AUGUST 14 2018

ASP.NET Core 2.2 - JWT Authentication Tutorial with Example API

Tutorial built with **ASP.NET Core 2.2**

In this tutorial we'll go through a simple example of how to implement JWT (JSON Web Token) authentication in an ASP.NET Core 2.2 API with C#.

The example API has just two endpoints/routes to demonstrate authenticating with JWT and accessing a restricted route with JWT:

- `/users/authenticate` - public route that accepts HTTP POST requests containing the username and password in the body. If the username and password are correct then a JWT authentication token and the user details are returned.
- `/users` - secure route that accepts HTTP GET requests and returns a list of all the users in the application if the HTTP Authorization header contains a valid JWT token. If there is no auth token or the token is invalid then a 401 Unauthorized response is returned.

The tutorial project is available on GitHub at https://github.com/cornflourblue/aspnet-core-jwt-authentication-api (https://github.com/cornflourblue/aspnet-core-jwt-authentication-api).

Update History:

- 08 Jan 2019 - Updated to **ASP.NET Core 2.2**. For details of the exact changes that were required to update from ASP.NET Core 2.1 to 2.2 see this commit (https://github.com/cornflourblue/aspnet-core-jwt-authentication-api/commit/c662aaa146eb3655089a9d93bdd0fa1f52afe2f6) on GitHub.
- 14 Aug 2018 - Built with **ASP.NET Core 2.1**. The code for this version of the tutorial is tagged on GitHub and available at https://github.com/cornflourblue/aspnet-core-jwt-authentication-api/releases/tag/v2.1 (https://github.com/cornflourblue/aspnet-core-jwt-authentication-api/releases/tag/v2.1).

Tools required to run the ASP.NET Core 2.2 JWT Example Locally

To develop and run ASP.NET Core applications locally, download and install the following:

- .NET Core SDK (https://www.microsoft.com/net/download/core) - includes the .NET Core runtime and command line tools
- Visual Studio Code (https://code.visualstudio.com/) - code editor that runs on Windows, Mac and Linux
- C# extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp) for Visual Studio Code - adds support to VS Code for developing .NET Core applications

Running the ASP.NET Core JWT Authentication API Locally

1. Download or clone the tutorial project code from https://github.com/cornflourblue/aspnet-core-jwt-authentication-api (https://github.com/cornflourblue/aspnet-core-jwt-authentication-api)
2. Start the api by running `dotnet run` from the command line in the project root folder (where the WebApi.csproj file is located), you should see the message `Now listening on: http://localhost:4000`. You can test the api directly using an application such as Postman (https://www.getpostman.com/) or you can test it with one of the single page applications below.

NOTE: You can also start the application in debug mode in VS Code by opening the project root folder in VS Code and pressing F5 or by selecting Debug -> Start Debugging from the top menu. Running in debug mode allows you to attach breakpoints to pause execution and step through the application code.

Running an Angular 6 client app with the ASP.NET Core JWT Auth API

For full details about the example Angular 6 application see the post Angular 6 - JWT Authentication Example & Tutorial (/post/2018/05/23/angular-6-jwt-authentication-example-tutorial). But to get up and running quickly just follow the below steps.

1. Download or clone the Angular 6 tutorial code from https://github.com/cornflourblue/angular-6-jwt-authentication-example (https://github.com/cornflourblue/angular-6-jwt-authentication-example)
2. Install all required npm packages by running `npm install` from the command line in the project root folder (where the package.json is located).
3. Remove or comment out the line below the comment `// provider used to create fake backend` located in the `/src/app/app.module.ts` file.
4. Start the application by running `npm start` from the command line in the project root folder, this will launch a browser displaying the Angular example application and it should be hooked up with the ASP.NET Core JWT Auth API that you already have running.

Running a React client app with the ASP.NET Core JWT Auth API

For full details about the example React application see the post React + Redux - JWT Authentication Tutorial & Example (/post/2017/12/07/react-redux-jwt-authentication-tutorial-example). But to get up and running quickly just follow the below steps.

1. Download or clone the React tutorial code from https://github.com/cornflourblue/react-redux-jwt-authentication-example (https://github.com/cornflourblue

/react-redux-jwt-authentication-example)
2. Install all required npm packages by running `npm install` from the command line in the project root folder (where the package.json is located).
3. Remove or comment out the 2 lines below the comment `// setup fake backend` located in the `/src/index.jsx` file.
4. Start the application by running `npm start` from the command line in the project root folder, this will launch a browser displaying the React example application and it should be hooked up with the ASP.NET Core JWT Auth API that you already have running.

Running a VueJS client app with the ASP.NET Core JWT Auth API

For full details about the example VueJS JWT application see the post Vue.js + Vuex - JWT Authentication Tutorial & Example (/post/2018/07/06/vue-vuex-jwt-authentication-tutorial-example). But to get up and running quickly just follow the below steps.

1. Download or clone the VueJS tutorial code from https://github.com/cornflourblue/vue-vuex-jwt-authentication-example (https://github.com/cornflourblue/vue-vuex-jwt-authentication-example)
2. Install all required npm packages by running `npm install` from the command line in the project root folder (where the package.json is located).
3. Remove or comment out the 2 lines below the comment `// setup fake backend` located in the `/src/index.js` file.
4. Start the application by running `npm start` from the command line in the project root folder, this will launch a browser displaying the VueJS example application and it should be hooked up with the ASP.NET Core JWT Auth API that you already have running.

ASP.NET Core JWT Authentication Project Structure

The tutorial project is organised into the following folders:

Controllers - define the end points / routes for the web api, controllers are the entry point into the web api from client applications via http requests.

Services - contain business logic, validation and data access code.

Entities - represent the application data.

Helpers - anything that doesn't fit into the above folders.

Click any of the below links to jump down to a description of each file along with its code:

- Controllers
  - UsersController.cs
- Entities
  - User.cs
- Helpers
  - AppSettings.cs
- Services
  - UserService.cs
- appsettings.Development.json

- appsettings.json
- Program.cs
- Startup.cs
- WebApi.csproj

ASP.NET Core JWT Users Controller

Path: /Controllers/UsersController.cs

The ASP.NET Core users controller defines and handles all routes / endpoints for the api that relate to users, this includes authentication and standard CRUD operations. Within each route the controller calls the user service to perform the action required, this enables the controller to stay 'lean' and completely separated from the business logic and data access code.

The controller actions are secured with JWT using the [Authorize] attribute, with the exception of the Authenticate method which allows public access by overriding the [Authorize] attribute on the controller with [AllowAnonymous] attribute on the action method. I chose this approach so any new action methods added to the controller will be secure by default unless explicitly made public.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using WebApi.Services;
using WebApi.Entities;

namespace WebApi.Controllers
{
  [Authorize]
    [ApiController]
    [Route("[controller]")]
    public class UsersController : ControllerBase
    {
        private IUserService _userService;

        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        [AllowAnonymous]
        [HttpPost("authenticate")]
        public IActionResult Authenticate([FromBody]User userParam)
        {
            var user = _userService.Authenticate(userParam.Username, userParam.Password);

            if (user == null)
                return BadRequest(new { message = "Username or password is incorrect" });

            return Ok(user);
        }

        [HttpGet]
        public IActionResult GetAll()
        {
            var users =  _userService.GetAll();
            return Ok(users);
        }
    }
}
```

Back to top

ASP.NET Core JWT User Entity

Path: /Entities/User.cs

The user entity class represents the data for a user in the application. Entity classes are used to pass data between different parts of the application (e.g. between services and controllers) and can be used to return http response data from controller action methods.

```
namespace WebApi.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
        public string Token { get; set; }
    }
}
```

Back to top

ASP.NET Core JWT App Settings

Path: /Helpers/AppSettings.cs

The app settings class contains properties defined in the appsettings.json file and is used for accessing application settings via objects that injected into classes using the ASP.NET Core built in dependency injection. For example the User Service accesses app settings via an `IOptions<AppSettings> appSettings` object that is injected into the constructor.

Mapping of configuration sections to classes is done in the `ConfigureServices` method of the Startup.cs file.

```
namespace WebApi.Helpers
{
    public class AppSettings
    {
        public string Secret { get; set; }
    }
}
```

Back to top

ASP.NET Core JWT User Service

Path: /Services/UserService.cs

The user service contains a method for authenticating user credentials and returning a JWT token, and a method for getting all users in the application.

I hardcoded the array of users in the example to keep it focused on JWT authentication, in a production application it is recommended to store user records in a database with hashed passwords. For an extended example that includes support for user registration and stores data with Entity Framework Core check out ASP.NET Core 2.2 - Simple API for Authentication, Registration and User Management (/post/2018/06/26/aspnet-core-21-simple-api-for-authentication-registration-and-user-management).

The top of the file contains an interface that defines the user service, below that is the concrete user service class that implements the interface.

On successful authentication the Authenticate method generates a JWT (JSON Web Token) using the `JwtSecurityTokenHandler` class that generates a token that is digitally

signed using a secret key stored in appsettings.json. The JWT token is returned to the client application which then must include it in the HTTP Authorization header of subsequent web api requests for authentication.

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using WebApi.Entities;
using WebApi.Helpers;

namespace WebApi.Services
{
    public interface IUserService
    {
        User Authenticate(string username, string password);
        IEnumerable<User> GetAll();
    }

    public class UserService : IUserService
    {
        // users hardcoded for simplicity, store in a db with hashed passwords in productior
        private List<User> _users = new List<User>
        {
            new User { Id = 1, FirstName = "Test", LastName = "User", Username = "test", Pas
        };

        private readonly AppSettings _appSettings;

        public UserService(IOptions<AppSettings> appSettings)
        {
            _appSettings = appSettings.Value;
        }

        public User Authenticate(string username, string password)
        {
            var user = _users.SingleOrDefault(x => x.Username == username && x.Password == p

            // return null if user not found
            if (user == null)
                return null;

            // authentication successful so generate jwt token
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim(ClaimTypes.Name, user.Id.ToString())
                }),
                Expires = DateTime.UtcNow.AddDays(7),
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), S
            };
            var token = tokenHandler.CreateToken(tokenDescriptor);
            user.Token = tokenHandler.WriteToken(token);

            // remove password before returning
            user.Password = null;

            return user;
        }
```

```
        public IEnumerable<User> GetAll()
        {
            // return users without passwords
            return _users.Select(x => {
                x.Password = null;
                return x;
            });
        }
    }
}
```

Back to top

ASP.NET Core JWT App Settings (Development)

Path: /appsettings.Development.json

Configuration file with application settings that are specific to the development environment.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

Back to top

ASP.NET Core JWT App Settings

Path: /appsettings.json

Root configuration file containing application settings for all environments.

IMPORTANT: The `"Secret"` property is used by the api to sign and verify JWT tokens for authentication, update it with your own random string to ensure nobody else can generate a JWT to gain unauthorised access to your application.

```
{
  "AppSettings": {
    "Secret": "THIS IS USED TO SIGN AND VERIFY JWT TOKENS, REPLACE IT WITH YOUR OWN SECRET,
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

Back to top

ASP.NET Core JWT Program

Path: /Program.cs

The program class is a console app that is the main entry point to start the application, it configures and launches the web api host and web server using an instance of `WebHostBuilder`. ASP.NET Core applications require a host in which to execute.

Kestrel is the web server used in the example, it's a new cross-platform web server for ASP.NET Core that's included in new project templates by default. Kestrel is fine to use on it's own for internal applications and development, but for public facing websites and applications it should sit behind a more mature reverse proxy server (IIS, Apache, Nginx etc) that will receive HTTP requests from the internet and forward them to Kestrel after initial handling and security checks.

```
using System.IO;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace WebApi
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .UseUrls("http://localhost:4000")
                .Build();
    }
}
```

Back to top

ASP.NET Core JWT Startup

Path: /Startup.cs

The startup class configures the request pipeline of the application and how all requests are handled.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using WebApi.Helpers;
using WebApi.Services;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Mvc;

namespace WebApi
{
    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 co
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCors();
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

            // configure strongly typed settings objects
            var appSettingsSection = Configuration.GetSection("AppSettings");
            services.Configure<AppSettings>(appSettingsSection);

            // configure jwt authentication
            var appSettings = appSettingsSection.Get<AppSettings>();
            var key = Encoding.ASCII.GetBytes(appSettings.Secret);
            services.AddAuthentication(x =>
            {
                x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            })
            .AddJwtBearer(x =>
            {
                x.RequireHttpsMetadata = false;
                x.SaveToken = true;
                x.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                    ValidateIssuer = false,
                    ValidateAudience = false
                };
            });

            // configure DI for application services
            services.AddScoped<IUserService, UserService>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP req
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            // global cors policy
            app.UseCors(x => x
```

```
                .AllowAnyMethod()
                .AllowAnyHeader());

            app.UseAuthentication();

            app.UseMvc();
        }
    }
}
```

Back to top

ASP.NET Core JWT Web Api csproj

Path: /WebApi.csproj

The csproj (C# project) is an MSBuild based file that contains target framework and NuGet package dependency information for the application.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.2</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
  </ItemGroup>
</Project>
```

Back to top

Web Development Sydney

Feel free to contact me (/contact) if you're looking for a web developer in Sydney, I also provide remote contracting services for clients outside Sydney.

Tags: ASP.NET Core (/posts/tag/aspnet-core), C# (/posts/tag/c), Authentication and Authorization (/posts/tag/authentication-and-authorization), Security (/posts/tag/security), JWT (/posts/tag/jwt)

Share:

More ASP.NET Core Posts

- ASP.NET Core 2.2 - Role Based Authorization Tutorial with Example API (/post

/2019/01/08/aspnet-core-22-role-based-authorization-tutorial-with-example-api)

- C# - Pure Pagination Logic in C# / ASP.NET (/post/2018/10/17/c-pure-pagination-logic-in-c-aspnet)
- ASP.NET Core Razor Pages - Pagination Example (/post/2018/10/15/aspnet-core-razor-pages-pagination-example)
- ASP.NET Core 2.2 - Basic Authentication Tutorial with Example API (/post/2018/09/08/aspnet-core-21-basic-authentication-tutorial-with-example-api)
- ASP.NET Core 2.2 - Simple API for Authentication, Registration and User Management (/post/2018/06/26/aspnet-core-21-simple-api-for-authentication-registration-and-user-management)

---

73 Comments Jason Watmore's Blog Login

Recommend 16 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

Narendar kumar • 7 months ago

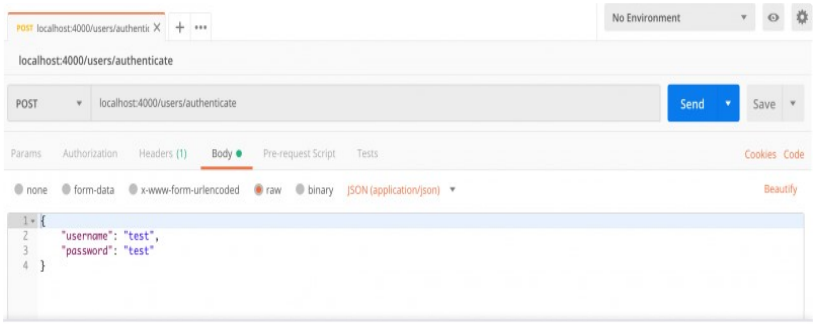
info:Microsoft.AspNetCore.Authorization.DefaultAuthorizationService[2]
Authorization failed.
info: Microsoft.AspNetCore.Authorization.DefaultAuthorizationService[2]
Authorization failed.
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[3]
Authorization failed for the request at filter
'Microsoft.AspNetCore.Mvc.Authorization.AuthorizeFilter'.
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[3]
Authorization failed for the request at filter
'Microsoft.AspNetCore.Mvc.Authorization.AuthorizeFilter'.
when i call GetAll() function it's showing these error what should i do to solved it? I have changed hosting environment to development

19 • Reply • Share ›

Jason Watmore Mod Narendar kumar • 5 months ago

Hi Narenda, it looks like you might be using Postman to hit the get all users route (`/users`) without a JWT token in the authorization header.

To get a JWT token first make a POST request to the authenticate route (`/users/authenticate`) with the username and password in the body (with "raw" and "JSON (application/json)" selected). Here's a screenshot of how it should look in Postman:



see more

• Reply • Share ›

maddy Jason Watmore • 3 months ago

Hi Jason,
I get a http 404 error on running the api. Can you help? I followed all the steps but it shows localhost page cant be found.

1 • Reply • Share ›

David Bridge • 3 months ago

I have some very similar code to this, though not derived from your tutorial I found this page as I was having some problem.
In my case I wasn't getting an error and everything appeared to work but my API returned 401 every time.
Having banged my head a lot on this I eventually got my answer from the wonder Shaun Wildermuth.
I had ...
[Authorize]

ABOUT

I'm a web developer in Sydney Australia and the technical lead at Point Blank Development (https://www.pointblankdevelopment.com.au), I've been building websites and web applications in Sydney since 1998.

Find me on:

(https://twitter.com/jason_watmore)

(https://github.com/cornflourblue)

(https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg/)

Support me on Patreon (https://www.patreon.com/jasonwatmore)

MONTHS

TAGS

Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular Directive, Angular UI Router, AngularJS, Animation, ASP.NET, ASP.NET Core, ASP.NET Web API, Authentication and Authorization, AWS, Basic Authentication, Bootstrap, C#, Chai, CKEditor, CSS3, DDD, Design Patterns, Dynamic LINQ, ELMAH, ES6, Exchange, Facebook, Fluent NHibernate, Google Analytics, Google API, Google Maps API, Google Plus, Heroku, HTML5, HTTP, IIS, Insecure Content, Instagram API, Ionic Framework, iOS, iPhone, JavaScript, jQuery, JWT, LinkedIn, LINQ, Login, MEAN Stack, Mocha, Modal, MongoDB, Moq, MVC, MVC5, NGINX, ngMock, NHibernate, Ninject, NodeJS, npm, Pagination, Pinterest, Razor Pages, React, Redmine, Redux, Registration, Repository, RxJS, Security, Shell Scripting, Sinon, SinonJS, TDD, Terraform, Twitter, TypeScript, Ubuntu, Umbraco, Unit of Work, Unit Testing, URL Rewrite, Validation, Vue, Vuex, Webpack, Windows Server 2008,

Powered by MEANie (https://jasonwatmore.com/meanie)