**Implement JWT In ASP.NET Core 3.1**

**Reference URLs:**

<https://www.codemag.com/Article/2105051/Implementing-JWT-Authentication-in-ASP.NET-Core-5#:~:text=JSON%20Web%20Tokens%20(commonly%20known,consumers%20in%20a%20secure%20manner>.

<https://www.c-sharpcorner.com/article/implement-jwt-in-asp-net-core-3-1/>

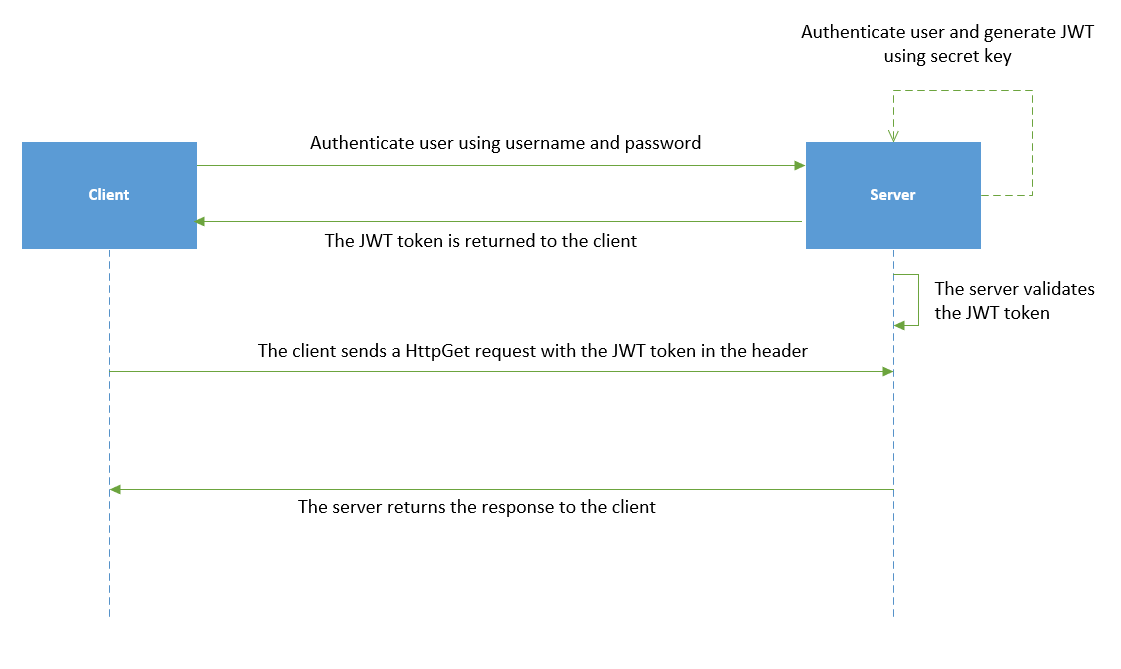
With the surge in APIs and their consumption globally, API security is extremely important these days. JWT authentication is a standard way for protecting APIs.

JSON Web Tokens (commonly known as JWT) is an open standard to pass data between client and server, and enables you to transmit data back and forth between the server and the consumers in a secure manner.

## What Are JSON Web Tokens (JWT)?

JSON Web Token is an open standard (RFC 7519) that defines a safe, compact, and self-contained, secured way for transmission of information between a sender and a receiver through a URL, a POST parameter, or inside the HTTP Header. It should be noted that the information to be transmitted securely between two parties is represented in JSON format and it is cryptographically signed to verify its authenticity. JWT is typically used for implementing authentication and authorization in Web applications. Because JWT is a standard, all JWTs are tokens but the reverse is not true. You can work with JSON Web Tokens in .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript, etc.

Following figure illustrates how a typical JWT authentication works.



In this article we will cover the following,

* Creating the method to generate the JWT token
* Creating the middleware needed to validate the token
* Decorating the API controller
* Testing our API with Fiddler

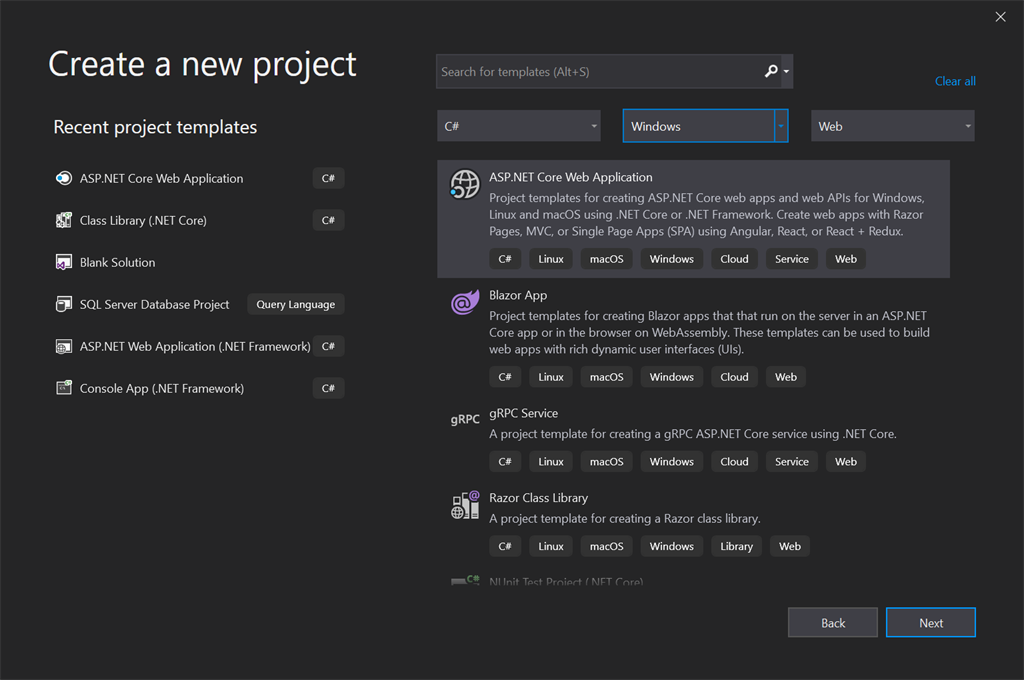
First things first, let's start with a brand new project. I am using VS 2019 Community Edition.

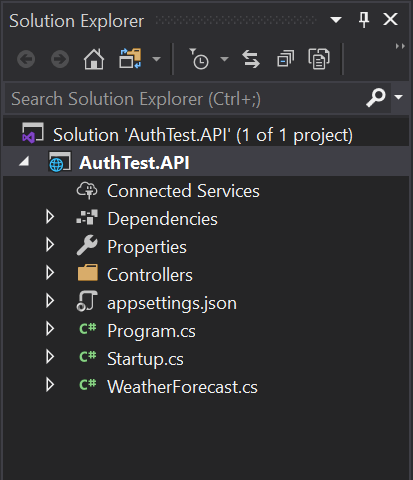
Create a new ASP.NET Core Web Application.  Choose the API with no authentication template.

I am calling my project AuthTest.API

IMPORTANT. I highly suggest you name your project, folders and classes the same as the article below, or you will find yourself having to track down and clean up the namespaces.

If you want the source code you can get it from [github](https://github.com/fscopel/token-based-authentication.git" \t "_blank).





Good! Now let's do some coding. I am adding two folders to the project: Services and Middleware just for organization purposes.

In the Services folder add a class called JwtServices.cs

We also need some Nuget packages. Right-click Dependencies -> Manage Nuget Packages...  on the Browse tab search and install both of these packages:

* System.IdentityModel.Tokens.Jwt
* Microsoft.IdentityModel.Tokens

Let's fill in the JwtService class.

1. **using** System;
2. **using** System.Text;
3. **using** System.Security.Claims;
4. **using** Microsoft.IdentityModel.Tokens;
5. **using** System.IdentityModel.Tokens.Jwt;
6. **using** Microsoft.Extensions.Configuration;
8. **namespace** AuthTest.API.Services
9. {
10. **public** **class** JwtService
11. {
12. **private** **readonly** **string** \_secret;
13. **private** **readonly** **string** \_expDate;
15. **public** JwtService(IConfiguration config)
16. {
17. \_secret = config.GetSection("JwtConfig").GetSection("secret").Value;
18. \_expDate = config.GetSection("JwtConfig").GetSection("expirationInMinutes").Value;
19. }
21. **public** **string** GenerateSecurityToken(**string** email)
22. {
23. var tokenHandler = **new** JwtSecurityTokenHandler();
24. var key = Encoding.ASCII.GetBytes(\_secret);
25. var tokenDescriptor = **new** SecurityTokenDescriptor
26. {
27. Subject = **new** ClaimsIdentity(**new**[]
28. {
29. **new** Claim(ClaimTypes.Email, email)
30. }),
31. Expires = DateTime.UtcNow.AddMinutes(**double**.Parse(\_expDate)),
32. SigningCredentials = **new** SigningCredentials(**new** SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
33. };
35. var token = tokenHandler.CreateToken(tokenDescriptor);
37. **return** tokenHandler.WriteToken(token);
39. }
40. }
41. }

In the construtor I am pulling some data out of the appsettings.json, but we haven't added those settings yet. Don't worry - we will right after this.

The reason for it is that the JWT generator needs some kind of secret string, some kind of password if you will, and an expiration date to generate the token.

The secret can be anything you want, just like a random password. I just typed in some random letters and numbers, and I decided the expiration is 1440 minutes (24hrs).

That means, the users for my API will have to get a new token every 24 hrs. It can be anything you want. You could choose to only expire the token if the user logs out (not recommended) or you could renew the token every so often. I am not covering that here.

Pay special attention to the Subject property line 27 through 30.

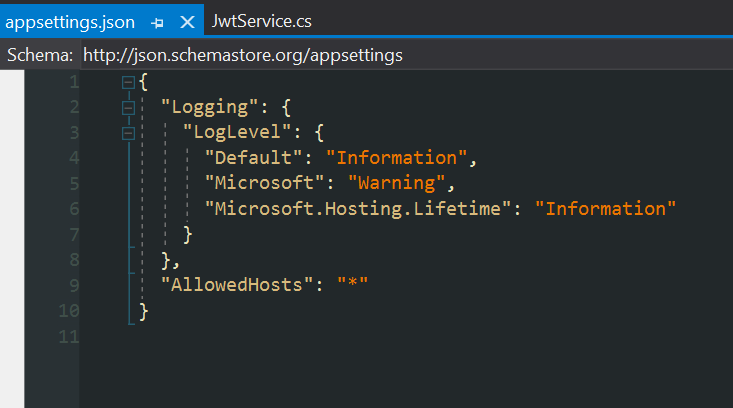
I am passing in an email to the function, GenerateSecurityToken(string email) and storing that email in the token. You could pass in some a user object GenerateSecurityToken(User user) for example and store a lot more information by adding new claims. This way you don't need to take trips to the DB to get that data, when the user makes a call into the system.

**Example**

Something like this...

1. **public** **string** GenerateSecurityToken(User user){
2. ...
3. Subject = **new** ClaimsIdentity(**new**[]
4. {
5. **new** Claim(ClaimTypes.Email, user.Email),
6. **new** Claim(ClaimTypes.Name, user.Name),
7. **new** Claim(ClaimTypes.Role, user.Role),
8. **new** Claim(ClaimTypes.DateOfBirth, user.DOB),
9. })
10. ...
11. }

We went on a bit of a tangent, let's get back to the appsettings.config. In the appsettings.json we need to add a new configuration.  This is what my appsettings.config looks like right now.



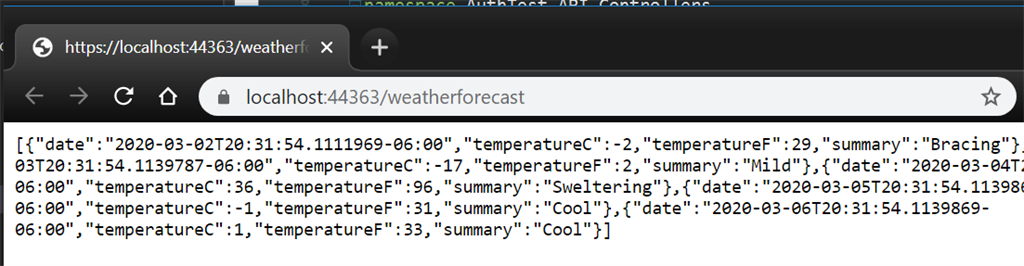
This is what it looks like after adding the JwtConfig section.

The JWT generation,

1. {
2. "Logging": {
3. "LogLevel": {
4. "Default": "Information",
5. "Microsoft": "Warning",
6. "Microsoft.Hosting.Lifetime": "Information"
7. }
8. },
9. "AllowedHosts": "\*",
10. "JwtConfig": {
11. "secret": "PDv7DrqznYL6nv7DrqzjnQYO9JxIsWdcjnQYL6nu0f",
12. "expirationInMinutes": 1440
13. }
14. }

We now have the secret and the expiration data points needed by the JwtService class.

Go ahead and run your app right now.  If Microsoft hasn't changed the template by the time you are following this article, you should probably get some fake weather json data on your browser.

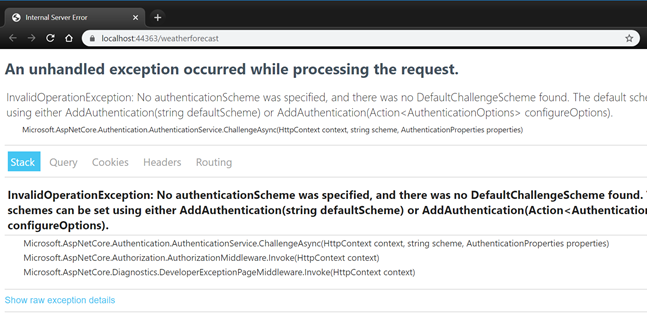


This means the app is working and currenlty not requiring any kind of authentication to serve up data. Let's go ahead and mess that up! :)

Go head add the [Authorize] attribute, you will need to bring in the Microsoft.AspNetCore.Authorization and try running the project again.

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** Microsoft.AspNetCore.Authorization;
5. **using** Microsoft.AspNetCore.Mvc;
6. **using** Microsoft.Extensions.Logging;
8. **namespace** AuthTest.API.Controllers
9. {
10. [ApiController]
11. [Authorize]
12. [Route("[controller]")]
13. **public** **class** WeatherForecastController : ControllerBase
14. {
15. **private** **static** **readonly** **string**[] Summaries = **new**[]
16. {
17. "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
18. };
20. **private** **readonly** ILogger<WeatherForecastController> \_logger;
22. **public** WeatherForecastController(ILogger<WeatherForecastController> logger)
23. {
24. \_logger = logger;
25. }
27. [HttpGet]
28. **public** IEnumerable<WeatherForecast> Get()
29. {
30. var rng = **new** Random();
31. **return** Enumerable.Range(1, 5).Select(index => **new** WeatherForecast
32. {
33. Date = DateTime.Now.AddDays(index),
34. TemperatureC = rng.Next(-20, 55),
35. Summary = Summaries[rng.Next(Summaries.Length)]
36. })
37. .ToArray();
38. }
39. }
40. }

When I ran the project I got the error below. The error means that ASP.NET Core sees the [Authorize] attribute but doesnt know how to handle that, because we haven't configured a middleware to do so.  Let's go ahead a take care of that.



Back to the project go ahead and create a new class inside the Middleware folder, let's call this one AuthenticationMiddleware

Before we make any changes to this new class we need to bring one more Nuget package:

* Microsoft.AspNetCore.Authentication.JwtBearer

Browse and install the above package, and update the AuthenticationMiddleware class with the code below

1. **using** System.Text;
2. **using** Microsoft.IdentityModel.Tokens;
3. **using** Microsoft.Extensions.Configuration;
4. **using** Microsoft.Extensions.DependencyInjection;
5. **using** Microsoft.AspNetCore.Authentication.JwtBearer;
7. **namespace** AuthTest.API.Middleware
8. {
9. **public** **static** **class** AuthenticationExtension
10. {
11. **public** **static** IServiceCollection AddTokenAuthentication(**this** IServiceCollection services, IConfiguration config)
12. {
13. var secret = config.GetSection("JwtConfig").GetSection("secret").Value;
15. var key = Encoding.ASCII.GetBytes(secret);
16. services.AddAuthentication(x =>
17. {
18. x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
19. x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
20. })
21. .AddJwtBearer(x =>
22. {
23. x.TokenValidationParameters = **new** TokenValidationParameters
24. {
25. IssuerSigningKey = **new** SymmetricSecurityKey(key),
26. ValidateIssuer = **true**,
27. ValidateAudience = **true**,
28. ValidIssuer = "localhost",
29. ValidAudience = "localhost"
30. };
31. });
33. **return** services;
34. }
35. }
36. }

Now that we have the middleware built we need to hook it up to our services.

Open the Startup.cs class, find the ConfigurationServices, and the Configure functions and update with the code below.

You will have to import the reference to the namespace where the AuthenticationExtension is located

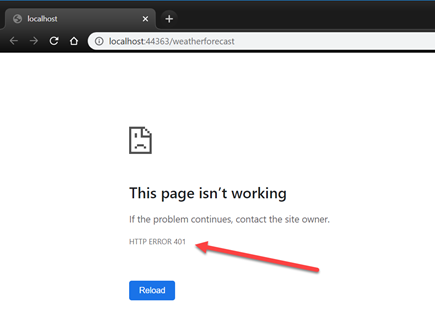
1. **using** AuthTest.API.Middleware;
2. **using** Microsoft.AspNetCore.Builder;
3. **using** Microsoft.AspNetCore.Hosting;
4. **using** Microsoft.Extensions.Configuration;
5. **using** Microsoft.Extensions.DependencyInjection;
6. **using** Microsoft.Extensions.Hosting;
8. **namespace** AuthTest.API
9. {
10. **public** **class** Startup
11. {
12. **public** Startup(IConfiguration configuration)
13. {
14. Configuration = configuration;
15. }
17. **public** IConfiguration Configuration { **get**; }
19. // This method gets called by the runtime. Use this method to add services to the container.
20. **public** **void** ConfigureServices(IServiceCollection services)
21. {
22. services.AddControllers();
23. services.AddTokenAuthentication(Configuration);
24. }
26. // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
27. **public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)
28. {
29. **if** (env.IsDevelopment())
30. {
31. app.UseDeveloperExceptionPage();
32. }
34. app.UseHttpsRedirection();
36. app.UseRouting();
37. app.UseAuthentication();
38. app.UseAuthorization();
40. app.UseEndpoints(endpoints =>
41. {
42. endpoints.MapControllers();
43. });
44. }
45. }
46. }

Cool! Let's go ahead a run a quick test! Go ahead and just start your app again.

Did you get a "This page isn't working" - If you did, good job!  That's exactlyy what we are lookin for. No more exceptions!

If you look closely you will see the server is displaying a 401 error - which means... drum roll... Unauthorized: https://httpstatuses.com/401

Our app now understands what we are looking for and since it didnt see a token in the request it returned an 401 - Unathorized error.



Stop the app and let's go ahead and create a new controller. Name this one TokenController

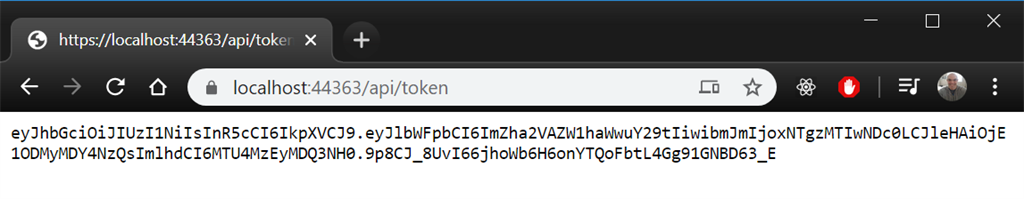
Right-click the Controllers folder and choose Add -> Controller...

Pick the API Controller - Empty template and click Add

1. **using** AuthTest.API.Services;
2. **using** Microsoft.AspNetCore.Mvc;
3. **using** Microsoft.Extensions.Configuration;
5. **namespace** AuthTest.API.Controllers
6. {
8. [Route("api/[controller]")]
9. [ApiController]
10. **public** **class** TokenController : ControllerBase
11. {
12. **private** IConfiguration \_config;
14. **public** TokenController(IConfiguration config)
15. {
16. \_config = config;
17. }
19. [HttpGet]
20. **public** **string** GetRandomToken()
21. {
22. var jwt = **new** JwtService(\_config);
23. var token = jwt.GenerateSecurityToken("fake@email.com");
24. **return** token;
25. }
26. }
27. }

Restart the app, and navigate to https://localhost:44363/api/token, your port number may vary.

Hopefully you got a token like me,



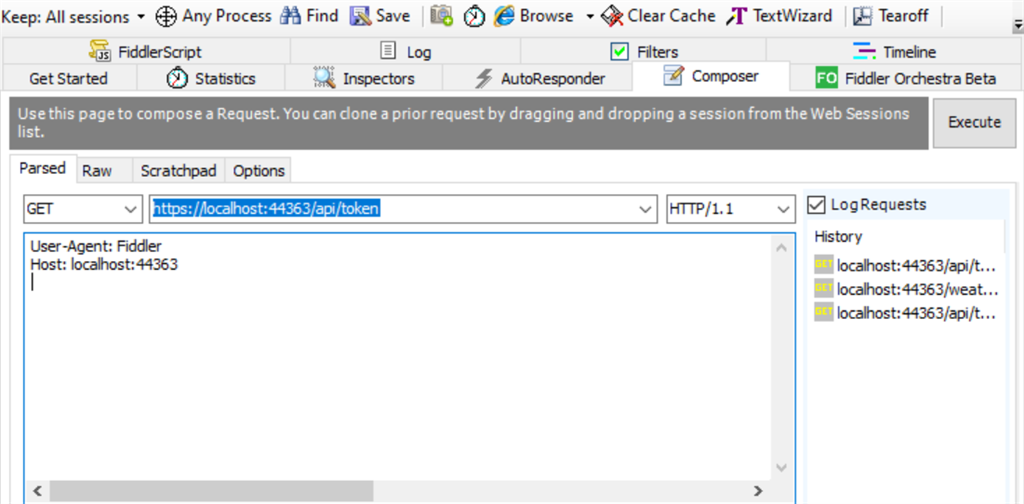
Now with Postman or Fiddler whichever tool you prefer, let's try to call into the WeatherForecastController and see if we can get through.

With the app running let's go ahead and make a call into the token endpoint to get a fresh token and then let's use that token to call into the weather forecast service.

Make sure your app is running end do a GET on .../api/token

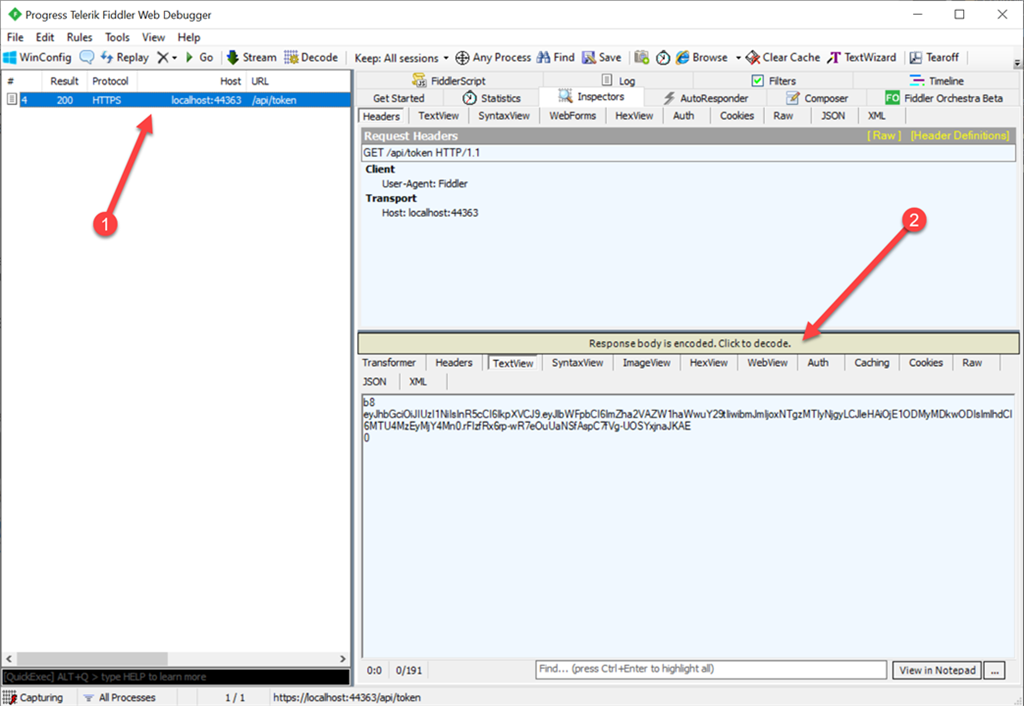
This is what fiddler looks like.

On the Composer tab choose GET from the dropdown and type in your URL, then click the Execute button.

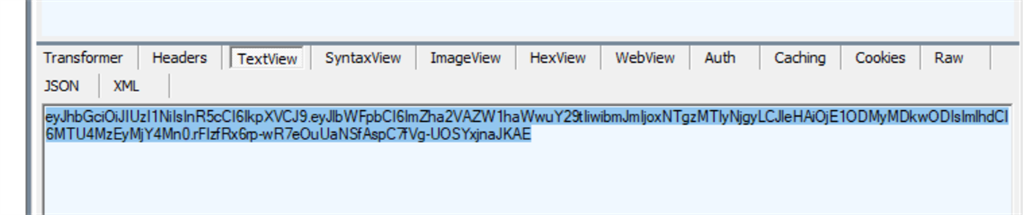


Double click the result on the left and then click on decode, to see your actual token.

Important: This is only happening because I am running my app in HTTPS. If I was running in HTTP, I would not need to decode the result.



After decoding the yellow text goes away and you can copy the token:



Let's compose another call into the WeatherForecast endpoint.

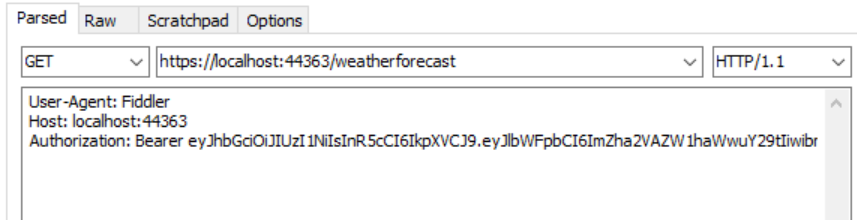
GET

header area:

User-Agent: Fiddler

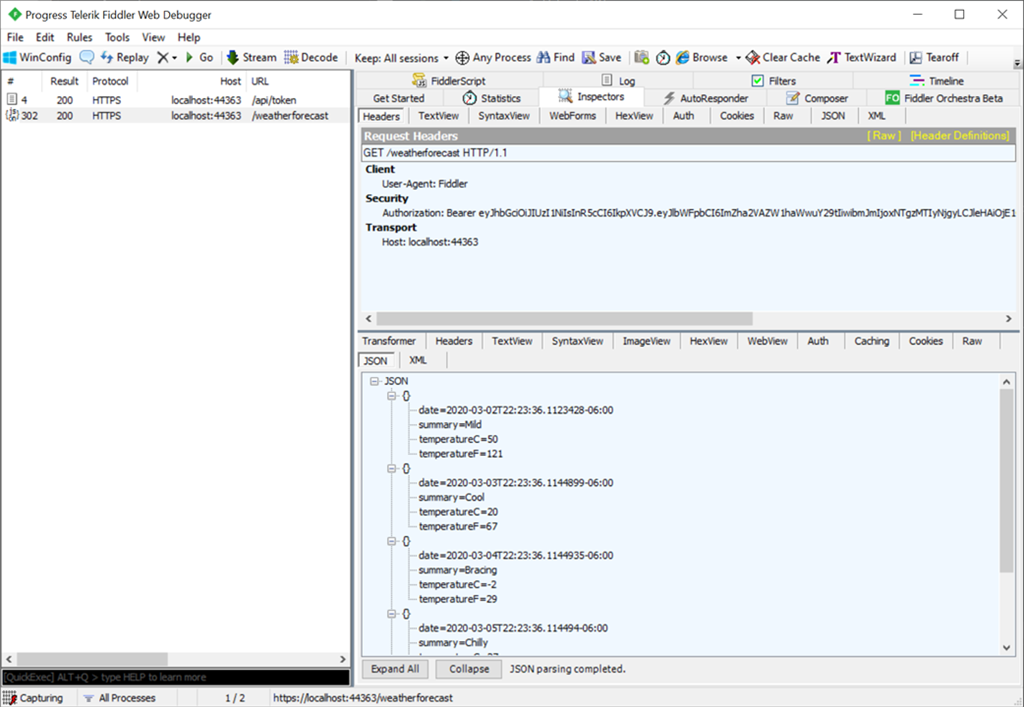
Host: localhost:44363

Authorization: Bearer YOUR TOKEN GOES HERE



Click the Execute button:

There it is - data!



To pass the token in Postman:

