# Disable Hot Reload

Add following in Properties/launchSettings.json

A screen shot of a computer

Description automatically generated

If it does not work then issue command

>dotnet watch --no-hot-reload

# Debugging

## Set up Debugging

Open command pallet and select

.NET: Generate Assets for Build and Debug

A screenshot of a computer

Description automatically generated

This will create the following files

A screenshot of a computer

Description automatically generated

## Launch.json

Open launch.json. It has two debuggers configurations available

A screenshot of a computer program

Description automatically generated

## Run Option #1

A screenshot of a computer

Description automatically generated

## Run Option #2

Do any of the following

>dotnet run

>dotnet watch

>dotnet watch --no-hot-reload

And then

A screenshot of a computer

Description automatically generated

# Custom Exception

Create a new folder ExceptionCustom in Core folder

## DataFailException.cs

namespace OAuth2.WebApi.Core.ExceptionCustom;

public class DataFailException : Exception

{

    public DataFailException()

    {

    }

    public DataFailException(string message) : base(message)

    {

    }

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

    {

    }

}

# ExceptionMiddleware

## Core >> Dto >> ApiExceptionDto.cs

namespace OAuth2.WebApi.Core.Dto;

/// <summary>

/// Dto used by the ExceptionMiddleware to pass the response back

/// </summary>

public class ApiExceptionDto

{

    public ApiExceptionDto(int statusCode, string message = null, string details = null)

    {

        StatusCode = statusCode;

        Message = message;

        Details = details;

    }

    /// <summary>

    /// Http Status Code

    /// </summary>

    public int StatusCode { get; set; }

    /// <summary>

    /// Message

    /// </summary>

    public string Message { get; set; }

    /// <summary>

    /// Exception Details

    /// </summary>

    public string Details { get; set; }

}

## Core >> Middleware >> ExceptionMiddleware.cs

* Create anew folder Middleware in Core folder
* Create a new class ExceptionMiddleware.cs

using System.ComponentModel.DataAnnotations;

using System.Net;

using System.Text.Json;

using OAuth2.WebApi.Core.Constants;

using OAuth2.WebApi.Core.Dto;

using OAuth2.WebApi.Core.ExceptionCustom;

namespace OAuth2.WebApi.Core.Middleware;

public class ExceptionMiddleware

{

    private readonly RequestDelegate \_next;

    private readonly ILogger<ExceptionMiddleware> \_logger;

    private readonly IHostEnvironment \_environment;

    /// <summary>

    /// Receives RequestDelegate which is whats next in the middle ware pipeline

    /// </summary>

    /// <param name="next">What is next in the pipeline</param>

    /// <param name="logger">So to log the exception</param>

    /// <param name="environment">The environment development/production</param>

    public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger, IHostEnvironment environment)

    {

        \_next = next;

        \_logger = logger;

        \_environment = environment;

    }

    /// <summary>

    /// The required method to invoke the middleware

    /// </summary>

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

    /// <returns></returns>

    public async Task Invoke(HttpContext context)

    {

        try

        {

            //pass the context to the next piece of middleware

            await \_next(context);

        }

        catch (DataFailException dfe)

        {

            \_logger.LogError(dfe, dfe.Message);

            await WriteGeneralError(context, dfe, HttpStatusCode.BadRequest);

        }

        catch (ValidationException vex)

        {

            \_logger.LogError(vex, vex.Message);

            await WriteGeneralError(context, vex, HttpStatusCode.BadRequest);

        }

        catch (UnauthorizedAccessException uex)

        {

            \_logger.LogError(uex, uex.Message);

            await WriteGeneralError(context, uex, HttpStatusCode.Unauthorized);

        }

        catch (Exception ex)

        {

            \_logger.LogError(ex, ex.Message);

            await WriteSpecificError(context, ex, HttpStatusCode.InternalServerError);

        }

    }

    private async Task WriteGeneralError(HttpContext context, Exception ex, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //write

        if (\_environment.IsDevelopment())

            await context.Response.WriteAsync($"{ex.Message} \r\n {ex.StackTrace?.ToString()}");

        else

            await context.Response.WriteAsync(ex.Message);

    }

    private async Task WriteSpecificError(HttpContext context, Exception ex, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //create the response model

        ApiExceptionDto response = null;

        if (\_environment.IsDevelopment())

        {

            //development put out the exact message and stack trace

            response = new ApiExceptionDto(context.Response.StatusCode, ex.Message, ex.StackTrace?.ToString());

        }

        else

        {

            //production do not put out the exact message and stack trace

            response = new ApiExceptionDto(context.Response.StatusCode, "Internal Server Error");

        }

        //want the json responses to go as camel case

        var jsonOptions = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

        //serialize the response

        var json = JsonSerializer.Serialize(response, jsonOptions);

        //write

        await context.Response.WriteAsync(json);

    }

}

## Register Exception Middleware >> Programs.cs

Register the exception middleware in the Programs.cs

Add the using towards the top

using OAuth2.WebApi.Core.Middleware;

and then add the middleware

A screen shot of a computer program

Description automatically generated

# Base API Controller

Build Base Api Controller and it will be used by other controllers.

using Microsoft.AspNetCore.Mvc;

namespace OAuth2.WebApi.Controllers;

[ApiController]

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

public class BaseApiController : ControllerBase

{

}

## UsersController

UserController will now derive from BaseApiController.

Here is updated top part

using Microsoft.AspNetCore.Mvc;

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Controllers;

public class UsersController : BaseApiController

# Registration – First Iteration

## How to Store Passwords?

|  |  |
| --- | --- |
| **Hashing**   * One way only * You cannot calculate from a hash * Calculate the password and then compare against the hash * Multiple users may end up with same hash | **Hashing and Salting**   * Password salt will scramble the hash * No two users will have the same hash * Store the Salt in DB along with hash |

Will start with Hashing and Salting but later will switch to JWT and ASP.NET Core Identity.

## IMPORTANT: Core >> DB >> DataContext.cs

GUID is not being autogenerated for the SQLite

So comment the following sections

/\*

        This isn't working, in AppUser create the guid

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

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

            modelBuilder.Entity<AppUser>()

            .Property(x => x.Guid)

            .ValueGeneratedOnAdd();

        }

    \*/

## Core >> Entities >> AppUser.cs

* Added two new properties and also applied Required attributes
* Guid has default Guid.NewGuid() applied

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)]

    [Required]

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

    [Required]

    public string UserName { get; set; }

    [Required]

    public byte[] PasswordHash { get; set; }

    [Required]

    public byte[] PasswordSalt { get; set; }

}

### Update Database

* At this point hard deleted the migration and the DB from the folder
* And then recreated the InitialMigrations

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

> dotnet ef database update

## Core >> Dto >> HashKeyDto.cs

* Create a new folder Dto inside Core folder.
* Create a new file HashKeyDto.cs
* HashKeyDto will be used by the CryptoExtension.cs [here](#_Core_>>_Extensions) to compute the Salt and Hash.

namespace OAuth2.WebApi.Core.Dto;

public class HashKeyDto

{

    public byte[] Salt { get; set; }

    public byte[] Hash { get; set; }

}

## Core >> Dto >> UserRegisterDto

* Create a UserRegisterDto.cs in Core/Dto folder

using System.ComponentModel.DataAnnotations;

namespace OAuth2.WebApi.Core.Dto;

public class UserRegisterDto

{

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

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

    public string UserName { get; set; }

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

    [StringLength(10, MinimumLength = 4)]

    [RegularExpression("^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d).+$", ErrorMessage = "Password must have an upper case, a lower case and a number")]

    public string Password { get; set; }

}

## Core >> Extensions >> CryptoExtensions.cs

Create CryptoExtensions.cs to create two methods to compute Hash and Salt.

using System.ComponentModel.DataAnnotations;

using System.Security.Cryptography;

using System.Text;

using OAuth2.WebApi.Core.Dto;

namespace OAuth2.WebApi.Core.Extensions;

public static class CryptoExtensions

{

    /// <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))

            return null;

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

        using var hmac = new HMACSHA512();

        var hash = hmac.ComputeHash(valueBytes);

        var dto = new HashKeyDto { Salt = hmac.Key, Hash = hash };

        return dto;

    }

    /// <summary>

    /// Compute HASh for the passed in value using the passed in saltKey

    /// </summary>

    /// <param name="value"></param>

    /// <param name="saltKey"></param>

    /// <returns>HashKeyDto</returns>

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

    {

        if (string.IsNullOrWhiteSpace(value))

            throw new ValidationException("Item1 is missing");

        if (saltKey == null)

            throw new ValidationException("Item2 is missing");

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

        using var hmac = new HMACSHA512(saltKey);

        var hash = hmac.ComputeHash(valueBytes);

        var dto = new HashKeyDto { Salt = hmac.Key, Hash = hash };

        return dto;

    }

}

## Core >> Data >> Repositories

### IUserReposiory.cs

Add four new signatures

    Task<bool> SaveAllAsync();

    void Update(AppUser appUser);

    Task<bool> UserExistsAsync(string userName);

    Task<bool> RegisterAsync(AppUser appUser);

### UserRepository.cs

Implement the 4 new methods

    /// <summary>

    /// Save to DB

    /// </summary>

    /// <returns></returns>

    public async Task<bool> SaveAllAsync()

    {

        //make sure that the changes have been saved

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

        return isSave;

    }

    /// <summary>

    /// Marking the entity only that it has been modified. Must call SaveAllAsync to save

    /// </summary>

    /// <param name="appUser"></param>

    /// <exception cref="ValidationException"></exception>

    public void Update(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        //ef adds a flag to the entity that it has been modified

        \_context.Entry<AppUser>(appUser).State = EntityState.Modified;

    }

    /// <summary>

    /// Checks if the userName is taken or not

    /// </summary>

    /// <param name="userName"></param>

    /// <returns></returns>

    public async Task<bool> UserExistsAsync(string userName)

    {

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

        return isUser;

    }

    /// <summary>

    /// Add the new user

    /// </summary>

    /// <param name="appUser"></param>

    /// <returns></returns>

    /// <exception cref="ValidationException"></exception>

    public async Task<bool> RegisterAsync(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

        var isSave = await SaveAllAsync();

        return isSave;

    }

## Core >> Data >> BusinessLogic

### IUserBusinessLogic.cs

Add the new signature to Register

    Task<AppUser> RegisterAsync(UserRegisterDto registerUser);

### UserBusinessLogic.cs

Implement the new method

    public async Task<AppUser> RegisterAsync(UserRegisterDto registerUser)

    {

        if (registerUser == null || string.IsNullOrWhiteSpace(registerUser.UserName) || string.IsNullOrWhiteSpace(registerUser.Password))

            throw new ValidationException("User info missing");

        //check user not already taken

        var isUser = await \_userRepository.UserExistsAsync(registerUser.UserName);

        if (isUser)

            throw new ValidationException("Username already taken");

        //hash the password using the CryptoExtension. It will give back hash and the Salt

        var passwordHashKey = registerUser.Password.ComputeHashHmacSha512();

        if (passwordHashKey == null)

            throw new ValidationException("Unable to handle provided password");

        //convert to AppUser to register, ID and GUID will be automatically input by EF

        var appUser = new AppUser()

        {

            UserName = registerUser.UserName.ToLower(),

            PasswordHash = passwordHashKey.Hash,

            PasswordSalt = passwordHashKey.Salt

        };

        //Register user

        var isRegister = await \_userRepository.RegisterAsync(appUser);

        if (!isRegister)

            throw new DataFailException("User not registerd");

        //get user from the DB

        var returnUser = await \_userRepository.GetAppUserAsync(registerUser.UserName);

        if (returnUser == null)

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

        return returnUser;

    }

## AccountController

Create a new AccountController

Add a single end point to receive the username and password to register

using Microsoft.AspNetCore.Mvc;

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Dto;

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Controllers;

public class AccountController : BaseApiController

{

    private readonly IUserBusinessLogic \_userBL;

    public AccountController(IUserBusinessLogic userBL)

    {

        \_userBL = userBL;

    }

    [HttpPost("register", Name = "RegisterUser")] // api/account/register

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

    {

        var user = await \_userBL.RegisterAsync(registerUser);

        if (user == null || user.Id <= 0)

            return BadRequest("Unable to Create User");

        return Ok(user);

    }

}

## Run via Postman

A screenshot of a computer

Description automatically generated

### Users Created

Check the text file [Registration Simple - Users Created via Postman.txt](Registration%20Simple%20-%20Users%20Created%20via%20Postman.txt) in Documents folder.

A screenshot of a computer

Description automatically generated

# Login – First Attempt

## Core >> Extensions >> ByteExtensions.cs

* Create a new extension to compare the byteArray.
* Remember while registering we stores PasswordHash and PasswordSalt in the DB.
* We will be receiving the password for login and then will check against the DB by computing the password hash using the salt stored in the DB

namespace OAuth2.WebApi.Core.Extensions;

public static class ByteExtensions

{

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

    {

        var areEqual = a.SequenceEqual(b);

        return areEqual;

    }

}

## Core >> Dto >> LoginDto

Create a LoginDto

using System.ComponentModel.DataAnnotations;

namespace OAuth2.WebApi.Core.Dto;

public class LoginDto

{

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

    public string UserName { get; set; }

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

    public string Password { get; set; }

}

## Core >> Data >> BusinessLogic

### IUserBusinessLogic.cs

Create a new signature

    Task<AppUser> LoginAsync(LoginDto loginInfo);

### UserBusinessogic.cs

Implement the new method

    public async Task<AppUser> LoginAsync(LoginDto loginInfo)

    {

        if (loginInfo == null)

            throw new ValidationException("Login info missing");

        var appUser = await \_userRepository.GetAppUserAsync(loginInfo.UserName);

        if (appUser == null || appUser.PasswordSalt == null || appUser.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 passwordHashKey = loginInfo.Password.ComputeHashHmacSha512(appUser.PasswordSalt);

        if (passwordHashKey == null)

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

        //both are byte[]

        if (!passwordHashKey.Hash.AreEqual(appUser.PasswordHash))

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

        return appUser;

    }

## AccountController.cs

Add a new end point for login

### Login

    /// <summary>

    /// /api/account/login

    /// </summary>

    /// <param name="loginInfo"></param>

    /// <returns></returns>

    [HttpPost("login", Name = "LoginUser")]

    public async Task<ActionResult<AppUser>> Login([FromBody] LoginDto loginInfo)

    {

        var user = await \_userBL.LoginAsync(loginInfo);

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

            return Unauthorized("Unable to login user");

        return Ok(user);

    }

## Testing with Postman

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

# JSON Web Token (JWT)

## What is JWT?

Self-contained and can contain

* Credentials
* Claims
* Other Information
* It is a long string, has 3 parts separated by a period
  + Header: algorithm and type of toke. Algorithm is used to encrypt the signature in the verify part of the token
  + Payload – data:
    - can contain info about claims, roles and credentials
    - 3 time stamps
      * mbf: cannot be used before this time
      * exp: expiry date and time for the token
      * iat: token issued date and time
  + Verify signature:
    - token signature is encrypted by the server using a secure key that never leaves the server.
    - This is the only part that is encrypted

## How Token authentication works?

A screenshot of a computer screen

Description automatically generated

## Benefits of JWT

* No session to manage
* Portable – a single token can be used woth multiple backends
* No cookie required – mobile friendly
* Performance – once token is issued there is no need to make a database request to verify a user’s authentication

## Add Package System.IdentityModel.Tokens.Jwt

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

## Adding Token Key

### appsettings.json and appsettings.Development.json

* Add the following key to both appsettings.json and appsettings.Development.json
* The key value could be any long string.

  "TokenKey": "7NCyQkWBsqV3bZsT4qShUN6qzpWUjmRs"

### Core >> Constants >> ConsfigKeyConstants.cs

Open Open Core >> Constants >> ConsfigKeyConstants.cs and the “TokenKey” name to it

    public const string TokenKey = "TokenKey";

### Core >> Extensions >> ConfigExtensions.cs

Open core >> Extensions >> ConfigExtensions.cs and add a new method to get the TokenKey value

    public static string GetTokenKey(this IConfiguration config)

    {

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

        return tokenKey;

    }

## TokenService

Create a new folder Services in the Core folder.

### ITokenService.cs

using OAuth2.WebApi.Core.Entities;

namespace OAuth2.WebApi.Core.Services;

public interface ITokenService

{

    string CreateToken(AppUser user);

}

### TokenService.cs

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.IdentityModel.Tokens;

using OAuth2.WebApi.Core.Entities;

using OAuth2.WebApi.Core.Extensions;

namespace OAuth2.WebApi.Core.Services;

public class TokenService : ITokenService

{

    private readonly IConfiguration \_configuration;

    /// <summary>

    /// key will remain the server and it will never go to the client

    /// </summary>

    private readonly SymmetricSecurityKey \_key;

    public TokenService(IConfiguration configuration)

    {

        \_configuration = configuration;

        //get the TokenKey from the config

        var tokenKey = \_configuration.GetTokenKey();

        if (string.IsNullOrWhiteSpace(tokenKey))

            throw new Exception("TokenKey missing");

        //convert tokenKey to bytesArray

        var tokenKeyByteArray = Encoding.UTF8.GetBytes(tokenKey);

        //create key

        \_key = new SymmetricSecurityKey(tokenKeyByteArray);

    }

    /// <summary>

    /// Create JWT Token

    /// </summary>

    /// <param name="user"></param>

    /// <returns></returns>

    public string CreateToken(AppUser user)

    {

        if (user == null)

            throw new Exception("User info missing");

        //claims

        //check folowing url for all available fields. We can create our own as well

        //[https://learn.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.jwt.jwtregisteredclaimnames?view=msal-web-dotnet-latest](https://learn.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.jwt.jwtregisteredclaimnames?view=msal-web-dotnet-latest%20)

        var claims = new List<Claim>()

        {

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

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

        };

        //signing credentials with key

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

        //describe token

        var tokenDescriptor = new SecurityTokenDescriptor()

        {

            Subject = new ClaimsIdentity(claims),

            Expires = DateTime.Now.AddDays(7), //typically shorter value

            SigningCredentials = credentials

        };

        //token handler

        var tokenHandler = new JwtSecurityTokenHandler();

        //token

        var token = tokenHandler.CreateToken(tokenDescriptor);

        var writeToken = tokenHandler.WriteToken(token);

        return writeToken;

    }

}

### Add TokenService as Service

* We have already created an extension for adding the service under [0003 Extensions-CORS.docx](0003%20Extensions-CORS.docx).
* Add it to method RegisterServices

First add the using statement

using OAuth2.WebApi.Core.Services;

and then add the TokenService as a service

A screen shot of a computer program

Description automatically generated

## Core >> Dto >> LoginResponseDto.cs

* Create a new class LoginResponseDto.cs in Core/Dto folder

namespace OAuth2.WebApi.Core.Dto;

public class LoginResponseDto

{

    public string UserName { get; set; }

    public Guid Guid { get; set; }

    public string Token { get; set; }

}

## Core >> Data >> BusinessLogic

### IUserBusinessLogic.cs

Return LoginResposneDto for methods LoginAsync and RegisterAsync

A screenshot of a computer screen

Description automatically generated

### UserBusinessLogic.cs

Inject the tokenService

A screenshot of a computer program

Description automatically generated

#### New Private Method BuildLoginResponse

Create a new private method BuildLoginResponse.

    private LoginResponseDto BuildLoginResponse(AppUser appUser)

    {

        var loginResponse = new LoginResponseDto()

        {

            UserName = appUser.UserName,

            Guid = appUser.Guid,

            Token = \_tokenService.CreateToken(appUser)

        };

        return loginResponse;

    }

#### Update RegisterAsync Method

* Change the return model to LoginResponseDto

A screen shot of a computer program

Description automatically generated

#### Update LoginAsync Method

* Change the return model to LoginResponseDto

A screen shot of a computer

Description automatically generated

## AccountController.cs

### Register EndPoint

Return LoginResponseDto and also update the check. Here is the updated code

    [HttpPost("register", Name = "RegisterUser")] // api/account/register

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

    {

        var user = await \_userBL.RegisterAsync(registerUser);

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

            return BadRequest("Unable to Create User");

        return Ok(user);

    }

### Login EndPoint

Return LoginResponseDto. Here is the updated code

    [HttpPost("login", Name = "LoginUser")]

    public async Task<ActionResult<LoginResponseDto>> Login([FromBody] LoginDto loginInfo)

    {

        var user = await \_userBL.LoginAsync(loginInfo);

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

            return Unauthorized("Unable to login user");

        return Ok(user);

    }

## Testing Postman

|  |  |
| --- | --- |
| Login A screenshot of a computer  Description automatically generated | Register |

## Checking the Token

<https://jwt.io/>

A screenshot of a computer

Description automatically generated

# Authentication

## Add Package

Add package Microsoft.AspNetCore.Authentication.JwtBearer

Check 0001 Project Setup.docx section Adding Package for details

## UsersController.cs

Apply [Authorize] attribute to the methods that will be behind login like following

A screen shot of a computer

Description automatically generated

Applied to all 3 GetUser methods

## Register Authentication Service and Use Middleware

### Core >> Extensions >> ServiceExtensions.cs

#### RegisterAuthentication

Add a new extension method to register authentication as service

Add using statements

A screenshot of a computer screen

Description automatically generated

Add new method

    /// <summary>

    /// Register authentications service

    /// </summary>

    /// <param name="services"></param>

    /// <param name="configuration"></param>

    public static void RegisterAuthentication(this IServiceCollection services, IConfiguration configuration)

    {

        //get the token from config and change to byte array

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

        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

                .AddJwtBearer(options =>

                {

                    options.TokenValidationParameters = new TokenValidationParameters()

                    {

                        ValidateIssuerSigningKey = true,

                        IssuerSigningKey = new SymmetricSecurityKey(tokenKey),

                        ValidateIssuer = false,

                        ValidateAudience = false

                    };

                });

    }

### Programs.cs

#### Register Authentication Service using above extension method

A screen shot of a computer code

Description automatically generated

#### Add Authentication Middleware

A screen shot of a computer

Description automatically generated

## Postman Testing

### Unauthorize Result

Get the user by ID and it will result in 401

A screenshot of a computer

Description automatically generated

### Test with Token

Login as Tyler

A screenshot of a computer

Description automatically generated

* Copy the Token
* Go to the Get Example again
* Go to Auth tab
* Select type as Bearer
* Paste the token and try again

A screenshot of a computer

Description automatically generated