NEW NEW Introducing DigitalOcean Managed MongoDB — a fully managed, database as a service for modern apps

TUTORIAL

How To Use JSON Web Tokens (JWTs) in Express.js

Node.js

By Danny Denenberg

Last Validated on March 22, 2021 • Originally Published on February 19, 2020 186.4k

Introduction

JSON Web Tokens (JWTs) supports authorization and information exchange.

One common use case is for allowing clients to preserve their session information after logging in. By storing the session information locally and passing it to the server for authentication when making requests, the server can trust that the client is a registered user.

In this article, you will learn about the applications of JWTs in a server-client relationship using Node.js and vanilla JavaScript.

Prerequisites

To follow along with this article, you will need the following installed on your machine:

- Node.js installed locally, which you can do by following How to Install Node.js and Create a Local Development Environment.

Step 1 — Generating a Token

`jsonwebtoken` is an implementation of JSON Web Tokens.

You can add it to your JavaScript project by running the following command in your terminal:

```
$ npm install jsonwebtoken
```

And import it into your files like so:

```
const jwt = require('jsonwebtoken');
```

To sign a token, you will need to have 3 pieces of information:

1. The token secret
2. The piece of data to hash in the token
3. The token expire time

The *token secret* is a long random string used to encrypt and decrypt the data.

To generate this secret, one option is to use Node.js's built-in `crypto` library, like so:

```
> require('crypto').randomBytes(64).toString('hex')
// '09f26e402586e2faa8da4c98a35f1b20d6b033c6097befa8be3486a829587fe2f90a832bd3ff9d42710a4da6
```

Warning: Be careful! If your `secret` is simple, the token verification process will be much easier to break by an unauthorized intruder.

Now, store this secret in your project's `.env` file:

.env

```
TOKEN_SECRET=09f26e402586e2faa8da4c98a35f1b20d6b033c60...
```

To bring this token into a Node.js file and to use it, you have to use `dotenv`:

```
$ npm install dotenv
```

And import it into your files like so:

```
const dotenv = require('dotenv');

// get config vars
dotenv.config();

// access config var
process.env.TOKEN_SECRET;
```

The *piece of data* that you hash in your token can be something either a user ID or username or a much more complex object. In either case, it should be an *identifier* for a *specific* user.

The *token expire time* is a string, such as 1800 seconds (30 minutes), that details how long until the token will be invalid.

Here's an example of a function for signing tokens:

```
function generateAccessToken(username) {
  return jwt.sign(username, process.env.TOKEN_SECRET, { expiresIn: '1800s' });
}
```

This can be sent back from a request to sign in or log in a user:

```
app.post('/api/createNewUser', (req, res) => {
  // ...

  const token = generateAccessToken({ username: req.body.username });
  res.json(token);

  // ...
});
```

This example takes the `username` value from the `req` (*request*). And provides the token as the `res` (*response*).

That concludes how `jsonwebtoken`, `crypto`, and `dotenv` can be used to generate a JWT.

Step 3 — Authenticating a Token

There are many ways to go about implementing a JWT authentication system in an Express.js application.

One approach is to utilize the middleware functionality in Express.js.

How it works is when a request is made to a specific route, you can have the `(req, res)` variables sent to an intermediary function before the one specified in the `app.get((req, res) => {})`.

The middleware is a function that takes parameters of `(req, res, next)`.

- The `req` is the sent request (GET, POST, DELETE, PUT, etc.).
- The `res` is the response that can be sent back to the user in a multitude of ways (`res.sendStatus(200)`, `res.json()`, etc.).
- The `next` is a function that can be called to move the execution past the piece of middleware and into the actual `app.get` server response.

Here is an example middleware function for authentication:

```
const jwt = require('jsonwebtoken');

function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization']
  const token = authHeader && authHeader.split(' ')[1]

  if (token == null) return res.sendStatus(401)

  jwt.verify(token, process.env.TOKEN_SECRET as string, (err: any, user: any) => {
    console.log(err)

    if (err) return res.sendStatus(403)

    req.user = user

    next()
  })
}
```

An example request using this middleware function would resemble something like this:

```
GET https://example.com:4000/api/userOrders
Authorization: Bearer JWT_ACCESS_TOKEN
```

And an example of a request that would use that piece of middleware would resemble something like this:

```
app.get('/api/userOrders', authenticateToken, (req, res) => {
  // executes after authenticateToken
  // ...
})
```

This code will authenticate the token provided by the client. If it is valid, it can proceed to the request. If it is not valid, it can be handled as an error.

Step 4 — Handling Client-Side Tokens

When the client receives the token, they often want to store it for gathering user information in future requests.

The most popular manner for storing auth tokens is in an `HttpOnly` cookie.

Here's an implementation for storing a cookie using client-side JavaScript code:

```
// get token from fetch request
const token = await res.json();

// set token in cookie
document.cookie = `token=${token}`
```

This approach stores the response locally where they can be referenced for future requests to the server.

That concludes the flow of requesting a token, generating a token, receiving a token, passing a token with new requests, and verifying a token.

Conclusion

In this article, you were introduced to JWTs and one approach to applying them to a Node.js application. This approach relied upon a combination of `jsonwebtoken`, `crypto`, `dotenv`, and

```
express.
```

For another approach to using JWTs, there is How To Implement API Authentication with JSON Web Tokens and Passport.

For more background on JWTs, there is the "Introduction" documentation.

If you'd like to learn more about Node.js, check out our Node.js topic page for exercises and programming projects.

Was this helpful? Yes No

Report an issue

About the authors

Danny Denenberg

is a Community author on DigitalOcean.

Bradley Kouchi

Editor

Still looking for an answer?

Ask a question

Search for more help

RELATED

Join the DigitalOcean Community

Join 1M+ other developers and:

- Get help and share knowledge in Q&A
- Subscribe to topics of interest
- Get courses & tools that help you grow as a developer or small business owner

Join Now

How To Build a Rate Limiter With Node.js on App Platform

Tutorial

How To Install and Use the Yarn Package Manager for Node.js

Tutorial

Comments

12 Comments

Leave a comment...

Sign In to Comment

perryraskin May 20, 2020

0

```
/api/creteNewUser
```

I believe you meant `createNewUser`? :)

Thanks for this post!

Reply Report

followben June 3, 2020

6

Cheers for the tutorial Danny!

Sadly Mario's post doesn't detail the drawbacks complexities and risks of using JWTs for securing a REST backend. They have a place, but your article should point out they're not a one-size-fits-all solution, especially for a backends built with node.js and deployed as a monolith. Revocation and refresh is non-trivial, for instance.

If folks are going to use them in place of sessions, please stop recommending Local Storage to persisting them client-side. For browser-based clients, *the node app* should send & retrieve JWTs via a HTTPS-only secure cookie, with either samesite as strict for known browsers or a separate csrf-token stored locally and validated against the payload. Check this vid for a good overview of the correct approach.

Reply Report

hamishclulee August 22, 2020

0

What about using a session token as an HttpOnly cookie along side the JWT token stored in localstorage?

I need to store some form of indication of user auth on the client, unless I want to consistently poll the backend to make sure my users are authed. Orginally I thought, ok, i'll just store the userid in localstorage, but it was basically an arbitary string, you could have changed it to 'monkeypawmokeypaw' and it would have satisfied the client that a user was logged in. Obviously all sensistive API endpoints are checked against the session token, so using the 'monkeypaw' example would just mean no clientside redirects, but no sensitive info was passed.

So then I thought, JWT and session token together would be the best of both worlds, where the token is stored in localstorage and sent as the bearer headed in all reqs, then the server verifies both JWT and session tokens.

Is using a session token and JWT side by side a bad idea as well?

I'm not well versed here, but after a lot of thought and back and forth, the above feels like the best way to go for me...

But I'm the first to admit I could be missing something..?

Reply Report

erindeji June 4, 2020

1

Really helpful. Was initially going with a wrong token format

// const token = authorization.replace("Bearer ", " ");

This article really helped. Thanks!

Reply Report

sorieil July 2, 2020

0

Thank you. This port helped me a lot.

Reply Report

zubair1103 October 7, 2020

0

How to validate a token in multi nodes environment? For examples there are 3 servers. User logged on server 1 and token is generated there. Now api call is sent to server 1 with token. This request will be validated and data will be returned.

If load balancer redirects this api call to server 2 then? Server 2 does not recognize that token.

How to store token in some shared location and validate from there?

Reply Report

gmwill934 October 12, 2020

1

how is the value of `req.headers['authorization']` set?

Reply Report

maxdbarton97 October 13, 2020

0

https://stackoverflow.com/a/35780539/5108158

Reply Report

alessandroamella December 6, 2020

1

DO NOT STORE THE JWT IN LOCALSTORAGE

If you store it inside localStorage, it's accessible by any script inside your page (which is as bad as it sounds, as an XSS attack can let an external attacker get access to the token).

Don't store it in local storage (or session storage). If any of the third-party scripts you include in your page gets compromised, it can access all your users' tokens.

The JWT needs to be stored inside an httpOnly cookie, a special kind of cookie that's only sent in HTTP requests to the server, and it's never accessible (both for reading or writing) from

JavaScript running in the browser.

Reply Report

ahmedfarooki January 8, 2021

0

I'd like to stress on what *alessandroamella* said, **DO NOT STORE the JWT token in Local Storage**.

Reply Report

jaegerbomb January 27, 2021

0

Very simple and helpful thanks.

Couple of amendments if I may (some already mentioned by others):

`/api/creteNewUser` ⇒ `/api/createNewUser`

`process.env.ACCESS_TOKEN_SECRET as string` should be `process.env.TOKEN_SECRET as string`

Reply Report

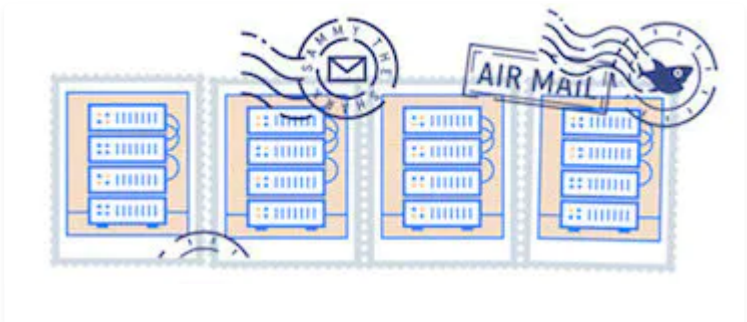
malikkurosaki June 28, 2021

0

thank you, your explanation is very easy to understand

Reply Report

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

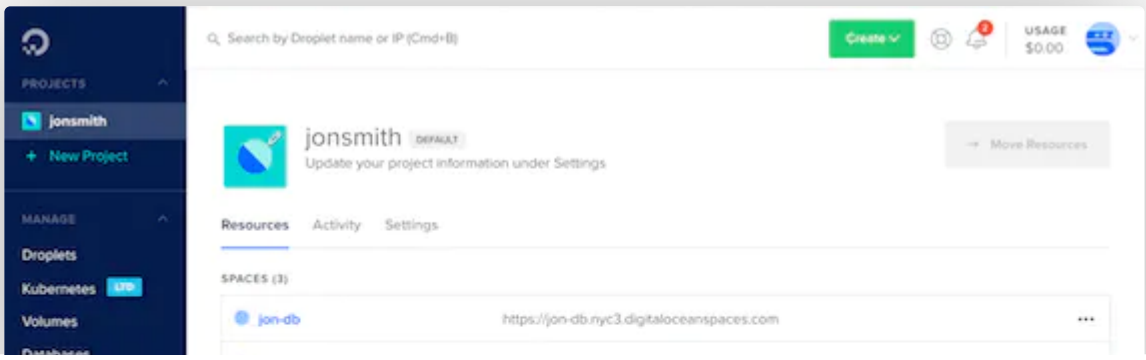
Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More



© 2021 DigitalOcean, LLC. All rights reserved.

Company

About
Leadership
Blog
Careers
Partners
Referral Program
Press
Legal
Security & Trust Center

Products

Pricing
Products Overview
Droplets
Kubernetes

Community

Tutorials
Q&A
Tools and Integrations
Tags

Contact

Get Support
Trouble Signing In?
Sales

Kubernetes
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
API Documentation
Documentation
Release Notes
Tags
Product Ideas
Write for DigitalOcean
Presentation Grants
Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct
Report Abuse
System Status