MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
AUG 6, 2018 3:00 AM PDT

How to use JWT tokens in ASP.Net Core 2

Take advantage of JSON Web Tokens to implement a loosely coupled security model in your ASP.Net Core applications

ASP.Net Core is an open source, cross-platform, lean, and modular framework for building high-performance web applications. Unlike earlier versions of the framework, ASP.Net Core 2 provides support for JSON Web Tokens. In this article, we'll draw on ASP.Net Core 2.x to see how JWT tokens can be used in a typical ASP.Net Core web application.

What are JWT tokens?

JSON Web Tokens (or JWTs for short) are very popular in the web development community these days. JWT is an open standard in which the sender and receiver can communicate via JSON in a secure manner. JWT tokens are typically used for authenticating and authorizing users. JWT tokens are comprised of three parts:

1. Header —provides metadata about the type of data and the algorithm used to encrypt the data being transferred
2. Payload—the actual data represented in JSON format
3. Signature—used to validate the integrity of the data being transferred

Create a new ASP.Net Core project

First let's create an ASP.Net Core project in Visual Studio. Assuming that .Net Core is installed in your system, follow these steps to create an ASP.Net Core application in Visual Studio 2017.

1. In the Visual Studio IDE, click on File > New > Project.
2. Select "ASP.Net Core Web Application (.Net Core)" from the list of templates displayed.
3. Specify a name for the project.
4. Click OK to save.
5. In the "New .Net Core Web Application…" window, select "Web API."
6. Ensure that "Enable Docker Support" is unchecked.
7. Select "No Authentication" as we won't be using authentication in this example.
8. Click OK.

When you click OK, a new ASP.Net Core project will be created containing an example Controller to build and execute RESTful HTTP services. The default controller will be named ValuesController.

Install JWT bearer authentication middleware

If you want to authenticate users using JWT, you will need to install the following package via the NuGet Package Manager UI in Visual Studio.

```
Microsoft.AspNetCore.Authentication.JwtBearer
```

Alternatively, you can type in the following command in the NuGet Package Manager Console.

```
> Install-Package Microsoft.AspNetCore.Authentication.JwtBearer -Version 2.0.0
```

You can implement basic authentication with JWT in ASP.Net Core fairly easily—it is simple. After you install the necessary middleware package, i.e. Microsoft.AspNetCore.Authentication.JwtBearer, just follow the steps outlined below.

What IT Leaders Need to Know About Data Science

Data Analytics Tools: Top 10 Pricing Guide

3 Ways To Deliver Best-in-Class Field Service Experience

Add the JWT bearer to your ConfigureServices method

Now for the code. The first step is to make a call to the AddAuthentication extension method in the ConfigureServices method of the Startup class as shown in the code snippet below.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddAuthentication().AddJwtBearer(options => {
                options.Audience ="http://localhost:34924/";
                options.Authority ="http://localhost:34925/";
            });
            services.AddMvc();
        }
```

Note that the ConfigureServices method is the method used to add services to the services container, making them available in your application via dependency injection. ConfigureServices will be called automatically by the runtime whenever the API is in executed.

Specify the token validation parameters for AddAuthentication

The next thing you should do is update the ConfigureServices method and specify the necessary validation parameters for the AddAuthentication method. Here is a quick rundown of the parameters of the AddAuthentication method and what they are used for.

1. Audience—used to specify the intended recipient of the incoming token

2. Authority—represents the address of the token issuing authority, i.e., the authentication server
3. AutomaticAuthenticate—used to specify if the user defined by the token should be logged in automatically
4. RequireHttpsMetadata—used to specify if the JWT token should be transferred only over HTTPS (note we aren't using HTTPS in our example)

The following code listing illustrates the updated ConfigureServices method. Note how the TokenValidationParameters have been used.

```
public void ConfigureServices(IServiceCollection services)
    {
      services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
          options.TokenValidationParameters = new TokenValidationParameters
          {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = Configuration["Jwt:Issuer"],
            ValidAudience = Configuration["Jwt:Issuer"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuratio
          };
        });
      services.AddMvc();
    }
```

Configure the JWT authentication service

Next you can configure JWT-based authentication service in the appsettings.json file as shown in the code snippet below.

```
"Jwt": {
    "Key": "IDGSecretKey",
    "Issuer": "http://localhost:34924/"
  }
```

Lastly, you should make a call to the UseAuthentication method in the Configure method as shown in the code listing below.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseAuthentication();
    app.UseMvc();
}
```

This was an introductory article on how we can leverage JWT bearer token authentication in ASP.Net Core. I will discuss JWTs in more detail—including how to generate JWT tokens and how to encrypt the data—in future articles here.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

Tim O'Reilly: the golden age of the programmer is over

Edge computing archetypes are emerging, and they are not pretty

Python developers want static typing

Amazon Braket: Get started with quantum computing

What’s new in Kubernetes 1.20

Running microservices on Google Cloud Platform

The decline of Heroku

Red Hat Enterprise Linux takes aim at edge

What is a computational storage drive? Much-needed

How IT priorities are shifting during the COVID-19 crisis

The 24 highest paying developer roles in 2020