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 production applications
        private List<User> _users = new List<User>
        {
            new User { Id = 1, FirstName = "Test", LastName = "User", Username = "test", Password = "test" }
        };

        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 == password);

            // 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), SecurityAlgorithms.HmacSha256Signatu
            };
            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, IT CAN BE ANY STRING"
  },
  "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 container.
        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);
```

```
        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 request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        // global cors policy
        app.UseCors(x => x
```

```
            app.UseCors(x => x
                .AllowAnyOrigin()
                .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

Subscribe or Follow Me For Updates

Subscribe to my YouTube channel or follow me on Twitter or GitHub to be notified when I post new content.

- Subscribe on YouTube at https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg/ (https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg?sub_confirmation=1)
- Follow me on Twitter at https://twitter.com/jason_watmore (https://twitter.com/jason_watmore)
- Follow me on GitHub at https://github.com/cornflourblue (https://github.com/cornflourblue)

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)

Narendar kumar • 9 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

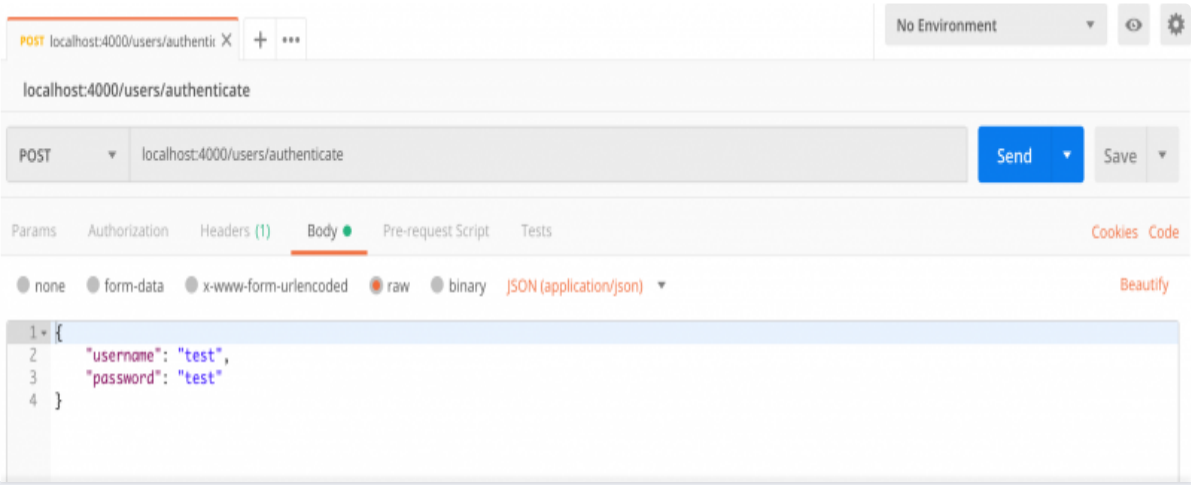
changed hosting environment to development

19 • Reply • Share ›

Jason Watmore Mod ➦ Narendar kumar • 7 months ago • edited

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 • 5 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 • 5 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]

on my Controller and found that the site was trying to use cookie authentication so although my JWT worked fine, the lack of a cookie auth made it fail.
I changed the attribute to ...
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

and this fixed the issue as now the controller ignores cookie auth and concentrates only on jwt.

here's a link...

https://wildermuth.com/2017...

Shaun refers to the issue about one third down the page as

"When we use the Authorize attribute, it actually binds to the first authentication system by default. The trick is to change the attribute to specify which auth to use:
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]"

In his case he actually wants to use both types but I found that although I am not using cookies at all, the application still wants the declaration to use JWT and without this it just fails to authorize.

Hope this helps someone

Hope this helps someone

David

9 • Reply • Share ›

Siddharth ➦ David Bridge • 11 days ago

Thankssss... This actually helps !!

• Reply • Share ›

Charlie Brown Jr. ➦ David Bridge • 21 days ago

David, you're the Dude !! Thanks a lot, it avoided me many headaches !!

• Reply • Share ›

Show more replies

Manuel Mejia Jr. • 7 months ago

Thank you, amazing detailed tutorial.

6 • Reply • Share ›

ikenna emman • 7 months ago

Hi, I noticed that the tutorial does not use UserManager and IdentityUser. Any idea on how to implement PasswordReset and EmailConfirmation Token using the JWT approach.

Thanks

3 • Reply • Share ›

SensuaCL • 6 months ago

Thanks, very usefull implementation, i realy apreciate your work here.:D

2 • Reply • Share ›

Anon • 9 months ago

Hi, thanks for the article. It's very useful. How would one determine which user is logged in, in a

controller action? Is there a way to get the current user?

4 • Reply • Share ›

Unkown ➦ Anon • 5 months ago

The controller has an User property.

2 • Reply • Share ›

Reza Septiandra • 2 months ago

its work fine. Thanks!

1 • Reply • Share ›

Vctor Usoro • 2 months ago

Thank you for the write up. Your a life saver.

1 • Reply • Share ›

Mansur Haider • 6 months ago

Very clear explanation. Thanks for sharing this kind of valuable article.

1 • Reply • Share ›

Laurent Knafo • 7 months ago

Thanx Jason, great post! helped me a lot!

1 • Reply • Share ›

أسامة الشمري • 7 months ago

So grateful for you... Thank you.

1 • Reply • Share ›

Liesbert García Moreno • 8 months ago

Thank you so much, just what I needed.

1 • Reply • Share ›

Enterprise Lab • 9 months ago

Thank you so much! It's helpful.

1 • Reply • Share ›

Mohammed Kamran Azam • 10 months ago

How can we invalidate the token if the user wants to logout from the app?

1 • Reply • Share ›

Jason Watmore Mod → Mohammed Kamran Azam • 9 months ago

To invalidate tokens you can save a list of invalidated tokens in a db or somewhere else on the server-side, then check tokens against this list during validation.

Cheers,
Jason

• Reply • Share ›

PRASANNA HIREMATH → Jason Watmore • a month ago

Hello Json,

Can we get the token sent from client (Angular app) into my Web Api ? If yes how can I get?

• Reply • Share ›

Show more replies

Ali Javani • 7 months ago

Einstein said: if you can't explain it simply, you don't understand it well enough, and i can promise "Json Watmore" understand authentication and authorization better than every one i

thanks Jason.

1 • Reply • Share ›

JR • 17 days ago

Is it safe to store the token in Web Storage (local/session)?

• Reply • Share ›

Jason Watmore Mod → JR • 16 days ago

Hi JR, yes I think it's fine to store tokens in local / session storage if you want to stay logged in between browser refreshes. Some people argue that it's not secure if your web app is vulnerable to XSS (Cross-Site Scripting) attacks, but if you're web app is vulnerable to XSS then you have a whole lot of problems.

So in short, make sure that your web app is not vulnerable to XSS (which you should do in any case) and your local storage will be safe.

For more info on XSS see https://en.wikipedia.org/wiki/Cross-site_scripting.

Cheers,
Jason

• Reply • Share ›

ajiehatajie • 2 months ago

If token expired ? What can doing

• Reply • Share ›

Jason Watmore Mod → ajiehatajie • 2 months ago

Hi **@ajiehatajie**,

If the token is expired a 401 response will be returned by the api, then you need to get a new token by authenticating again.

Cheers,

Jason

• Reply • Share ›

ajiehatajie → Jason Watmore • 2 months ago

What about refresh token method ?

• Reply • Share ›

Show more replies

Gowrisankar • 2 months ago

I've spent a week now securing my Web API, creating custom filters and uses of authentication tokens. My problem now was when I'm requesting in my Web API using POSTMAN and the user was already sign out I can still get values from my API.

How can i manage to force expire my access token? Or is there other way to manage this kind of situation?

• Reply • Share ›

Claude Glauser • 2 months ago

Tnx for the tutorial. CORS is not mentioned but activated in the sample. Is this meant for generating the auth token for other sites? How is this done in "production" in a large corporation? Are there only a few authentication services and many sites?

• Reply • Share ›

Jason Watmore Mod → Claude Glauser • 2 months ago

Hi Claude, I've enabled CORS in the example so the API will work with web clients running on a different url, the front end examples I built to run with the API all run on a different url than the API.

Cheers
Jason

• Reply • Share ›

Vasili Ermuratski • 3 months ago

Hello Jason,

I am amazed how you have not resolved the UserService instance in config and it still worked ??

I added services.AddTransient<iuserservice, userservice="">();
to public void ConfigureServices(IServiceCollection services)
it works....

Can you please explain

• Reply • Share ›

Jason Watmore Mod → Vasili Ermuratski • 3 months ago

Hi Vasili,

It's configured in Startup.cs - https://github.com/cornflou...

Cheers,
Jason

1 • Reply • Share ›

David Pantea • 3 months ago

Hello,
I also receive 401 Unauthorized when i try to access Get() method from UsersController.
I setted the header: Authorization = 'Bearer ' + token
Please advice!

• Reply • Share ›

Keyvan Sadralodabai ➦ David Pantea • 3 months ago

This has to do with an invalid bearer token more than likely.

• Reply • Share ›

Hardik Patel • 3 months ago

I am able to get the token in postman.
but when i insert the token for GetAll() method, it gives 404.
Am i missing anything?
I have tried many ways, like keeping token in quotes, used content-type and all.

• Reply • Share ›

Jason Watmore Mod → Hardik Patel • 3 months ago

Hi Hardik, the route to get all users is `http://localhost:4000/users`.

The route for an action method is set in the http attribute e.g. `[HttpPost("authenticate")]` sets the route to `/users/authenticate`. If the http attribute doesn't set a route e.g. `[HttpGet]` then it sets it to the default route for the controller.

Cheers,
Jason

• Reply • Share ›

Supporter **Hardik Patel** → Jason Watmore • 3 months ago

Thanks a lot Jason!
Sorry for such a silly question...

• Reply • Share ›

Judicael Abi • 4 months ago

Could not get any response
There was an error connecting to http://localhost:63175/api/UtilisateurOni/listeUtilisateurs.
Why this might have happened:
The server couldn't send a response:
Ensure that the backend is working properly
Self-signed SSL certificates are being blocked:
Fix this by turning off 'SSL certificate verification' in Settings > General
Proxy configured incorrectly
Ensure that proxy is configured correctly in Settings > Proxy
Request timeout:
Change request timeout in Settings > General

• Reply • Share ›

AndrewDay • 4 months ago

How can I create an ASP.NET Core project targeting .NET Framework and continure the tutorial as I need to reference an existing layer.

• Reply • Share ›

David Ćubela • 5 months ago • edited

I have a slight problem:
I am forced to use [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)] on my controllers, otherwise it won't work.
How can I make it so that I don't have to write it every time?
Ah, reading further down, on David Bridge's post, I noticed that this happens if we have both cookie and JWT tokens included. Now, I never explicitly included Cookies anywhere in my app, however, if I'm not mistaken, using Identity automatically uses cookies, doesn't it?
So, by adding UserManager and stuff, I automatically used Cookies?
services.AddIdentity<user, role="">()

.AddEntityFrameworkStores<ngschoolscontext>()
.AddRoles<identityrole<guid>>()
.AddDefaultTokenProviders();
And that's why I need to explicitly tell my controllers to use JWT, and NOT Cookies?
• Reply • Share ›

Maro • 5 months ago
Such a great tutorial. However, since this is not using the ASP.NET Core Identity, it has no roles, no bruteforce protection, etc. I was unable to find a solution to this on the internet.
• Reply • Share ›

Jason Watmore Mod → Maro • 5 months ago
Hi Maro, you can add roles and bruteforce protection without identity, I've posted details in the below tutorials:

- ASP.NET Core 2.2 - Role Based Authorization Tutorial with Example API
- C# - Incremental Delay to Prevent Brute Force or Dictionary Attack - this post is in MVC 5 but the code could easily be converted to .net core.

Cheers,
Jason
1 • Reply • Share ›

Unkown • 5 months ago • edited
Jason, thanks for sharing.

How can we ensure anyone won't get the password when the request is sent to /authenticate? I see lots of posts using token authentication using http only.

In all my projects I use https from the beginning. I don't get how we are securing an application by using a token over http. Are authentication headers special so they are always encrypted even over http?

I got this from Wikipedia:

"The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with Base64 in transit, but not encrypted or hashed in any way. Therefore, Basic Authentication is typically used in conjunction with HTTPS to provide confidentiality. "

https://en.wikipedia.org/wi...

Thanks again.

• Reply • Share ›

Jason Watmore Mod → Unkown • 5 months ago

Hi Domício, as far as I know https is the way to secure any information sent over the wire, so in production applications you should always use https if you're handling any sensitive data.

In your local environment you can run either http or https, running https locally requires a bit more setup which is outside the scope of this tutorial which is why I didn't include it, I wanted to keep it focused on JWT authentication. If you're interested in running https in your local environment with ASP.NET Core you can check out this post by Scott Hanselman.

Cheers,
Jason

1 • Reply • Share ›

Unkown → Jason Watmore • 5 months ago

Thanks, Jason.

1 • Reply • Share ›

Onedevteam.com • 6 months ago • edited

HI Jason

thanks for great tutorials...

I have one question, how to access token in another controller?

For example, i have "ProductsController", and i would like to read user name from token, so i could attach specific product to currently logged user.

How can i achieve this??

edit: i do know how to write user name to token, and how to decode it, i just don't know how to access token in other controller...

• Reply • Share ›

Nazmul Hossain • 6 months ago

Thanks for your excellent blog post.

I download your code and test everything OK .
But i create a new asp.net core 2.2 web api project , write code as yours and get this error :(

see more

• Reply • Share ›

Jason Watmore Mod ➦ Nazmul Hossain • 6 months ago

Hi Nazmul, it could be that your secret isn't long enough, another person ran into a similar error and it was caused by the secret length, try a longer string to see if it fixes the issue.

Cheers,
Jason

2 • Reply • Share ›

Nazmul Hossain ➦ Jason Watmore • 6 months ago

Thanks. now OK .

• Reply • Share ›

Jitendra Jadav • 6 months ago

Hi Jason,
I have gone through your blog and tried your solution it is working fine but I have implemented same and it is not working authorization it is giving me error 401 Unauthorized
while "OnTokenValidated" event fired every times and give me error 401 token is also valid
please find postman

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

Alerts, Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular 8, 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 (/post/2016/10/29/meanie-mean-stack-blogging-platform)