Written by Salil Ponde on 28 Dec 2018

JWT Authentication and Authorisation in ASP.NET Core Web API

* *To understand this article you should have basic knowledge of creating Web APIs in ASP.NET Core*
* *This code in the article has been tested on ASP.NET Core version 2.2*
* *The source code for this article is available on GitHub at*[*https://github.com/salslab/AspNetCoreJwtSample.git*](https://github.com/salslab/AspNetCoreJwtSample.git)



One of the popular methods to implement authentication in Web APIs today is JWT (JSON Web Token) authentication. In this article we will look at what a JSON Web Token is, how we can issue these tokens and how we can use them to implement authentication and authorisation in ASP.NET Core Web APIs.

What is a JSON Web Token

A JSON Web Token consists of three parts as below which are delimited by dots.

**HEADER** . **PAYLOAD** . **SIGNATURE**

Lets look at what each part means.

Payload

A payload consists of *claims* which are nothing but name-value pairs. Below is an example of a payload.

{

"name": "Fred Bloggs",

"company": "Microsoft",

"role": "Administrator"

}

This payload consists of three claims - name, company and role. These claims are details of a person named Fred Bloggs. So Fred Bloggs is called the *subject* of this token. You can define as many claims you want for the subject and choose their names as per your choice.

In addition to the details of the subject the token can also consist of claims which hold details of the token itself. Many such commonly used claims have standard names and are called [Registered Claim Names](https://tools.ietf.org/html/rfc7519#section-4.1). Some of the commonly used registered claim names are:

* **Issued At (iat)** - The time when the token was created
* **Expires (exp)** - The token expiry time
* **Issuer (iss)** - Who has issued the token
* **Audience (aud)** - Who is the token intended for

Lets add these claims to our payload.

{

"name": "Fred Bloggs",

"company": "Microsoft",

"role": "Administrator",

"iat": "1516239022",

"exp": "1516240022",

"iss": "[salslab.com](http://salslab.com/)",

"aud": "[salslab.com](http://salslab.com/)"

}

The times are represented as seconds since Unix Epoch. We don't need to worry about the conversion as the .NET classes will take care of that for us.

The above JSON is encoded as Base64 and stored in the PAYLOAD part of the token.

Header

The token is signed using a MAC algorithm. You are free to choose the algorithm you want to use for signing. We need to store the name of the algorithm (*alg*) in the header of the token. In addition to this we need to store the type (*typ*) of the token which in our case is JWT.

If we use HMACSHA256 for signing, our JWT header would like like below.

{

"alg": "HS256",

"typ": "JWT"

}

The above JSON is encoded as Base64 and stored in the HEADER part of the token.

Signature

The Base64 encoded header and payload along with the delimiting dot character are signed using an algorithm such as HMACSHA256. This signature is stored in the SIGNATURE part of the token and can be used to verify the integrity of the header and payload.

Note that the data in the JWT we have seen now is not encrypted. So you should NOT store secrets in this token without first encrypting them.

You can learn more about JWT on [jwt.io](https://jwt.io/) or refer to [RFC 7519](https://tools.ietf.org/html/rfc7519).

How JWT authentication works

The most common JWT authentication workflow is as follows.

1. The user authenticates with the server by providing his credentials to the server, for example, username and password.
2. If the credentials are correct the server issues a JWT token to the user.
3. The user sends this JWT token along with the requests which require authentication. The token is usually passed in the *Authorization* HTTP header of the request. The header looks like below.
4. Authorization: Bearer JWT\_TOKEN\_HERE

The server verifies the signature of the token to make sure the payload and header is not tampered and also ensures that the token has not expired. If the verification succeeds the authentication is considered successful.

The above communication should always be done over HTTPS to avoid man-in-the-middle attacks.

How to issue a JWT in ASP.NET Core

You can create JWTs in ASP.NET Core using the *JwtSecurityTokenHandler* class. Below is the *Login* method from [AccountService.cs](https://github.com/salslab/AspNetCoreJwtSample/blob/master/ASPNetCoreJWTSample/Services/AccountService.cs) in our sample application. For demonstration purpose the user data is hardcoded in the class and passwords are stored in plain text. For real applications passwords should be securely hashed and user data should be securely stored (preferably using ASP.NET Core Identity).

private readonly IEnumerable \_users = new List

{

new User { Id = 1, Username = "fred", Password = "123", Role = "Administrator"},

new User { Id = 2, Username = "alice", Password = "456", Role = "Accountant"},

new User { Id = 3, Username = "joe", Password = "789", Role = "Guest"},

};

public string Login(LoginDto loginDto)

{

var user = \_users.Where(x => x.Username == loginDto.Username && x.Password == loginDto.Password).SingleOrDefault();

if (user == null)

{

return null;

}

var signingKey = Convert.FromBase64String(\_configuration["Jwt:SigningSecret"]);

var expiryDuration = int.Parse(\_configuration["Jwt:ExpiryDuration"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Issuer = null, // Not required as no third-party is involved

Audience = null, // Not required as no third-party is involved

IssuedAt = DateTime.UtcNow,

NotBefore = DateTime.UtcNow,

Expires = DateTime.UtcNow.AddMinutes(expiryDuration),

Subject = new ClaimsIdentity(new List {

new Claim("userid", user.Id.ToString()),

new Claim("role", user.Role)

}),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(signingKey), SecurityAlgorithms.HmacSha256Signature)

};

var jwtTokenHandler = new JwtSecurityTokenHandler();

var jwtToken = jwtTokenHandler.CreateJwtSecurityToken(tokenDescriptor);

var token = jwtTokenHandler.WriteToken(jwtToken);

return token;

}

If no third party is involved in your system, for example, if you are developing the Web API and the front-end for the web application yourself, then populating *Issuer*and *Audience* is optional.

We are adding *userid* and *role* claims to the token which we can use for authorisation. The token is signed using HMAC SHA256 Signature algorithm with the key stored in the configuration file.

The key should be at least 64 bytes long. In our sample application you can generate a random key using the *GenerateKey* action in the [KeyGenController](https://github.com/salslab/AspNetCoreJwtSample/blob/master/ASPNetCoreJWTSample/Controllers/KeyGenController.cs).

[HttpGet]

public IActionResult GenerateKey()

{

using (var rng = new RNGCryptoServiceProvider())

{

var key = new byte[64];

rng.GetBytes(key);

return Ok(Convert.ToBase64String(key));

}

}

The *Login* method in *AccountService* is used in the *Login* action of the [AccountController](https://github.com/salslab/AspNetCoreJwtSample/blob/master/ASPNetCoreJWTSample/Controllers/AccountController.cs) as below.

[HttpPost]

public IActionResult Login(LoginDto loginDto)

{

var jwtToken = \_accountService.Login(loginDto);

if (jwtToken == null)

{

return Unauthorized();

}

return Ok(jwtToken);

}

The client can make a POST request to this method with the username and password to obtain the JWT token. This token must be passed in the *Authorization*HTTP header of the requests which need authentication.

How to authenticate a JWT in ASP.NET Core

We have seen above how we can issue a JWT. Now we need to setup the system such that it can validate the JWT passed in the *Authorization* HTTP header in the requests. This can be done as shown below in [Startup.cs](https://github.com/salslab/AspNetCoreJwtSample/blob/master/ASPNetCoreJWTSample/Startup.cs).

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient();

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

var signingKey = Convert.FromBase64String(Configuration["Jwt:SigningSecret"]);

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(signingKey)

};

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseAuthentication();

app.UseMvc();

}

As we have not set *Issuer* and *Audience* in our token we should set *ValidateIssuer*and *ValidateAudience* to false in the token validation parameters.

To test our setup the sample application provides [InvoiceController.cs](https://github.com/salslab/AspNetCoreJwtSample/blob/master/ASPNetCoreJWTSample/Controllers/InvoiceController.cs) with the below actions.

[HttpGet]

public IActionResult View()

{

return Ok("You can view invoices!");

}

[Authorize(Roles = "Administrator,Accountant")]

[HttpGet]

public IActionResult Create()

{

var userIdClaim = HttpContext.User.Claims.Where(x => x.Type == "userid").SingleOrDefault();

return Ok($"Your User ID is {userIdClaim.Value} and you can create invoices!");

}

[Authorize(Roles = "Administrator")]

[HttpGet]

public IActionResult Delete()

{

var userIdClaim = HttpContext.User.Claims.Where(x => x.Type == "userid").SingleOrDefault();

return Ok($"Your User ID is {userIdClaim.Value} and you can delete invoices!");

}

The *View* action does not require any authentication.

The *Create* action can only be accessed by users having Administrator or Accountant roles. So you will be able to call this action only if you login as *fred* or *alice*.

The *Delete* action can only be accessed by users having Administrator role. So you will be able to call this action only if you login as *fred*.

Summary

In this article we learnt what a JSON Web Token is and how we can create these tokens in ASP.NET Core. We generated a random signing key for signing our tokens.

Later we saw how we can authorise controller actions using roles.

The sample application does not use HTTPS. But your production applications should always use HTTPS.

**About the Author**

Salil Ponde is software developer, blogger and DevOps enthusiast.  
He has over 11 years of experience in the industry.

Top of Form

Join the discussion…

[](https://disqus.com/by/disqus_gDPFOZZ6Ja/)

[**maddy**](https://disqus.com/by/disqus_gDPFOZZ6Ja/) • [2 months ago](https://salslab.com/a/jwt-authentication-and-authorisation-in-asp-net-core-web-api#comment-4344410185)

Hi,this is really helpful. But i get a 404 error when i try to run it. Did you include a valuescontroller in your project as well?

[](https://disqus.com/by/salilponde/)

[**Salil Ponde**](https://disqus.com/by/salilponde/) **Mod** [maddy](https://salslab.com/a/jwt-authentication-and-authorisation-in-asp-net-core-web-api#comment-4344410185) • [2 months ago](https://salslab.com/a/jwt-authentication-and-authorisation-in-asp-net-core-web-api#comment-4344424911)

There is no ValuesController in the project. The git repository URL is present at the top of the article.

Which URL is giving you 404?

Once you run the project try doing: GET http://localhost:5000/api/invoice/create  
That should give you 401 Unauthorised.

Now try doing: POST http://localhost:5000/api/account/login with body set to { username: "fred", password: "123" }. You will get the JWT in response.

Now again make a request: GET http://localhost:5000/api/invoice/create and set Authorization header to: Bearer JWT\_TOKEN\_HERE

Now you should get a 200 OK response as "Your User ID is 1 and you can create invoices!"

[](https://disqus.com/by/disqus_gDPFOZZ6Ja/)

[**maddy**](https://disqus.com/by/disqus_gDPFOZZ6Ja/) [Salil Ponde](https://salslab.com/a/jwt-authentication-and-authorisation-in-asp-net-core-web-api#comment-4344424911) • [2 months ago](https://salslab.com/a/jwt-authentication-and-authorisation-in-asp-net-core-web-api#comment-4344453237)

In postman? I was running in visual studio to check the output. The launch url set to api/values mostly is giving that error i suppose. I should get the token as output if i run the code right?