# Entity Framework

## Wait – Do first

Check the Adding Packages section in document [0001 Project Setup.docx](0001%20Project%20Setup.docx).

# Constants

* We will be using a lot of constants so these will go in their on section
* Create a folder Core/Constants
* Make sure to put the namespace as OAuth2.WebApi.Core.Constants

## ConfigKeyConstants.cs

namespace OAuth2.WebApi.Core.Constants;

public class ConfigKeyConstants

{

    public const string DefaultConnection = "DefaultConnection";

    public const string AllowedHosts = "AllowedHosts";

    public const string LoggingLevelDefault = "Logging:LogLevel:Default";

    public const string LoggingLevelMsAspNetCore = "Logging:LogLevel:Microsoft.AspNetCore";

}

## ContentTypeConstants.cs

namespace OAuth2.WebApi.Core.Constants;

public class ContentTypeConstants

{

    public const string ApplicationJson = "application/json";

}

# Entities

Create all entities in folder Core/Entities

## AppUser.cs

Make sure to change the namespace to namespace OAuth2.WebApi.Core.Entities;.

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace OAuth2.WebApi.Core.Entities;

public class AppUser

{

    /// <summary>

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

    /// </summary>

    [Key]

    public int Id { get; set; }

    /// <summary>

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

    /// </summary>

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    public Guid GuId { get; set; }

    public string UserName { get; set; }

}

# Setting Up Connection String

## appsettings.Development.json

Go to appsettings.Development.json add the connect string to it.

We are using SQLite so this connection will be very simple

A screen shot of a computer

Description automatically generated

# DataContext Class

## Setting up DataContext

* Create the DataContext class in Core/DB folder
* Make sure to change the namespace to namespace OAuth2.WebApi.Core.DB;.
* DataContext will derive from DbContext.
* Create a DbSet for AppUser

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.DB;

/// <summary>

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

/// </summary>

public class DataContext : DbContext

{

    public DataContext(DbContextOptions options) : base(options)

    {

    }

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

    protected override void OnModelCreating(ModelBuilder modelBuilder)

    {

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

        modelBuilder.Entity<AppUser>()

        .Property(x => x.GuId)

        .ValueGeneratedOnAdd();

    }

}

## Add DataContext as a Service

Go to programs.cs and add a service.

A screen shot of a computer program

Description automatically generated

# Adding Migrations

## Important

* Issue dotnet build command to make sure that there are no errors
* Create migrations in folder Core/DB/Migrations so that all DB related stuff is in one location
* Make sure that you are in OAuth2.WebApi folder on command prompt

## Adding Initial Migration

> dotnet ef migrations add InitialCreate -o Core/DB/Migrations

Build started...

Build succeeded.

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

A screenshot of a computer

Description automatically generated

## Create Database

>dotnet ef database update

A screenshot of a computer

Description automatically generated

# Look at SQLite Database

Open SQLlite by SHIFT+CTRL+P and typing sqlite and then selecting SQLite: Open Database

A screenshot of a computer

Description automatically generated

It will ask for the database to open so select it

A black box with white text

Description automatically generated

This will give SQLite explorer so use it to look at the database and tables.

A black and white text

Description automatically generated with medium confidence

# Adding Data to Users Table Manually

We’ll seed the users later.

For now insert few records in the database manually.

Check Script - Creating Users Manually.txt in Documents folder for the script to create users.

A black screen with white text

Description automatically generated

# Repositories

* Create all repositories in folder Core/Data/Repositories
* Make the namespace as OAuth2.WebApi.Core.Data.Repositories

## UserRepository

### IUserRepository.cs

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Data.Repositories;

public interface IUserRepository

{

    Task<IEnumerable<AppUser>> GetAppUsersAsync();

    Task<AppUser> GetAppUserAsync(int id);

    Task<AppUser> GetAppUserAsync(string userName);

    Task<AppUser> GetAppUserAsync(Guid guid);

}

### UserRepository.cs

using System.ComponentModel.DataAnnotations;

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.DB;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Data.Repositories;

public class UserRepository : IUserRepository

{

    private readonly DataContext \_context;

    public UserRepository(DataContext context)

    {

        \_context = context;

    }

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

    {

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

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

        return users;

    }

    public async Task<AppUser> GetAppUserAsync(int id)

    {

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

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

        return user;

    }

    public async Task<AppUser> GetAppUserAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

        var user = await \_context.Users.SingleOrDefaultAsync(x => x.UserName.ToLower() == userName.ToLower());

        return user;

    }

    public async Task<AppUser> GetAppUserAsync(Guid guid)

    {

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

        return user;

    }

}

# BusinessLogic

* Create all businessLogic in folder Core/Data/BusinessLogic
* Make the namespace as OAuth2.WebApi.Core.Data.BusinessLogic

## UserBusinessLogic

### IUserBusinessLogic.cs

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Data.BusinessLogic;

public interface IUserBusinessLogic

{

    Task<IEnumerable<AppUser>> GetAppUsersAsync();

    Task<AppUser> GetAppUserAsync(int id);

    Task<AppUser> GetAppUserAsync(string userName);

    Task<AppUser> GetAppUserAsync(Guid guid);

}

### UserBusinessLogic.cs

using System.ComponentModel.DataAnnotations;

using OAuth2.WebApi.Core.Data.Repositories;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Data.BusinessLogic;

public class UserBusinessLogic : IUserBusinessLogic

{

    private readonly IUserRepository \_userRepository;

    public UserBusinessLogic(IUserRepository userRepository)

    {

        \_userRepository = userRepository;

    }

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

    {

        var users = await \_userRepository.GetAppUsersAsync();

        return users;

    }

    public Task<AppUser> GetAppUserAsync(int id)

    {

        var user = \_userRepository.GetAppUserAsync(id);

        return user;

    }

    public async Task<AppUser> GetAppUserAsync(string userName)

    {

        if (string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid userName");

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

        return user;

    }

    public async Task<AppUser> GetAppUserAsync(Guid guid)

    {

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

        return user;

    }

}

# DependencyInjection (DI)

Add UserRepository and UserBusinessLogic for DI as services in program.cs

A screen shot of a computer program

Description automatically generated

# UsersController

using Microsoft.AspNetCore.Mvc;

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Controllers;

[ApiController]

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

public class UsersController : ControllerBase

{

    private readonly IUserBusinessLogic \_userBL;

    public UsersController(IUserBusinessLogic userBL)

    {

        \_userBL = userBL;

    }

    /// <summary>

    /// /api/users

    /// </summary>

    /// <returns></returns>

    [HttpGet]

    //public ActionResult<IEnumerable<AppUser>> GetUsers()

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

    {

        var users = await \_userBL.GetAppUsersAsync();

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

            return NotFound("No users found!");

        return Ok(users);

    }

    /// <summary>

    /// /api/users/2

    /// </summary>

    /// <param name="id"></param>

    /// <returns></returns>

    [HttpGet("{id}", Name = "GetUserById")]

    //public ActionResult<AppUser> GetUser(int id)

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

    {

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

        if (user == null)

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

        return Ok(user);

    }

    /// <summary>

    /// /api/users/name/Bob

    /// </summary>

    /// <param name="name"></param>

    /// <returns></returns>

    [HttpGet("name/{name}", Name = "GetUserByName")]

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

    {

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

        if (user == null)

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

        return Ok(user);

    }

    /// <summary>

    /// /api/users/guid/1870F084-8E81-44FC-AC6A-8640697F5B1A

    /// </summary>

    /// <param name="guid"></param>

    /// <returns></returns>

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

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

    {

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

        if (user == null)

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

        return Ok(user);

    }

}

# Postman Testing

## Run WEB API Project

>dotnet run

## Setup up URL variable

Setup a global variable

A screenshot of a computer

Description automatically generated

And then adding the url



## Testing End Points

|  |  |
| --- | --- |
| Get All Users A screenshot of a computer  Description automatically generated | Get User By Id |
| Get User By Name | Get User by GUID |