



## By Vitor Freitas

I'm a passionate software developer and researcher from Brazil, currently living in Finland. I write about Python, Django and Web Development on a weekly basis. [Read more.](#)



## TUTORIAL

---

# How to Use JWT Authentication with Django REST Framework

 Dec 19, 2018  7 minutes read  15 comments  38,335 views



(Picture: <https://www.pexels.com/photo/black-and-gray-laptop-computer-on-brown-wooden-surface-1035592/>)

JWT stand for **JSON Web Token** and it is an authentication strategy used by client/server applications where the client is a Web application using JavaScript and some frontend framework like Angular, React or VueJS.

In this tutorial we are going to explore the specifics of JWT authentication. If you want to learn more about Token-based authentication using Django REST Framework (DRF), or if you want to know how to start a new DRF project you can read this tutorial: [How to Implement Token Authentication using Django REST Framework](#). The concepts are the same, we are just going to switch the authentication backend.

- [How JWT Works?](#)
- [Installation & Setup](#)
- [Example Code](#)
- [Usage](#)
  - [Obtain Token](#)

- [Refresh Token](#)
  - [What's The Point of The Refresh Token?](#)
  - [Further Reading](#)
- 

## How JWT Works?

The JWT is just an authorization token that should be included in all requests:

A screenshot of a terminal window with a light gray background. It shows a curl command: `curl http://127.0.0.1:8000/hello/ -H 'Authorization: Bearer eyJ0eXAiOiJKVj0:'`. The text is in a monospaced font, and the command is highlighted with a light gray selection bar. There are scrollbars on the right and bottom of the terminal window.

```
curl http://127.0.0.1:8000/hello/ -H 'Authorization: Bearer eyJ0eXAiOiJKVj0:'
```

The JWT is acquired by exchanging an username + password for an **access token** and an **refresh token**.

The **access token** is usually short-lived (expires in 5 min or so, can be customized though).

The **refresh token** lives a little bit longer (expires in 24 hours, also customizable). It is comparable to an authentication session. After it expires, you need a full login with username + password again.

Why is that?

It's a security feature and also it's because the JWT holds a little bit more information. If you look closely the example I gave above, you will see the token is composed by three parts:

A diagram showing the structure of a JWT token. It consists of a light gray rounded rectangle containing the text 'xxxxx.yyyyy.zzzzz'. On the right side of the rectangle, there are two small gray triangles, one pointing up and one pointing down, indicating a scrollable area.

```
xxxxx.yyyyy.zzzzz
```

Those are three distinctive parts that compose a JWT:

```
header.payload.signature
```

So we have here:

```
header = eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
payload = eyJ0b2t1bGl90eXB1IjoIYWVjZXRzIiwiaXNjaXhwIjoNTQzODI4NDMxLCJqdGkiOiI:
signature = Ju70kdcaHKn1Qaz8H42zrOYk0Jx9kIckTn9Xx7vhikY
```

This information is encoded using Base64. If we decode, we will see something like this:

## header

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

## payload

```
{
  "token_type": "access",
  "exp": 1543828431,
  "jti": "7f5997b7150d46579dc2b49167097e7b",
  "user_id": 1
}
```

## signature

The signature is issued by the JWT backend, using the header base64 + payload base64 + `SECRET_KEY`. Upon each request this signature is verified. If any information in the header or in the payload was changed by the client it will invalidate the signature. The only way of checking and validating the signature is by using your application's `SECRET_KEY`. Among other things, that's why you should always keep your `SECRET_KEY` **secret!**

---

## Installation & Setup

For this tutorial we are going to use the `django-rest-framework-simplejwt` library, recommended by the DRF developers.

```
pip install django-rest-framework-simplejwt
```

### settings.py

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

### urls.py

```
from django.urls import path
from rest_framework_simplejwt import views as jwt_views

urlpatterns = [
    # Your URLs...
    path('api/token/', jwt_views.TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', jwt_views.TokenRefreshView.as_view(), name='token_refresh'),
]
```

---

## Example Code

For this tutorial I will use the following route and API view:

### views.py

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated

class HelloView(APIView):
    permission_classes = (IsAuthenticated,)

    def get(self, request):
        content = {'message': 'Hello, World!'}
        return Response(content)
```

### urls.py

```
from django.urls import path
from myapi.core import views

urlpatterns = [
    path('hello/', views>HelloView.as_view(), name='hello'),
]
```

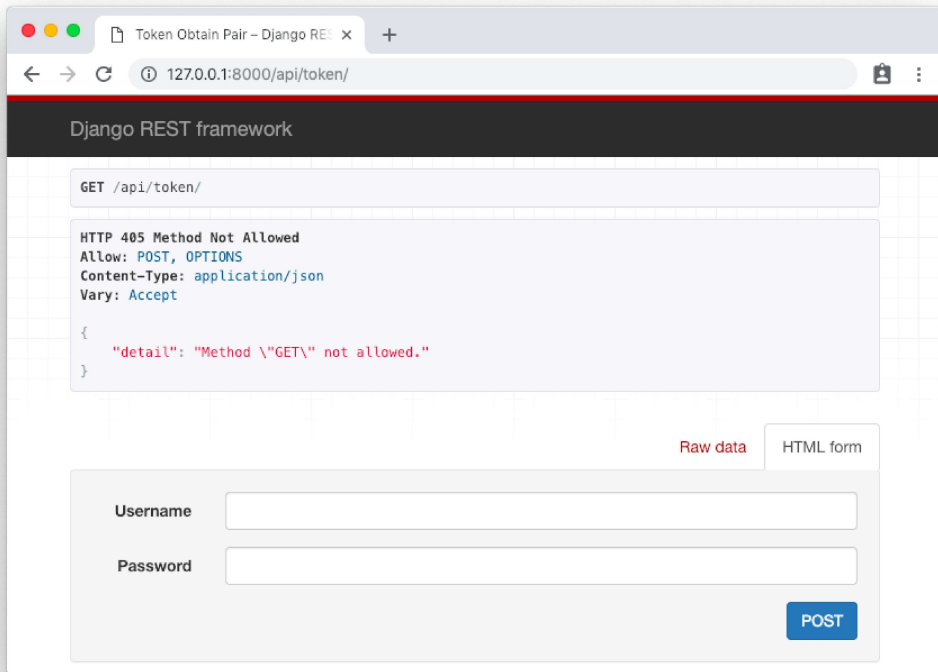
---

## Usage

I will be using [HTTPie](#) to consume the API endpoints via the terminal. But you can also use [cURL](#) (readily available in many OS) to try things out

locally.

Or alternatively, use the DRF web interface by accessing the endpoint URLs like this:



## Obtain Token

First step is to **authenticate and obtain the token**. The endpoint is `/api/token/` and it only accepts **POST** requests.

```
http post http://127.0.0.1:8000/api/token/ username=vtor password=123
```

```

drf-jwt-example — -bash — 113x17
[Vitor-MacBookAir:drf-jwt-example vitorfs$ http post http://127.0.0.1:8000/api/token/ username=vitor password=123 ]
HTTP/1.1 200 OK
Allow: POST, OPTIONS
Content-Length: 438
Content-Type: application/json
Date: Wed, 19 Dec 2018 12:52:39 GMT
Server: WSGIServer/0.2 CPython/3.6.7
Vary: Accept
X-Frame-Options: SAMEORIGIN

{
  "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoiaWVhbnQ1I3MmIzYjI0YjNmOGI1MjJlNTIjMzI0YjF9.D92tTuVl_YeNk3tILGHTcn6t8cxLCBxz9FKD3qzhUg8",
  "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoiaWVhbnQ1I3MmIzYjI0YjNmOGI1MjJlNTIjMzI0YjF9.D92tTuVl_YeNk3tILGHTcn6t8cxLCBxz9FKD3qzhUg8"
}

```

So basically your response body is the two tokens:

```

{
  "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoiaWVhbnQ1I3MmIzYjI0YjNmOGI1MjJlNTIjMzI0YjF9.D92tTuVl_YeNk3tILGHTcn6t8cxLCBxz9FKD3qzhUg8",
  "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoiaWVhbnQ1I3MmIzYjI0YjNmOGI1MjJlNTIjMzI0YjF9.D92tTuVl_YeNk3tILGHTcn6t8cxLCBxz9FKD3qzhUg8"
}

```

After that you are going to store both the **access token** and the **refresh token** on the client side, usually in the [localStorage](#).

In order to access the protected views on the backend (i.e., the API endpoints that require authentication), you should include the **access token** in the header of all requests, like this:

```

http http://127.0.0.1:8000/hello/ "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoiaWVhbnQ1I3MmIzYjI0YjNmOGI1MjJlNTIjMzI0YjF9.D92tTuVl_YeNk3tILGHTcn6t8cxLCBxz9FKD3qzhUg8"

```



```
drf-jwt-example — -bash — 113x17
Vitor-MacBookAir:drf-jwt-example vitorfs$ http http://127.0.0.1:8000/hello/ "Authorization: Bearer eyJ0eXAiOiJKV1QiOiJLC3RhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1Ijo1YWNjZXMzIiwiaXNjaXo6NTQ1MjE0MjAwLCJqdGkiOiJlMGQxZDY2MjE5ODc4ZTY3OWY0NjM0ZDU2NTQ2YTlwcisInVzZXJkaWQ1OjF9.9eHat3CyRQYnb5EdcgVFzUyMobXzxIAVh_1AggyvzCE"
HTTP/1.1 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 27
Content-Type: application/json
Date: Wed, 19 Dec 2018 12:55:28 GMT
Server: WSGIServer/0.2 CPython/3.6.7
Vary: Accept
X-Frame-Options: SAMEORIGIN

{
  "message": "Hello, World!"
}
Vitor-MacBookAir:drf-jwt-example vitorfs$
```

You can use this **access token** for the next five minutes.

After five min, the token will expire, and if you try to access the view again, you are going to get the following error:

```
http http://127.0.0.1:8000/hello/ "Authorization: Bearer eyJ0eXAiOiJKV1QiOiJLC3RhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1Ijo1YWNjZXMzIiwiaXNjaXo6NTQ1MjE0MjAwLCJqdGkiOiJlMGQxZDY2MjE5ODc4ZTY3OWY0NjM0ZDU2NTQ2YTlwcisInVzZXJkaWQ1OjF9.9eHat3CyRQYnb5EdcgVFzUyMobXzxIAVh_1AggyvzCE"
401 Unauthorized
```

```

drf-jwt-example — -bash — 113x26
Vitor-MacBookAir:drf-jwt-example vitorfs$ http http://127.0.0.1:8000/hello/ "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1Ijo1VWVhZmZlIiwiaWF0IjE0MjU2NTQyOTIwMCI6InVzZXJfYWQ1OjF9.9eHat3CyRQYnb5EdcgVFzUyMobXzxIAVh_1AggyvzCE"
HTTP/1.1 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Length: 183
Content-Type: application/json
Date: Wed, 19 Dec 2018 12:58:00 GMT
Server: WSGIServer/0.2 CPython/3.6.7
Vary: Accept
WWW-Authenticate: Bearer realm="api"
X-Frame-Options: SAMEORIGIN

{
  "code": "token_not_valid",
  "detail": "Given token not valid for any token type",
  "messages": [
    {
      "message": "Token is invalid or expired",
      "token_class": "AccessToken",
      "token_type": "access"
    }
  ]
}

Vitor-MacBookAir:drf-jwt-example vitorfs$

```

## Refresh Token

To get a new **access token**, you should use the refresh token endpoint

`/api/token/refresh/` posting the **refresh token**:

```
http post http://127.0.0.1:8000/api/token/refresh/ refresh=eyJ0eXAiOiJKV1Q
```

```
drf-jwt-example — -bash — 113x17
```

```
Vitor-MacBookAir:drf-jwt-example vitorfs$ http post http://127.0.0.1:8000/api/token/refresH refresh=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bGl9eXBIIjoimVmcmVzaCI6ImV4cCMTU0NTMwODIyMiwiYWVnRmRpIjoiaWZAYOGFIjnc0ZTlndjNDMDImNzLWYyZjg3MTU1NGUxDDBzOQILCjEic2VyX2kiKjoxfQ.Md8A03dRqBvWYWeZsd_A1J39z6b6HEwIUZ7i1o1PE HTTP/1.1 200 OK
```

```
Allow: POST, OPTIONS
Content-Length: 218
Content-Type: application/json
Date: Wed, 19 Dec 2018 13:11:18 GMT
Server: WSGIServer/0.2 CPython/3.6.7
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

```
{
  "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bGl9eXBIIjoimYmVnRmRpIjoiaWZAYOGFIjnc0ZTlndjNDMDImNzLWYyZjg3MTU1NGUxDDBzOQILCjEic2VyX2kiKjoxfQ.Md8A03dRqBvWYWeZsd_A1J39z6b6HEwIUZ7i1o1PE"
}
```

The return is a new **access token** that you should use in the subsequent requests.

The **refresh token** is valid for the next 24 hours. When it finally expires too, the user will need to perform a full authentication again using their username and password to get a new set of **access token + refresh token**.

## What's The Point of The Refresh Token?

At first glance the **refresh token** may look pointless, but in fact it is necessary to make sure the user still have the correct permissions. If your **access token** have a long expire time, it may take longer to update the information associated with the token. That's because the authentication check is done by cryptographic means, instead of querying the database and verifying the data. So some information is sort of cached.

There is also a security aspect, in a sense that the **refresh token** only travel in the POST data. And the **access token** is sent via HTTP header,

which may be logged along the way. So this also give a short window, should your **access token** be compromised.

---

## Further Reading

This should cover the basics on the backend implementation. It's worth checking the [djangoRESTframework\\_simplejwt settings](#) for further customization and to get a better idea of what the library offers.

The implementation on the frontend depends on what framework/library you are using. Some libraries and articles covering popular frontend frameworks like angular/react/vue.js:

- [Angular JWT library](#)
- [Angular 2 JWT library](#)
- [Secure Your React and Redux App with JWT Authentication](#)
- [Authenticating via JWT using Django, Axios, and Vue](#)

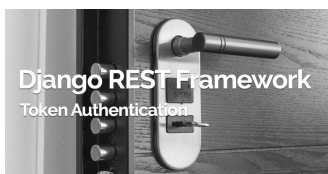
The code used in this tutorial is available at [github.com/sibtc/drf-jwt-example](https://github.com/sibtc/drf-jwt-example).

---

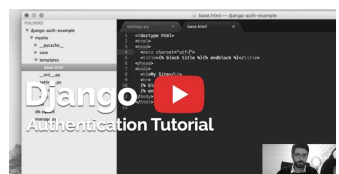
## Related Posts



[How to Save Extra Data to a Django](#)



[How to Implement Token Authentication](#)



[Django Authentication Video Tutorial](#)

[django](#)[drf](#)[rest](#)[api](#)[auth](#)[jwt](#)


Share this post



CommentsCommunity

1

Login

 Recommend 5

 Tweet

 Share

Sort by Best

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name



**Philip J Cox** • 4 months ago

After further digesting and going over this blog post, it really tells us nothing. There is so much missing to actually get this library working, or am I missing something? I just keep a response from the API the credentials do not exist.

5



• Reply • Share >



**Berk Elmas** → Philip J Cox

• 2 months ago

you need to have a valid user with the username and password values on the body

^ | v • Reply • Share ›



**KIM ATIBULA** • 7 hours ago

Hi How can i set username and password ?

^ | v • Reply • Share ›



**Philip J Cox** • 4 months ago

There is a security concern within this post people need to be aware of. I love a good post with good information in it, so I don't mean any disrespect. Here is a link from AuthO, a company who really understands authentication security, and they propose that Refresh tokens should never be stored on the client side, this is extremely dangerous. Refresh tokens are for internal server communication authentication.

<https://auth0.com/docs/tokens>

^ | v • Reply • Share ›



**Berk Elmas** → Philip J Cox

• 2 months ago

First, it is widely recommended not to have password and other sensitive credentials in token. So, simplejwt library lets you override which values to store in the token.

<https://pypi.org/project/django-simplejwt/> from this link, you can have a look at customizing token claims; by doing this, you can restrict security vulnerability to a limited time which is the expiration time

time which is the expiration time interval of the token in our case. Secondly, jwt tokens are widely used in mobile apps and modern Javascript frameworks like React, Vue etc. to handle authentication; if you have another way of solution, please let us know about it.

1 ^ | v • Reply • Share ›



**William Buck** • 5 months ago

What would be the best form of practice for refresh and access tokens regarding a timed out access token?

Would the front end try to access the view and if "invalid\_token" code is received then send a request to refresh then token and try the original request again?

^ | v • Reply • Share ›



**Ankit Kathuria** • 6 months ago

How to log out or invalidate the token?

^ | v • Reply • Share ›



**Berk Elmas** ➔ Ankit Kathuria  
• 2 months ago

just delete the token from client side on local storage or session storage and then you are logged out.

^ | v • Reply • Share ›



**Philip J Cox** ➔ Ankit Kathuria  
• 4 months ago

This is also a discussion I am currently having. I would be interested in hearing the authors opinion on this, if he is so inclined to reply to his readers. When we issue a token, are we committed to

we issue a token, are we committed to allowing that user those sets of permissions for that given time, what if we change the permissions before the token times out and the user can continue requests? Is this how the refresh token works?

^ | v • Reply • Share ›



**Jhon Sanz** • 6 months ago

You are amazing, thank you so much friend

^ | v • Reply • Share ›

## Subscribe to our Mailing List

Receive updates from the Blog!

SUBSCRIBE

## Popular Posts



# django

## Extend User Model

[How to Extend Django User Model](#)



[How to Setup a SSL Certificate on Nginx for a Django Application](#)



[How to Deploy a Django Application to Digital Ocean](#)

