# Project

* Site-03-Api-Auth-JWT-Reg-Login-Extensions
  + Copied from: Site-02-Entity-Cors
* 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 |
| Extensions/CryptoExtension.cs | Controllers/AccountController |
| Dtos/HashKeyDto.cs |  |
| Extensions/ByteExtension.cs |  |
| Extensions/StringExtension.cs |  |
| ExceptionCustom/DataFailException |  |
| Extensions/EnvConfigExtension.cs |  |
| Dtos/Helper/Envconfig.cs |  |
| Dtos/LoggedInUserdto |  |
| Dtos/LoginDto.cs |  |
| Dtos/UserRegisterDto.cs |  |
| Extensions/AppServiceExtensions.cs |  |
| Services/ITokenService.cs |  |
| Services/TokenService.cs |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| DB/Entities/AppUser.cs | Program.cs |
| Repositories/IUserRepository.cs | Controller/UserController |
| Repositories/UserRepository.cs |  |
| BusinessLogic/IUserBusinessLogic |  |
| BusinessLogic/UserBusinessLogic |  |

# Login Properties – AppUser

## Add Properties to AppUser.cs and Required

Location: MSC.Core/DB/Entities

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

    [Required]

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

    [Required]

    public string UserName { get; set; }

    [Required]

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

    [Required]

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

## Add Migrations / Update Database

|  |  |
| --- | --- |
| At the solution level run the following commands. Remember the migrations are created in MSC.WebApi/DbFile/Migrations  > dotnet ef migrations add UserPasswordAdded  > dotnet ef database update |  |

# ExceptionCustom - DataFailException

## DataFailException.cs

Location: MSC.Core/ExceptionCustom

using System;

namespace MSC.Core.ExceptionCustom;

public class DataFailException : Exception

{

    public DataFailException()

    {

    }

    public DataFailException(string message) : base(message)

    {

    }

    public DataFailException(string message, Exception innerException) : base(message, innerException)

    {

    }

}

# Setting Up and Using AppSettings.config

## Read

<https://code-maze.com/aspnetcore-read-appsettings-values-from-a-json-file/>

<https://www.c-sharpcorner.com/article/asp-net-core-how-to-read-values-from-appsettings-json/>

## App Settings JSON

|  |  |
| --- | --- |
| Appsettings.json | Appsettings.development.json |
| {    "Logging": {      "LogLevel": {        "Default": "Information",        "Microsoft.AspNetCore": "Warning"      }    },    "AllowedHosts": "\*",    "AllowSpecificOrigins": ["https://localhost:4200", "http://localhost:4200"],    "TokenKey": "7NCyQkWBsqV3bZsT4qShUN6qzpWUjmRs"  } | {    "Logging": {      "LogLevel": {        "Default": "Information",        "Microsoft.AspNetCore": "Information"      }    },    "ConnectionStrings": {      "DefaultConnection": "Data source=DbFile/MySocialConnect.db"    },    "AllowSpecificOrigins": ["https://localhost:4200", "http://localhost:4200"],    "TokenKey": "7NCyQkWBsqV3bZsT4qShUN6qzpWUjmRs"  } |

## MSC.Core/Constants

### ConfigKeyConstants.cs

Add the keys to the ConfigKeyConstants.cs per the app settings

namespace MSC.Core.Constants;

public class ConfigKeyConstants

{

    public const string DefaultConnection = "DefaultConnection";

    public const string TokenKey = "TokenKey";

    public const string AllowSpecificOrigins = "AllowSpecificOrigins";

}

## Option 1 – Options Pattern & IConfiguration

### Setup a model per the App.config

Reading the full app settings into a single file rather than individual sections as on the read #1 link

using System.Collections.Generic;

using System.Text.Json.Serialization;

namespace MSC.Core.Dtos.Helper;

/// <summary>

/// structure is the same as asssettings.json. Add via DI container, check the programs class for details

/// </summary>

public class EnvConfig

{

    public Logging Logging { get; set; }

    public ConnectionStrings ConnectionStrings { get; set; }

    public string DefaultConnectionString => ConnectionStrings?.DefaultConnection ?? string.Empty;

    public string AllowedHosts { get; set; }

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

    public string TokenKey { get; set; }

}

public class ConnectionStrings

{

    public string DefaultConnection { get; set; }

}

public class Logging

{

    public LogLevel LogLevel { get; set; }

}

public class LogLevel

{

    public string Default { get; set; }

    [JsonPropertyName("Microsoft.AspNetCore")]

    public string MicrosoftAspNetCore { get; set; }

}

### Programs.cs – add services to the container

builder.Services.Configure<EnvConfig>(builder.Configuration);

### Use

Showing use of both IConfiguration and IOptions<EnvConfig>

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Options;

using MSC.Core.Dtos.Helper;

namespace MSC.WebApi.Controller;

// /api/sample

public class SampleController : BaseApiController

{

    private readonly IOptions<EnvConfig> \_envConfig;

    private readonly IConfiguration \_config;

    public SampleController(IOptions<EnvConfig> envConfig, IConfiguration config)

    {

        \_envConfig = envConfig;

        \_config = config;

    }

    [HttpPost("sample1")]

    public async Task<ActionResult<bool>> Sample1()

    {

        //IConfiguration

        var config = \_config;

        //using Ioptions

        var howdy = \_envConfig;

        return Ok(true);

    }

}

## Option 2 – Extension Method (Used This In Project)

### Nuget Package Microsoft.Extensions.Configuration.Binder

Install the nuget package for MSC.Core

### MSC.Core/Extensions/EnvConfigExtension.cs

using System.Collections.Generic;

using Microsoft.Extensions.Configuration;

using MSC.Core.Constants;

namespace MSC.Core.Extensions;

public static class EnvConfigExtension

{

    #region Items

    public static string GetDefaultConnectionString(this IConfiguration config)

    {

        var connectionString = config.GetConnectionString(ConfigKeyConstants.DefaultConnection);

        return connectionString;

    }

    public static List<string> GetAllowSpecificOrigins(this IConfiguration config)

    {

        var allowSpecificOrigins = config.GetSectionValue<List<string>>(ConfigKeyConstants.AllowSpecificOrigins, null);

        return allowSpecificOrigins;

    }

    public static string GetTokenKey(this IConfiguration config)

    {

        var tokenKey = config.GetSectionValue<string>(ConfigKeyConstants.TokenKey, string.Empty);

        return tokenKey;

    }

    public static string GetLoggingLevelDefault(this IConfiguration config)

    {

        var loggingLevelDefault = config.GetSectionValue<string>(ConfigKeyConstants.LoggingLevelDefault, string.Empty);

        return loggingLevelDefault;

    }

    public static string GetLoggingLevelMsApnetCore(this IConfiguration config)

    {

        var loggingLevelDefault = config.GetSectionValue<string>(ConfigKeyConstants.LoggingLevelMsAspNetCore, string.Empty);

        return loggingLevelDefault;

    }

    #endregion items

    #region GetSectionValue

    public static T GetSectionValue<T>(this IConfiguration config, string sectionName)

    {

        if (!config.GetSection(sectionName).Exists())

        {

            return default(T);

        }

        var sValue = config.GetSection(sectionName).Get<T>();

        return sValue;

    }

    public static T GetSectionValue<T>(this IConfiguration config, string sectionName, T defaultValue)

    {

        if (!config.GetSection(sectionName).Exists())

        {

            return defaultValue;

        }

        var sValue = config.GetSection(sectionName).Get<T>();

        return sValue;

    }

    #endregion GetSectionValue

}

### Use

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Options;

using MSC.Core.Dtos.Helper;

using MSC.Core.Extensions;

namespace MSC.WebApi.Controller;

// /api/sample

public class SampleController : BaseApiController

{

    private readonly IOptions<EnvConfig> \_envConfig;

    private readonly IConfiguration \_config;

    public SampleController(IOptions<EnvConfig> envConfig, IConfiguration config)

    {

        \_envConfig = envConfig;

        \_config = config;

    }

    [HttpPost("sample1")]

    public async Task<ActionResult<bool>> Sample1()

    {

        //IConfiguration

        var config = \_config;

        //using Ioptions

        var howdy = \_envConfig;

        return Ok(true);

    }

    [HttpPost("sample2")]

    public async Task<ActionResult<bool>> Sample2()

    {

        //IConfiguration with extension

        var config = \_config;

        var tokenKey = \_config.GetTokenKey();

        //using Ioptions

        var howdy = \_envConfig;

        return Ok(true);

    }

}

# MSC.Core/Dtos

## HashKeyDto.cs

Location: MSC.Core/Dtos

namespace MSC.Core.Dtos;

public class HashKeyDto

{

    public byte[] Salt { get; set; }

    public byte[] Hash { get; set; }

}

## UserRegisterDto.cs

using System.ComponentModel.DataAnnotations;

namespace MSC.Core.Dtos;

public class UserRegisterDto

{

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

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

    public string UserName { get; set; }

    /\*

    ^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?\d)(?=.\*?[@#$&()<>]).{8,}$

    (?=.\*?[A-Z]): Requires at least one uppercase letter.

    (?=.\*?[a-z]): Requires at least one lowercase letter.

    (?=.\*?\d): Requires at least one digit (number)

    (?=.\*?[@#$&()<>]): Requires at least one special character from the set @#$&()<>

    .{8,}: Ensures a minimum length of 8 characters (note used)

    \*/

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

    [StringLength(16, MinimumLength = 4)]

    [RegularExpression("^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?\\d)(?=.\*?[@#$&()<>]).+$", ErrorMessage = "Password must have an upper case, a lower case, a number and one special character from the set @#$&()<>")]

    public string Password { get; set; }

}

## LoginDto.cs

using System.ComponentModel.DataAnnotations;

namespace MSC.Core.Dtos;

public class LoginDto

{

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

    public string UserName { get; set; }

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

    public string Password { get; set; }

}

## LoggedInUserDto.cs

using System;

namespace MSC.Core.Dtos;

public class LoggedInUserDto

{

    public string UserName { get; set; }

    public Guid Guid { get; set; }

    public string Token { get; set; }

}

# MSC.Core/Extensions

## StringExtension.cs

### ToTitleCase

Location: MSC.Core/Extensions

using System.Globalization;

namespace MSC.Core.Extensions;

public static class StringExtension

{

    public static string ToTitleCase(this string s)

    {

        if(string.IsNullOrWhiteSpace(s)) return s;

        var newS = CultureInfo.InvariantCulture.TextInfo.ToTitleCase(s.ToLowerInvariant());

        return newS;

    }

}

## CryptoExtension.cs

### Compute Password Hash - Extension

Location: MSC.Core/Extensions

Create two methods using HMACSha512 to

* compute PasswordHash and PasswordSalt to store in the database
* on login to get the salt key from database and reverse passwordHash

using System.ComponentModel.DataAnnotations;

using System.Security.Cryptography;

using System.Text;

using MSC.Core.Dtos;

namespace MSC.Core.Extensions;

public static class CryptoExtension

{

    /// <summary>

    /// Compute HASh for the passed in value

    /// </summary>

    /// <param name="value"></param>

    /// <returns>HashKeyDto</returns>

    public static HashKeyDto ComputeHashHmacSha512(this string value)

    {

        if(string.IsNullOrWhiteSpace(value))

            throw new ValidationException("Value is missing");

        var valueBytes = Encoding.UTF8.GetBytes(value);

        using var hmac = new HMACSHA512();

        var hash = hmac.ComputeHash(valueBytes);

        var salt = hmac.Key;

        var dto = new HashKeyDto(){ Hash = hash, Salt = salt };

        return dto;

    }

    /// <summary>

    /// Compute HASh for the passed in value using the passed in salt. This is for reverse hash compute

    /// </summary>

    /// <param name="value"></param>

    /// <param name="salt"></param>

    /// <returns></returns>

    public static HashKeyDto ComputeHashHmacSha512(this string value, byte[] salt)

    {

        if(string.IsNullOrWhiteSpace(value))

            throw new ValidationException("Value is missing");

        if(salt == null)

            throw new ValidationException("Key is missing");

        var valueBytes = Encoding.UTF8.GetBytes(value);

        using var hmac = new HMACSHA512(salt);

        var hash = hmac.ComputeHash(valueBytes);

        var saltN = hmac.Key;

        var dto = new HashKeyDto(){ Hash = hash, Salt = saltN };

        return dto;

    }

}

## ByteExtension.cs

### AreEqual Byte Extension

Location: MSC.Core/Extensions

Above CryptoExtension creates and returns the hash as byte array. This extension will help with comparing the two byte arrays are same or not

using System.Linq;

namespace MSC.Core.Extensions;

public static class ByteExtension

{

    public static bool AreEqual(this byte[] a, byte[] b)

    {

        var areEqual = a.SequenceEqual(b);

        return areEqual;

    }

}

# JWT – JSON Web Token

## Info

JWT stands for JSON Web Token.

* These are self-contained and can contain credentials, claims, and other information.
  + Self-contained means no session to maintain.
  + Portable – a single token can be used with multiple back ends.
  + No cookies required – mobile friendly
  + Performance – once a token is issued there is no need to make a database request to verify a user authentication.
* Contains three parts, separated by a period
  + First part: It is the header of the token and contains algorithm and type (JWT) of token. Algorithm is used to encrypt the signature in third part of the token.
  + Second part:
    - this is the payload where we can contain claims (roles etc) and credentials.
    - Will have three time stamps
      * nbf: token cannot be used before certain time
      * exp: expiry date time
      * iat: issue date
  + Third part: this is where the token signature is encrypted.

## Request/Response

Graphical user interface, text, application

Description automatically generated with medium confidence

## Nuget Packages

* System.IdentityModel.Tokens.Jwt by Microsoft – add to MSC.Core project

## Token Service

### TokenKey – App Settings Json

Add the following to both app settings json. The key could be any thing

Random string generator: <http://www.unit-conversion.info/texttools/random-string-generator/>

512 characters

"TokenKey": ""

### MSC.Services/ITokenService

Create an interface ITokenService

using MSC.Core.DB.Entities;

namespace MSC.Core.Services;

public interface ITokenService

{

    string CreateToken(AppUser user);

}

### MSC.Services/TokenSevice

Implement the TokenService

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using MSC.Core.DB.Entities;

using MSC.Core.Extensions;

namespace MSC.Core.Services;

public class TokenService : ITokenService

{

    //the key will stay on the server, it will never go to the client

    private readonly SymmetricSecurityKey \_key;

    public TokenService(IConfiguration config)

    {

        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));

    }

    public 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())

        };

        //signing credentials

        var credentials = new SigningCredentials(\_key, SecurityAlgorithms.HmacSha512);

        //describe the token

        var tokenDescriptor = new SecurityTokenDescriptor

        {

            Subject = new ClaimsIdentity(claims),

            Expires = DateTime.Now.AddDays(7), //expire after 7 days

            SigningCredentials = credentials

        };

        //token handler

        var tokenHandler = new JwtSecurityTokenHandler();

        //create token

        var token = tokenHandler.CreateToken(tokenDescriptor);

        //write token

        var writeToken = tokenHandler.WriteToken(token);

        //return

        return writeToken;

    }

}

### Add TokenService for DI – MSC.Webapi

//add resources for DI

builder.Services.Configure<EnvConfig>(builder.Configuration);

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

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

builder.Services.AddScoped<ITokenService, TokenService>();

# MSC.Core/Repositories

## IUserRepository.cs

Add 4 new methods

    Task<AppUser> GetUserRawAsync(string userName);

  Task<bool> UserExists(string userName);

    Task<bool> RegisterUserAsync(AppUser user);

    Task<bool> SaveAllAsync();

## UserRepository.cs

The implementation for the 4 new methods

    public async Task<AppUser> GetUserRawAsync(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("Invalid user name");

        AppUser user = null;

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

        return user;

    }

    public async Task<bool> UserExists(string userName)

    {

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

    }

    public async Task<bool> RegisterUserAsync(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

        var isSave = await SaveAllAsync();

        return isSave;

    }

    public async Task<bool> SaveAllAsync()

    {

        //make sure that the changes have been saved

        var isSave = await \_context.SaveChangesAsync() > 0;

        return isSave;

    }

# MSC.Core/BusinessLogic

## IUserBusinessLogic.cs

Add the 4 new methods

    Task<AppUser> GetUserRawAsync(string userName);

    Task<bool> UserExists(string userName);

    Task<LoggedInUserDto> LoginAsync(LoginDto login);

    Task<LoggedInUserDto> RegisterUserAsync(UserRegisterDto registerUser);

## UserBusinessLogic.cs

Inject token service via constructor

    private readonly IUserRepository \_userRepo;

    private readonly ITokenService \_tokenService;

    public UserBusinessLogic(IUserRepository userRepo, ITokenService tokenService)

    {

        \_userRepo = userRepo;

        \_tokenService = tokenService;

    }

Add the methods

    public async Task<AppUser> GetUserRawAsync(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("User name missing");

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

        return user;

    }

    public async Task<bool> UserExists(string userName)

    {

        if(string.IsNullOrWhiteSpace(userName))

            throw new ValidationException("User name missing");

        return await \_userRepo.UserExists(userName);

    }

    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");

        //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();

        appUser.PasswordHash = hashSalt.Hash;

        appUser.PasswordSalt = hashSalt.Salt;

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

        if(!isRegister)

            throw new DataFailException("User not registerd");

        var returnUser = await \_userRepo.GetUserRawAsync(registerUser.UserName);

        if(returnUser == null)

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

        var loggedInUser = new LoggedInUserDto();

        loggedInUser.UserName = returnUser.UserName;

        loggedInUser.Guid = returnUser.Guid;

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

        return loggedInUser;

    }

    public async Task<LoggedInUserDto> LoginAsync(LoginDto login)

    {

        if (login == null)

            throw new ValidationException("Login info missing");

        var user = await \_userRepo.GetUserRawAsync(login.UserName);

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

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

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

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

        if (hashKeyLogin == null)

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

        //both are byte[]

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

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

        var loggedInUser = new LoggedInUserDto();

        loggedInUser.UserName = user.UserName;

        loggedInUser.Guid = user.Guid;

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

        return loggedInUser;

    }

# MSC.WebApi

## AccountController.cs

This will get the register end point

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MSC.Core.BusinessLogic;

using MSC.Core.DB.Entities;

namespace MSC.WebApi.Controller;

// /api/account

public class AccountController : BaseApiController

{

    private readonly IUserBusinessLogic \_userBusinessLogic;

    public AccountController(IUserBusinessLogic userBusinessLogic)

    {

        \_userBusinessLogic = userBusinessLogic;

    }

    [HttpPost("register")]

    public async Task<ActionResult<LoggedInUserDto>> Register([FromBody] UserRegisterDto registerUser)

    {

        if(await \_userBusinessLogic.UserExists(registerUser.UserName))

            return BadRequest("Username already taken");

        var loggedInUser = await \_userBusinessLogic.RegisterUserAsync(registerUser);

        if(loggedInUser == null || string.IsNullOrWhiteSpace(loggedInUser.UserName))

            return BadRequest("Unable to create registration");

        return loggedInUser;

    }

    [HttpPost("login")]

    public async Task<ActionResult<LoggedInUserDto>> Login([FromBody] LoginDto login)

    {

        var user = await \_userBusinessLogic.LoginAsync(login);

        if (user == null || string.IsNullOrWhiteSpace(user.UserName) || string.IsNullOrWhiteSpace(user.Token))

            return Unauthorized("Unable to login user");

        return Ok(user);

    }

}

# Running via Postman

## Register Result

{

    "userName": "marianasmit",

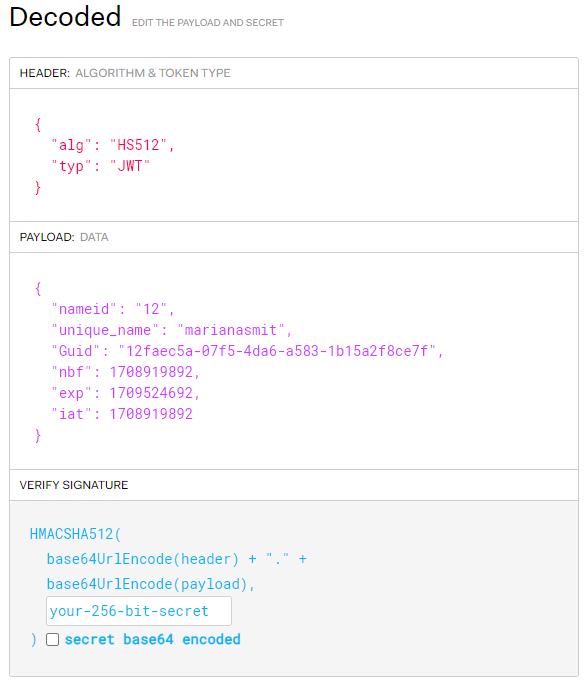
    "guid": "12faec5a-07f5-4da6-a583-1b15a2f8ce7f",

    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiIxMiIsInVuaXF1ZV9uYW1lIjoibWFyaWFuYXNtaXQiLCJHdWlkIjoiMTJmYWVjNWEtMDdmNS00ZGE2LWE1ODMtMWIxNWEyZjhjZTdmIiwibmJmIjoxNzA4OTE5ODkyLCJleHAiOjE3MDk1MjQ2OTIsImlhdCI6MTcwODkxOTg5Mn0.a-vhCyI00DHy-UosS2QFBTmK4NqX51Z5qBWKrc9o1L9azZAdA\_h9FE8T7Ek8abYYo5H00Ws7F5NgHI2V9u7BQg"

}

Check the token

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



## Login Result

Loging in with Mariana

{

    "userName": "marianasmit",

    "guid": "12faec5a-07f5-4da6-a583-1b15a2f8ce7f",

    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiIxMiIsInVuaXF1ZV9uYW1lIjoibWFyaWFuYXNtaXQiLCJHdWlkIjoiMTJmYWVjNWEtMDdmNS00ZGE2LWE1ODMtMWIxNWEyZjhjZTdmIiwibmJmIjoxNzA4OTE5OTg1LCJleHAiOjE3MDk1MjQ3ODUsImlhdCI6MTcwODkxOTk4NX0.l1z6EE8AcsAmRKSnbF-uD8v8gdUp\_XCwuLwDlfFI7K82tXQSTpKeSgT8jQ\_ihfIPpDXDWEuRuJcYKeLE-lbVqQ"

}

Check the token

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



# Authentication

## UsersController

Apply authorize attribute to the users controller

// /api/users

[Authorize]

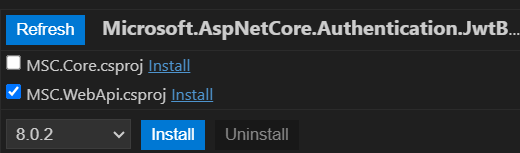
public class UsersController : BaseApiController

## Authentication Middleware

### Microsoft.AstNetCore.Authentication.JwtBearer by Microsoft NuGetPackage

Package: Microsoft.AstNetCore.Authentication.JwtBearer by Microsoft

Important: Install in both projects



### Program.cs

#### Add Authentication Service

//CORS

builder.Services.AddCors();

//Autentication

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options => {

    options.TokenValidationParameters = new TokenValidationParameters

    {

        ValidateIssuerSigningKey = true,

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

        ValidateIssuer = false,

        ValidateAudience = false

    };

});

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

#### Add Authentication Middleware

Under configure http request pipe line, add authentication

/\*\*\*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"));

app.UseAuthentication();

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

# Postman Collection

Postman collection can be found inside the root.

It is named Postman\_collection.json

Check Site-03 for details

# Cleanup Programs with Extension Methods

## MSC.Core/Extensions

### AppServiceExtensions.cs

Move the custom code from Program.cs to this extension

using System.Linq;

using System.Text;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.IdentityModel.Tokens;

using MSC.Core.BusinessLogic;

using MSC.Core.DB.Data;

using MSC.Core.Repositories;

using MSC.Core.Services;

namespace MSC.Core.Extensions;

public static class AppServiceExtensions

{

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

    {

        //\*\*\*\*\* check Program\_site03\_before\_extensions.txt for details - in the root of the project setup\*\*\*\*\*\*\*

        //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.

        services.AddDbContext<DataContext>(opt => {

            opt.UseSqlite(config.GetDefaultConnectionString(),

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

                        );

        });

        return services;

    }

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

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        return services;

    }

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

    {

        services.AddCors();

        return services;

    }

    public static string AddCorsServicePolicyBased(this IServiceCollection services, IConfiguration config)

    {

        var myAllowSpecificOrigins = "\_myAllowSpecificOrigins";

        //https://stackoverflow.com/questions/42858335/how-to-hardcode-and-read-a-string-array-in-appsettings-json

        var allowedSpecificOrigins = config.GetAllowSpecificOrigins();

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

        {

            services.AddCors(options =>

            {

                options.AddPolicy(name: myAllowSpecificOrigins,

                                policy =>

                                {

                                    policy.WithOrigins(allowedSpecificOrigins.ToArray())

                                    .AllowAnyHeader()

                                    .AllowAnyMethod();

                                });

            });

        }

        return myAllowSpecificOrigins;

    }

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

    {

        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;

    }

}

## Program.cs

* Check “Program\_site03\_before\_extensions.txt” for code used before applying extensions
* This file is in the root of the solution folder

Cleaned Program.cs file

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using MSC.Core.Extensions;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

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

IConfiguration configuration = builder.Configuration; // allows both to access and to set up the config

//DBContext

builder.Services.AddDbContext(configuration);

//add resources for DI

//cannot bind the full environment config inside the extension so keeping it here for referenc purposes

//using extension for getting the root pieces and for subsections creating the configure in root only

//builder.Services.Configure<EnvConfig>(configuration);

builder.Services.AddServices(configuration);

//CORS -- using policy based

//builder.Services.AddCorsService(configuration);

var myAllowSpecificOrigins = builder.Services.AddCorsServicePolicyBased(configuration);

//Autentication

builder.Services.AddAuthenticationService(configuration);

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

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

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

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

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

app.UseHttpsRedirection();

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

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

//using policy based cors and not simple

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

app.UseCors(myAllowSpecificOrigins);

app.UseAuthentication();

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

app.UseAuthorization();

app.MapControllers();

app.Run();