Implementing JWT and Refresh Token in .Net Core 2.2 Web API

Mr. Simi Follow
Feb 11 · 6 min read

So you've built a web API and everything seems to be working fine.

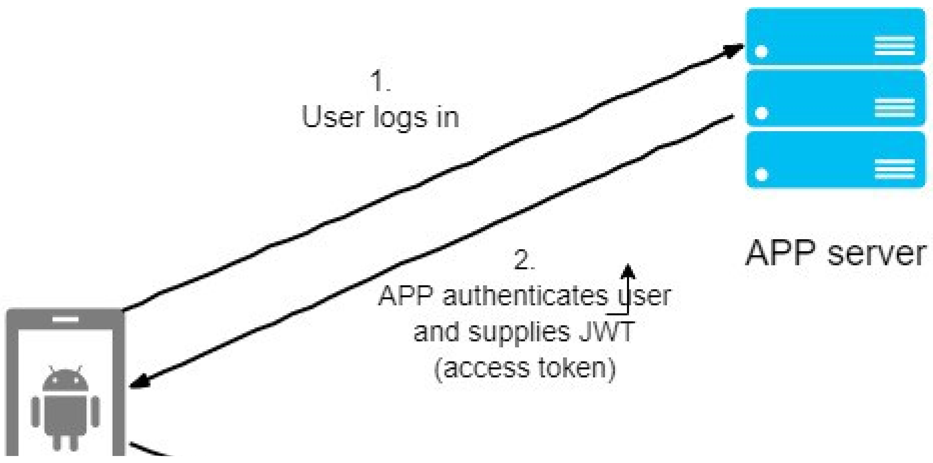
But Wait! Anyone can access your API, hit the correct url and boom your API is connected to. Though that might be heroic of you, but it shouldn't be… I mean after all the bugs and stackoverflow.

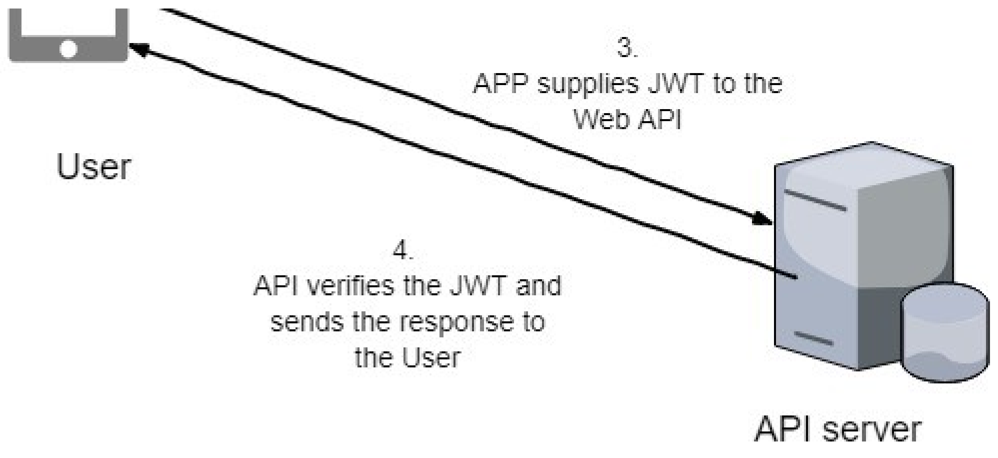
Thus, in this tutorial we would be learning about how to secure the web API that you've have built using JWT.

What is JWT?

JWT, short for *Json Web Token,* is itself an access token (a private key) that is given to authenticated user which gives them the right to access your API endpoints.

SO, a user logins in to your app, the app verifies that the user is who they say they are (authenticate), then the app issues an access token (JWT) to the user which the user in turns use to access your Web API but before your API serves anything, it makes sure that the app was actually the one who gave the user the access token…. that's the trip.




User
3.
APP supplies JWT to the Web API
4.
API verifies the JWT and sends the response to the User
API server

access token

One last thing:

The JWT is a private key generated by encrypting the payload, the header and the signature using some secret key. And can be verified using this same secret key.

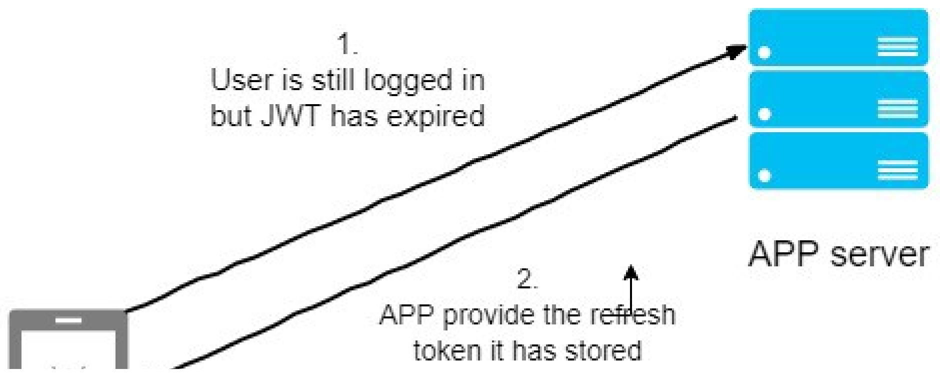
No fears about not explaining what the payload, the header and the signature, it will be clear in no time.

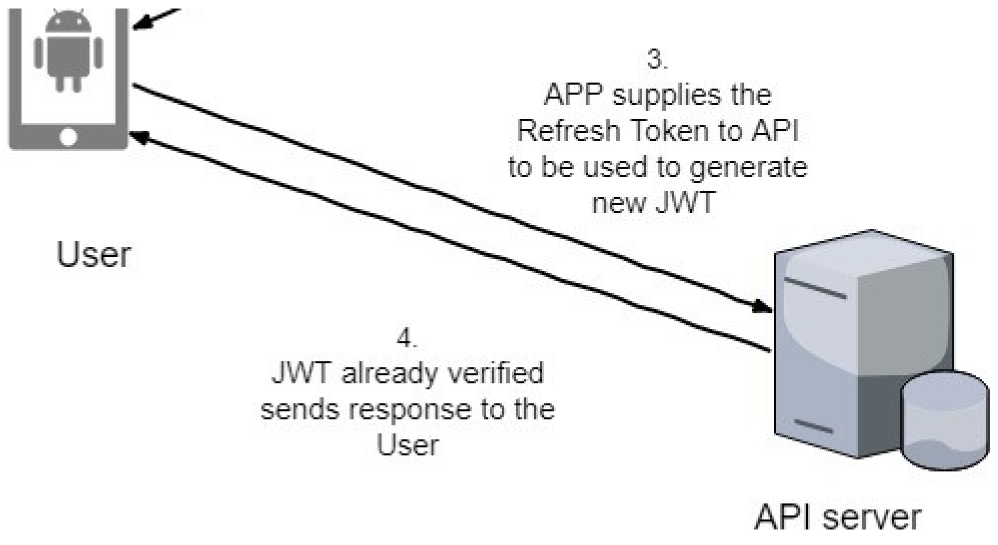
But the thing with **JWT** is that it **expires**. And when it expires, you wouldn't want to ask the user to login again, right?

So what will you do? Refresh Token

What is Refresh Token?

I mean, when creating the **JWT** why don't we create a certain "encrypted string " called refresh token, that is saved on the user app database. Such that when the **JWT** expires, we only need to supply this refresh token and if it is verified, we can generate another **JWT + refresh token** and the cycle goes on…


1.
User is still logged in but JWT has expired
APP server
2.
APP provide the refresh token it has stored


3.
APP supplies the Refresh Token to API to be used to generate new JWT
User
4.
JWT already verified sends response to the User
API server

refresh token

> *JWT makes sure that those accessing your API are authenticated, while Refresh token makes sure you do not get on the nerves of your user by asking them to login again.*

Let's Build, shall we?

I would be using Visual Studio 2017, but feel free to use visual studio code.

If you are using Visual studio 2017 like me, head on to create a new web API project with the template given, build and run to make sure there's no problem.

For Visual studio code using the dotnet cli run `dotnet new webapi -n <projectname>` then run a `dotnet restore` to build it then a `dotnet run` to make sure our API is up.

We are up and running. Next…the models!

To make things a lot easier, we are using two models, one for the **user login info** and the other for the **refresh token.**

Create two classes and write in this code:

Now the Logic

Head on to the controllers folder and add another class called `AccountsController.cs` (takes care of user logging in a providing the access token).

And since we are trying to make things as simple as possible, we would have a custom user we are want to accept.

What have we written?

We already pointed out that we would have a custom user, hence `request.Username == "Jane" && request.Password == "Password!"`

`key` It is the secret key that is used to encrypt the data and it is gotten from the configuration file which in this case is the file name `applicationsettings.json`

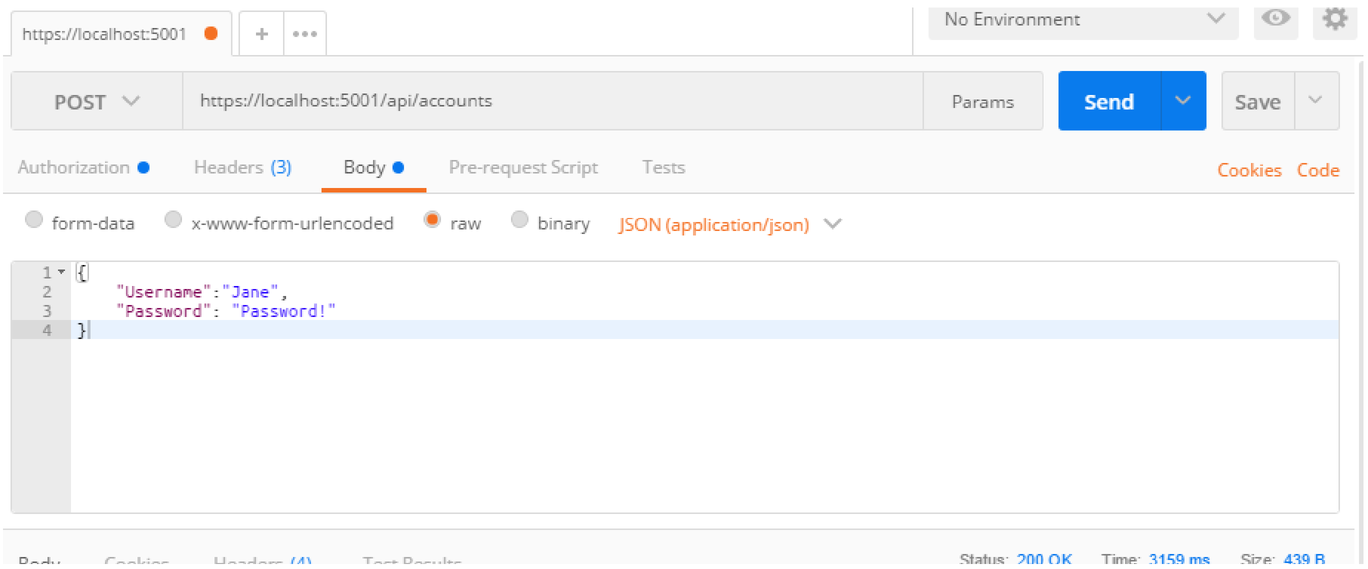
`cred` is when we are creating the type of security we want using the Security Algorithm of type SHA256.

`token` that's what we have been talking about that makes our app secure and it consists of the `payload` which in this case is the issuer, audience, claims, expires; the `signature` which is the signingCredentials.

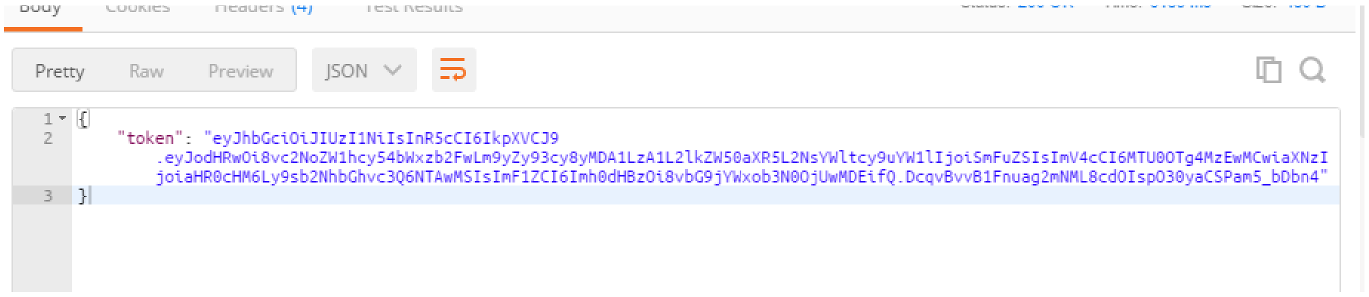
Finally, update the ConfigureServices and the Configure method of the `startup.cs` class as follows:

We are set and ready to run!

I will be using postman, but feel free to use anything that allows you to deliver a payload to an API endpoint



```
{
    "Username":"Jane",
    "Password": "Password!"
}
```



```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSmFuZSIsImV4cCI6MTU0OTg4MzEwMCwiaXNzI
        joiaHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMSIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjUwMDEifQ.DcqvBvvB1Fnuag2mNML8cdOIspO30yaCSPam5_bDbn4"
}
```

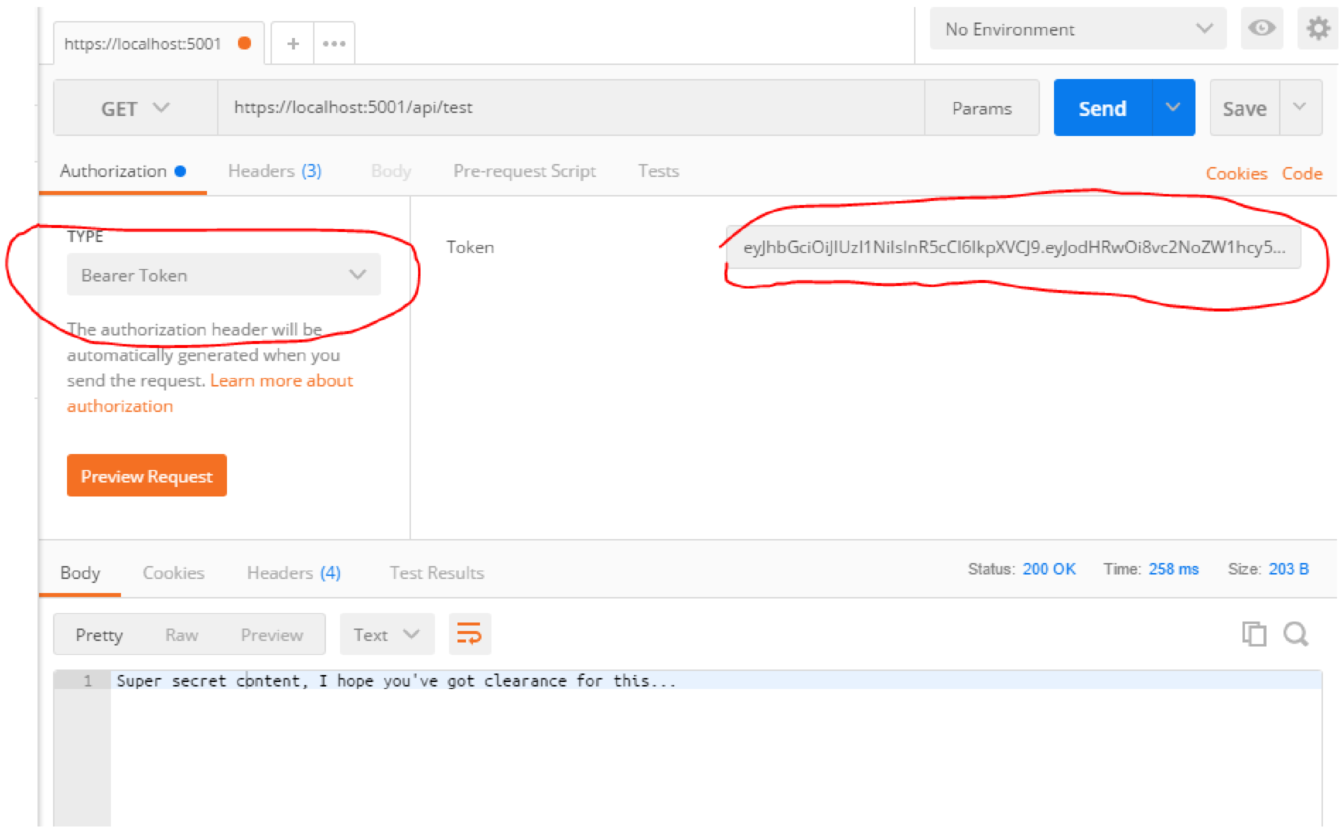
using postman to consume the API

Yay! it works…we have successfully created our access token.

Now let’s use it

Let’s create a controller that requires the access token to display its result:

just some controller that returns a string if the access token is supplied



```
Super secret content, I hope you've got clearance for this...
```

postman consuming Web API

thumbs up! we have done a great job.

Now, we could stop here if we want to, but then we would have to deal with our consumers anger when it tells them to login after the access token has expired. Thus, we are not stopping, we are moving forward…refresh tokens the way to go!

Adding Refresh Token functionality

Still remember our `refreshToken` class? That's what we are using now.

First of, let's set up a database functionality to help store our ` refreshToken. To do this create an `AppDbContext.cs` class and update the ConfigureServices method of the `startup.cs` file

Then, let's update our `AccountController` to include a `RefreshToken` method update our `login` method and include `_db` in the controller:

We are almost done…

The new thing is this:

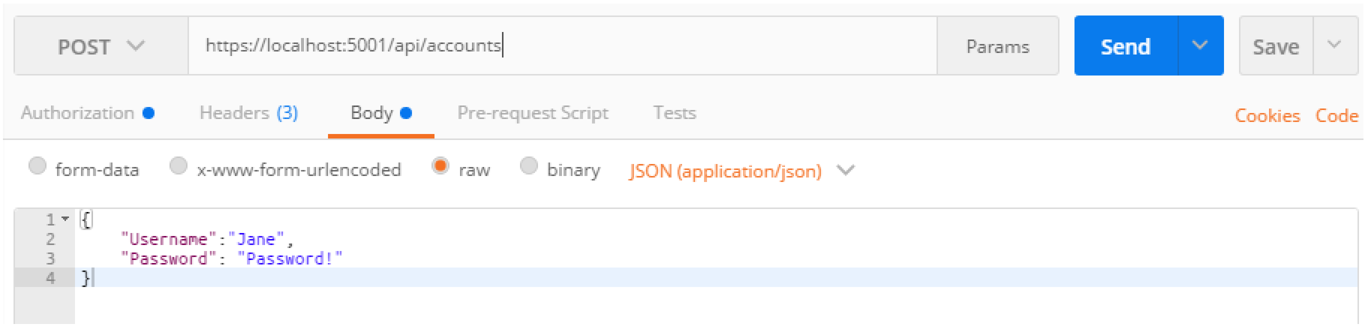
`_refreshTokenObj` is created on login and added to the database

And for the `RefreshToken` method, the `refreshToken` string parameter is posted to it and it verifies if there is any in the database that matches it, if there is it creates a new JWT and a new refreshToken and the cycle begins.

Now let's test again…

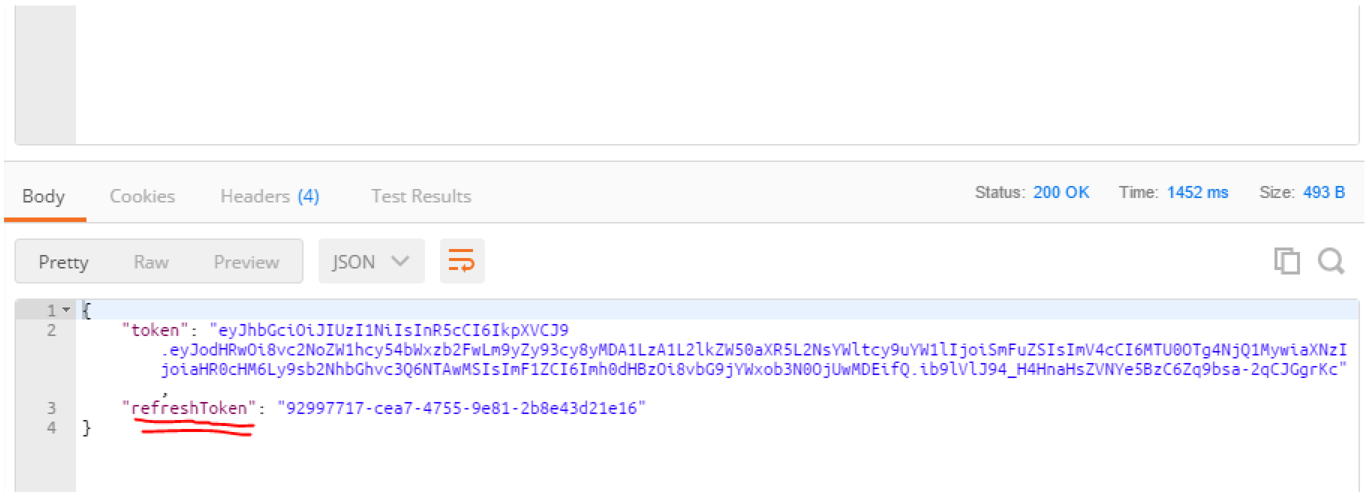
First try the previous access token, it should give you an Unauthorized Error 401

So, let's login again, but this time if the access token expires we would use the refresh token…



https://localhost:5001/api/accounts

```
{
    "Username":"Jane",
    "Password": "Password!"
}
```

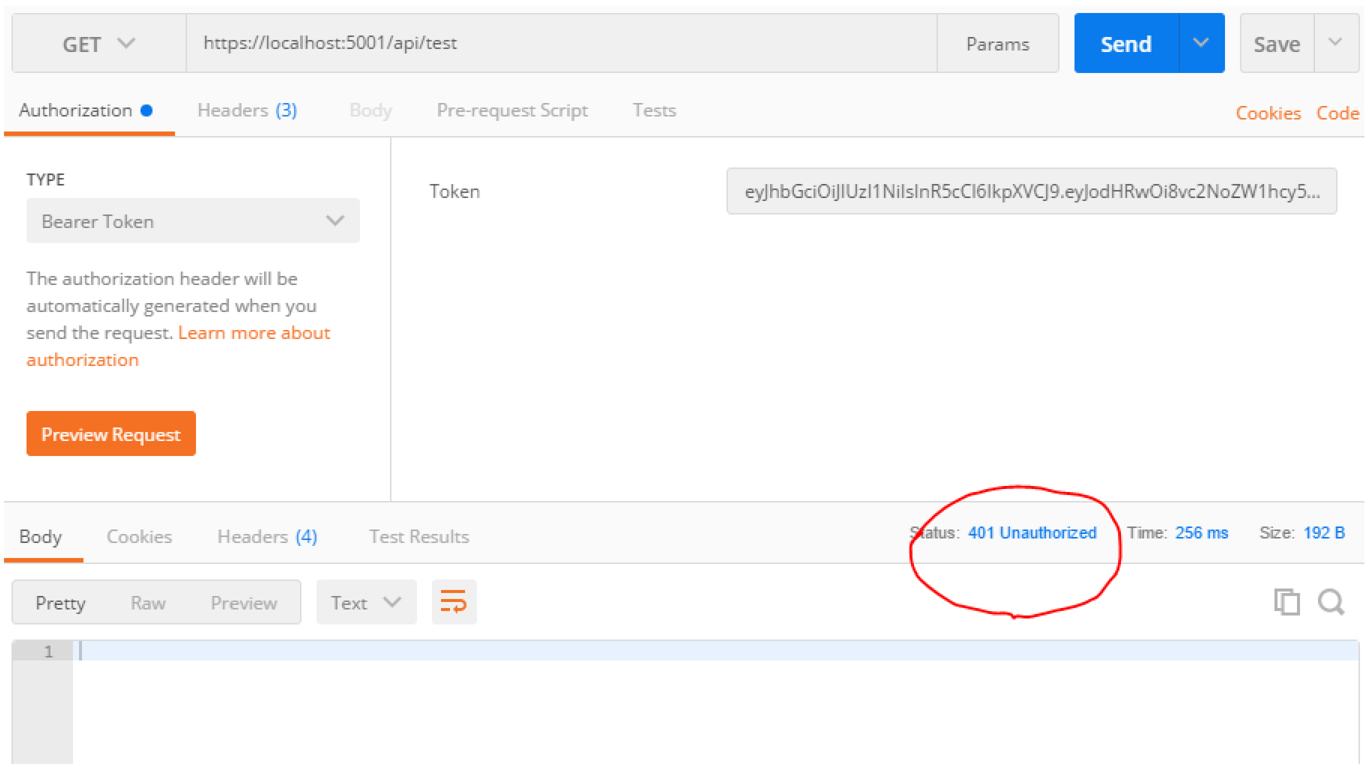


Status: 200 OK Time: 1452 ms Size: 493 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSmFuZSIsImV4cCI6MTU0OTg4NjQ1MywiaXNzI
      joiaHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMSIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjUwMDEifQ.ib9lVlJ94_H4HnaHsZVNYe5BzC6Zq9bsa-2qCJGgrKc"
      ,
    "refreshToken": "92997717-cea7-4755-9e81-2b8e43d21e16"
}
```

here we go! looking alive already!!

Wait about 1 minute then try to hit the test controller



https://localhost:5001/api/test

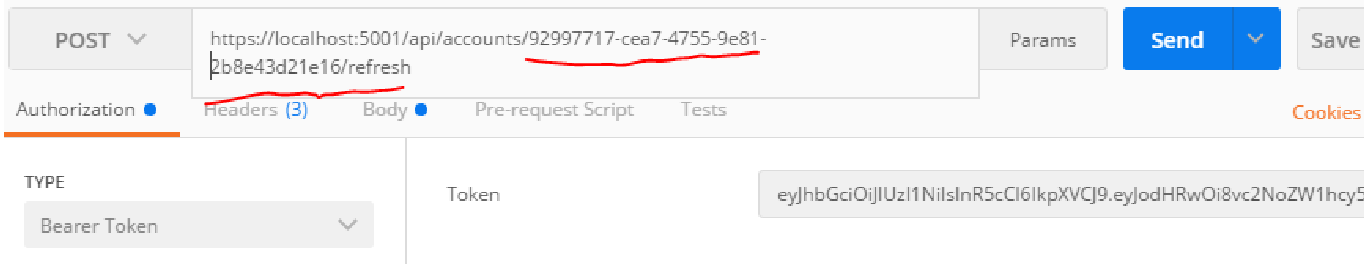
TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5...

Status: 401 Unauthorized Time: 256 ms Size: 192 B

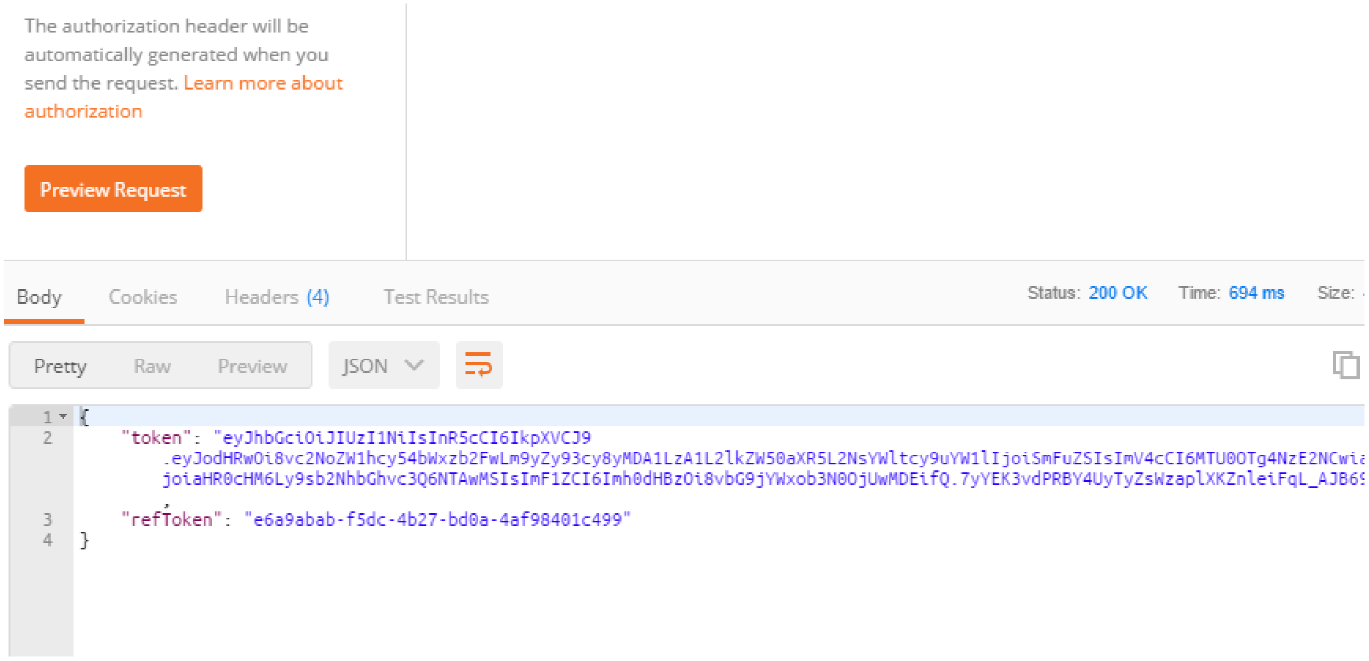
let's hit the refreshToken route `{refreshToken}/refresh`



https://localhost:5001/api/accounts/92997717-cea7-4755-9e81-2b8e43d21e16/refresh

TYPE: Bearer Token

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5


The authorization header will be automatically generated when you send the request. Learn more about authorization
Preview Request
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 694 ms
Size:
Pretty
Raw
Preview
JSON

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSmFuZSIsImV4cCI6MTU0OTg4NzE2NCwia
        joiaHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMSIsImF1ZCI6Imh0dHBzOi8vbG9jYWxob3N0OjUwMDEifQ.7yYEK3vdPRBY4UyTyZsWzaplXKZnleiFqL_AJB6
        ,
    "refToken": "e6a9abab-f5dc-4b27-bd0a-4af98401c499"
}
```

there we go a newly generated JWT that we can use to continue accessing our API.

Conclusion

This is obviously a bare bones tutorial towards JWT and Refresh token, but the aim here is to understand how it works and not write a production ready code.

You can tinker with it by including things like `expirationTime` for the refresh token or using an encrypted algorithm to generate `refreshToken` string and not a `Guid` .

My point being, have fun with it. Here is the link https://github.com/mrsimi/jwt-refreshtoken-webapi to the full code on github

Honest advice here, if you are going to write any production code, it would be way easier to use the frameworks and libraries that have been specifically built for this sort of stuff OAuth, OpenId etc. Whatever makes you work easier. Stay loyal to the craft not the tool.

Thanks for reading and feel free to comment below…:)

Peace

API Jwt Jwt Token Refresh Token Dotnet Core