("MSC.WebApi")

                );

});

//add resources for DI

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

//CORS

builder.Services.AddCors();

/\*\*\*Custom Section End\*\*\*/

#### Change 2

app.UseHttpsRedirection();

/\*\*\*Custom Section Start\*\*\*/

//ordering is important here. UseCors before UseAuthentication and UseAuthentication before UseAuthorization

app.UseCors(x => x.AllowAnyHeader().AllowAnyMethod().WithOrigins("https://localhost:4200", "http://localhost:4200"));

/\*\*\*Custom Section End\*\*\*/

app.UseAuthorization();