**Section07 JWT Bearer**

**Notes:-**

**1-we will apply JWT Bearer which is similar with the claim based principle but with nice approach**

**2-JWT web token does not relate to the OAuth directly, but it has become the standard of the authorization token standard**

**(Good implementation of the authorization token but the JWT used to expose the public information and still maintain the validity of the token to make sure that the information put on the token is the same information that put by the server not change at all)**

**3-in the previous sections we use the authentication middleware as authorization layer , but now we have to impelement JwtBearer instead of the OAuth authentication**

**Steps:-**

**1-create new project called OAuth>Server**

**2-install the nugget package called**

**Microsoft.AspNetCore.Authentication.JwtBearer**

**3-JWT is stateless which means that each time you make request you have to include JWT token as authorization header attribute to make JWT barrer middleware validate to the Toke if its validate and not expired**

**4-on the Models > JWTSettings.cs to set the JWT barrier**

**public static class JWTSettings{**

**public const string Issuer = Audiance;**

**public const string Audiance = "https://localhost:44348/";**

**public const string Secret = "not\_too\_short\_secret\_key\_or\_otherwise\_it\_will\_Generate\_error";}**

**5-on the startup.cs we set the configuration of the JWT barrier**

**public class Startup{**

**public Startup(IConfiguration configuration){Configuration = configuration;}**

**public IConfiguration Configuration { get; }**

**// This method gets called by the runtime. Use this method to add services to the container.**

**public void ConfigureServices(IServiceCollection services){**

**services.AddAuthentication("OAuth")**

**.AddJwtBearer("OAuth", config =>{**

**var secretBytes = Encoding.UTF8.GetBytes(JWTSettings.Secret);**

**var key = new SymmetricSecurityKey(secretBytes);**

**config.Events = new JwtBearerEvents(){**

**OnMessageReceived = context =>{**

**if (context.Request.Query.ContainsKey("access\_token")){**

**context.Token = context.Request.Query["access\_token"];}**

**return Task.CompletedTask;}};**

**//the JWT Baerer will check to the issuer and audiance and the key used to decrypt**

**//the JWT Token passed through the request to check if Token is valid**

**config.TokenValidationParameters = new TokenValidationParameters(){**

**ValidIssuer = JWTSettings.Issuer,**

**ValidAudience = JWTSettings.Audiance,**

**IssuerSigningKey = key};});**

**services.AddControllersWithViews();}**

**// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.**

**public void Configure(IApplicationBuilder app, IWebHostEnvironment env){**

**if (env.IsDevelopment()){app.UseDeveloperExceptionPage();}**

**app.UseRouting();**

**app.UseAuthentication();**

**app.UseAuthorization();**

**app.UseEndpoints(endpoints =>{endpoints.MapDefaultControllerRoute();});}}**

**6-on the HomeController.cs we set the authenticate action to generate JWT Token**

**using Microsoft.AspNetCore.Authorization;**

**using Microsoft.AspNetCore.Mvc;**

**using Microsoft.Extensions.Logging;**

**using Microsoft.IdentityModel.Tokens;**

**using Microsoft.VisualBasic;**

**using Server.Models;**

**using System;**

**using System.IdentityModel.Tokens.Jwt;**

**using System.Security.Claims;**

**using System.Text;**

**namespace Server.Controllers{**

**public class HomeController : Controller{**

**private readonly ILogger<HomeController> \_logger;**

**public HomeController(ILogger<HomeController> logger){\_logger = logger;}**

**public IActionResult Index(){return View();}**

**[Authorize]**

**public IActionResult Secret(){return View();}**

**public IActionResult Authenticate(){**

**var claims = new[]{**

**new Claim(JwtRegisteredClaimNames.Sub,"some\_id"),**

**new Claim("granny","cookie")};**

**var secretBytes = Encoding.UTF8.GetBytes(JWTSettings.Secret);**

**var key = new SymmetricSecurityKey(secretBytes);**

**//we will use SHA 256 into generate the certificate**

**var algorithim = SecurityAlgorithms.HmacSha256;**

**var signingCredentials = new SigningCredentials(key, algorithim);**

**var token = new JwtSecurityToken(**

**JWTSettings.Issuer,**

**JWTSettings.Audiance,**

**claims**

**,notBefore:DateTime.Now**

**,expires:DateTime.Now.AddHours(1)**

**, signingCredentials);**

**var tokenJson = new JwtSecurityTokenHandler().WriteToken(token);**

**return Ok(new { access\_token = tokenJson});}**

**public IActionResult Decode(string part){**

**var bytes = Convert.FromBase64String(part);**

**return Ok(Encoding.UTF8.GetString(bytes));}}}**

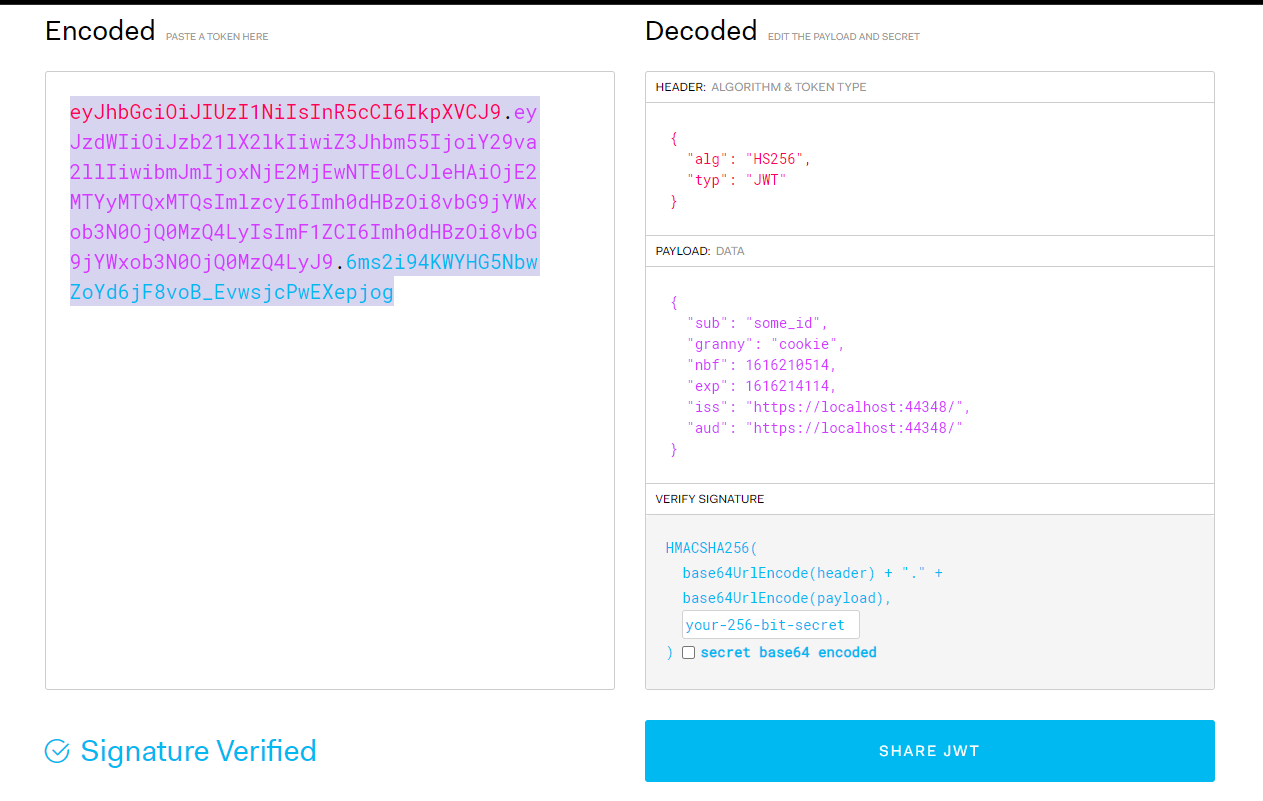
**//you can see that when you hit the request**

**https://localhost:44348/Home/Authenticate**

**{"access\_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzb21lX2lkIiwiZ3Jhbm55IjoiY29va2llIiwibmJmIjoxNjE2Mjg5MjcxLCJleHAiOjE2MTYyOTI4NzEsImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyJ9.tH5Ao61eHsrDTaIRreDgI6B\_l1Pmu3zft0ZZBiyHrlc"}**

**You can get the JWT token generated and pass to the**

[JSON Web Tokens - jwt.io](https://jwt.io/)



**We see that the first part red section refer to the algorithim used to generate key which used HMAC 256 in order to generate certificate by pass key and algorithim**

**Second part is the claims passed through the JWT Barrer (bing section)**

**Third part is the signature that make by the first and second sections**

**So if anybody change the JWT token it will make invalid token**

**We can pass the JWT token either with query string or with the**

[**https://localhost:44348/Home/Secret**](https://localhost:44348/Home/Secret)

**on header**

**Authorization**

**Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzb21lX2lkIiwiZ3Jhbm55IjoiY29va2llIiwibmJmIjoxNjE2MjExMjMwLCJleHAiOjE2MTYyMTQ4MzAsImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyJ9.hFPAHrIVPiVmCqGg\_GcP9b-3ia\_O6VL3qL6lImnd3Zw**

**Or with**

**https://localhost:44348/Home/Secret?access\_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzb21lX2lkIiwiZ3Jhbm55IjoiY29va2llIiwibmJmIjoxNjE2MjEyOTc0LCJleHAiOjE2MTYyMTY1NzQsImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjQ0MzQ4LyJ9.BnSBs90SoPmWKRk\_csrxHRF-bQGCdIA26s-KEkQ9I4A**

**you can decode the JWT header section and see the generated token as below**

**https://localhost:44348/Home/Decode?part=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9**