

# JSON Web Token Authentication With Node.js



Let's now protect some endpoints by making use of JSON Web Tokens. What are JWTs? JSON

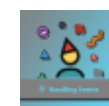
web tokens are base64url encoded JSON Objects which are encrypted by the use of a private key. Embedded within the JWT can be one or more sets of name and value pairs. You might have things like a user id or user name embedded within the token. You can not see this data thanks to the encryption in place, but it can be deciphered on the server since the private key is available to the application. JWT.io has a [great introduction](#) if you would like all the details. For now, let's see how we can use JSON web tokens to protect various endpoints in our Node API.

## Using Authorization Middleware

In the [route and model](#) that we just set up for Platforms of video game systems, we are allowing any user to send a post or put request to the api which modifies data on the server. In fact, pretty much all operations except a simple GET request should have some type of protection. Let's set that up now.

### 1. Add a middleware directory

First up, we can add a middleware directory to hold the authorization middleware that we will be using.



[How To Handle Events In React](#)



[Crud In Laravel 4](#)



[PHP URL Encode Example](#)



[Laravel Vue Component Example](#)



[What are MySQL Operators?](#)



[Introduction To C# And Visual Studio For Beginners](#)



[How To Use WP\\_Query In WordPress](#)



[Dependency Injection for Beginners](#)



[How To Create User Registration in Laravel](#)



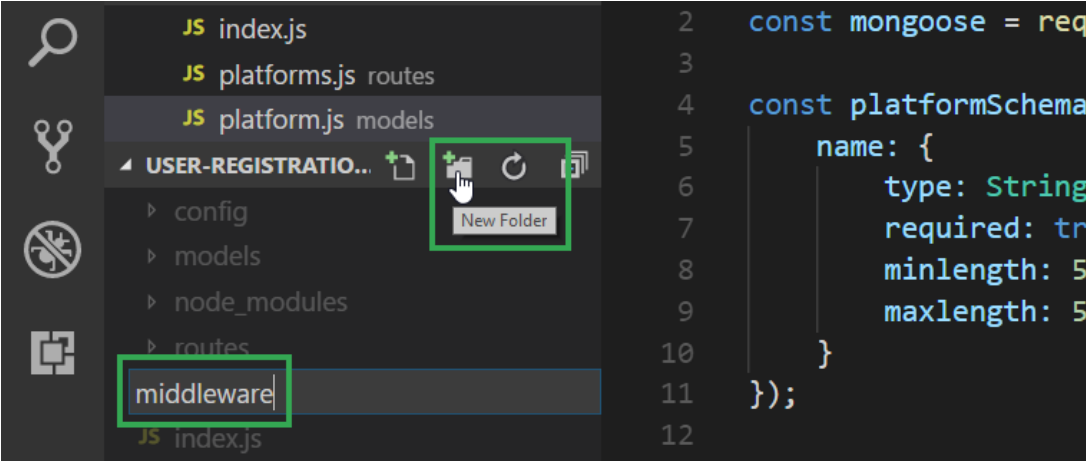
[Combine PHP Functions To Make Your Own](#)



[Twitter Bootstrap Tutorials](#)

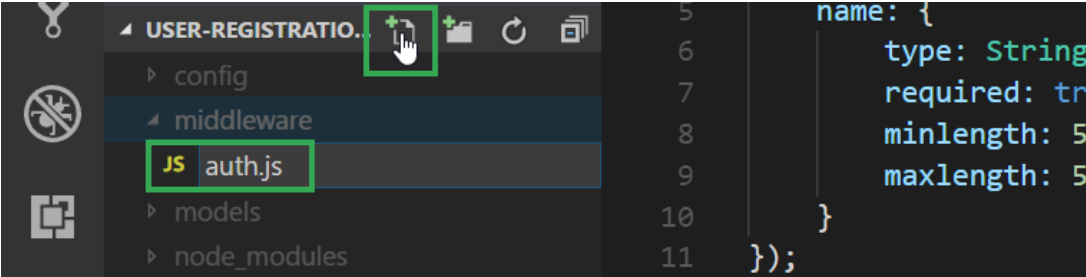


[How To Compare Two Arrays of Data and Calculate Position Differences](#)



## 2. Create an auth.js file

We are creating a new auth.js file to store the logic to read request headers so that this can be used across any endpoint we like.



## 3. Add code to read and verify JWT

In this auth.js file we complete a few steps. First, we require the json web token and config packages. Then we set up a function which is exported so we can use elsewhere as we like. Lastly, we make sure the logic is in place to read the token, and either grant or deny access based on the provided token. Note the use of the [jwt.verify\(\)](#) function which does the heavy lifting for us.

### /middleware/auth.js

```
1 const jwt = require('jsonwebtoken');
2 const config = require('config');
3
4 module.exports = function (req, res, next) {
5
6   const token = req.header('x-auth-token');
7   if (!token) {
8     return res.status(401).send('Access denied. No JWT');
9   }
10
11   try {
12     const decoded = jwt.verify(token, config.get('Private'));
13     req.user = decoded;
14     next();
15   }
16   catch (ex) {
17     res.status(400).send('Invalid JWT.');
```



[Working With Common MySQL String Functions](#)



[ES6 Modules With Traceur.js](#)



[C# Classes For A CRM Application](#)



[PHP Booleans and Constants Tutorial](#)



[Laravel Collections Tutorial](#)



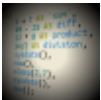
[Underscore JS Map Function](#)



[Configure Form and Button For A Post Request](#)



[Conditional Rendering In React](#)



[MySQL Function Tutorial](#)



[Create React App Tutorial](#)



[Vue.js Express Tutorial](#)



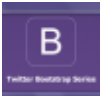
[VueJS Bootstrap Pagination Component](#)



[Even More Awesome WordPress Fundamentals](#)



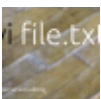
[CSS Grid Tutorial](#)



[What Is Twitter Bootstrap?](#)



[Traverse the DOM With jQuery.](#)



[Vi Editor Tutorial For Beginners](#)



[Process HTML Forms With PHP](#)



[Most Popular JavaScript Frameworks](#)

# Protecting Routes With our new Middleware

Ok we created the middleware, now let’s use it! This is quite easy. To start, we are going to apply the auth middleware to the POST route in the platforms.js routes file. All we have to do is require the new auth middleware, then use it as the second argument to the router.post() function.

```
const auth = require('../middleware/auth');
const { Platform, validate } = require('../models/platform');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

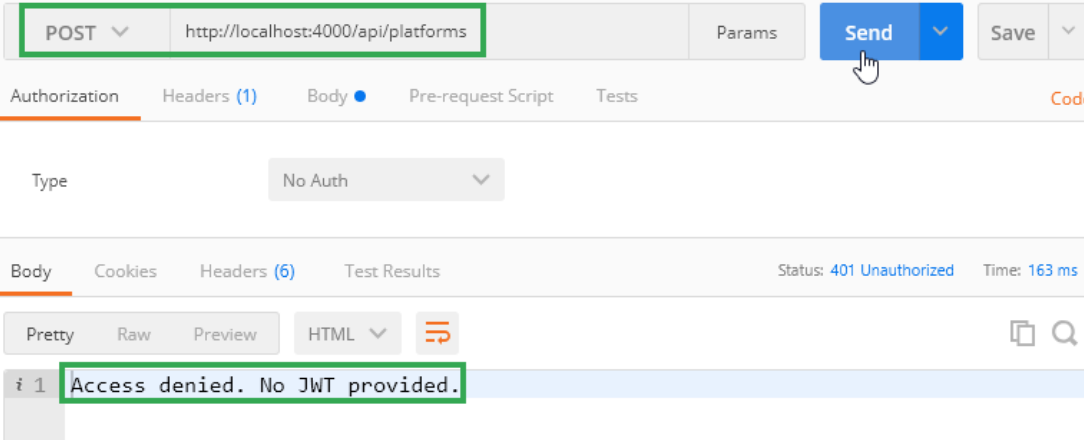
router.post('/', auth, async (req, res) => {
  const { error } = validate(req.body);
  if (error) {
    return res.status(400).send(error.details[0].message);
  }

  let platform = new Platform({ name: req.body.name });
  platform = await platform.save();

  res.send(platform);
});
```

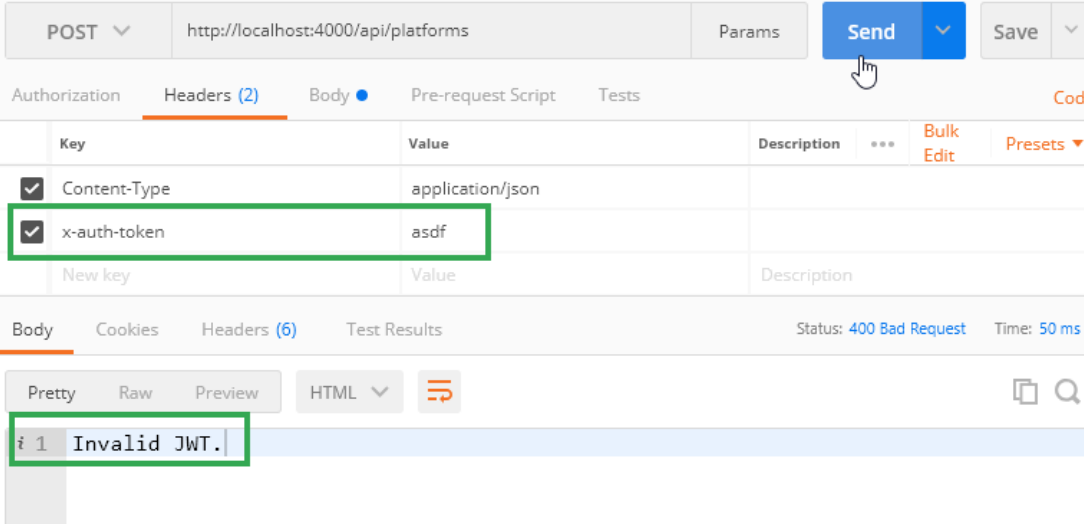
## Test Auth Middleware Using Postman

Looks like everything is in place, let’s send a POST request to http://localhost:4000/api/platforms without providing any token.



So far so good, we are getting the error of “Access denied. No JWT provided.”

Now, we can send a JSON Web Token, but it is a bogus token. Just something silly we make up.



Now we want to make a POST request using a valid JWT. So first we need to find a JWT to use. We can do this by inspecting the response headers generated by the



[Laravel Blade Templating](#)



[Node.js Routes Tutorial](#)



[What Are PHP Arrays?](#)



[Node Package Manager Tutorial](#)



[How To Display API Data Using React.js](#)



[Express.js Beginner Tutorial](#)



[How Do Functions Work in Python?](#)



[Command Types In Linux](#)



[Laravel Migration Generator](#)



[Create A React Element From Scratch](#)



[The Art of Creating a WordPress Post](#)



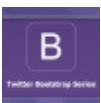
[New String Methods In ES6](#)



[The Declarative Nature of SQL](#)



[Install MongoDB With Compass On Windows](#)



[Twitter Bootstrap Classes](#)



[Install Laravel Homestead on Windows](#)



[PHP Variables and Strings Tutorial](#)



[Build A Regular Expression Tester With Laravel](#)



[The 27 Most Popular File](#)



http://localhost:4000/api/users/ endpoint. Now check this out. Before we use this in our test, let's see how to decode the JWT. You can visit <https://jwt.io/> and paste in the token in question. Then, you can also paste in your own private key. In our case, we simply called it `PrivateKey`. With this, we can verify the token and also see anything embedded within the token.

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiIyYjNmOGRGMGVNTczODExMTg5Nzg1ZmUuIlCjpYXQ0eE1MZA2MjkzMjkuYTlIo8BsohcqaGZRlcIUUIQ-H0EI3e1q9Dq02qRSig2o
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  
  "_id": "5b3b8c70eb573812189785fe",  
  "lat": 1538629232  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
base64UrlEncode(header) + "." +  
base64UrlEncode(payload),  


PrivateKey

  
) ☒ secret base64 encoded
```

☒ Signature Verified

SHARE JWT

This shows us that the user id of `5b3b8c70eb573812189785fe` is embedded in the token. Now who in the database has that user id? That's right! Mr Momo!

```
_id: ObjectId("5b3b8c70eb573812189785fe")  
name: "Mr Momo"  
email: "Mosely@outlook.com"  
password: "$2b$10$nyp8ve1akkf51sKKP4Ooue.T0CmaCwLjToej71Ag7STiKMjiRh040"  
_v: 0
```

That means Mr Momo is a valid user, and he should be able to make a POST request now with no problem. We like Mr Momo.

Ok for this last test, we need to do a couple of things. First off, we need to include the JSON Web Token in the POST request we send to the api. How to do that? Simple, just populate the `x-auth-token` field in Postman.

POST

http://localhost:4000/api/platforms

Params

Send

Save

Authorization

Headers (2)

Body

Pre-request Script

Tests

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...	
New key	Value	Description

Now, since we are posting to `http://localhost:4000/api/platforms`, we recall that this is going to create a new Platform on the server. We will create the “Android” platform like so.

POST ▼ http://localhost:4000/api/platforms Params Send ▼ Save ▼

Authorization Headers (2) **Body** ● Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ▼

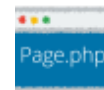
```

1 {
2   "name": "Android"
3 }
  
```

## Functions in PHP




## How To Delete An Item From An Array In React



# How To Create Custom Templates For WordPress Pages



# The Top 15 Most Popular JavaScript String Functions




## How To Filter Models By Another

### Model



 Top 12 Websites  
for Twitter

## Bootstrap



# The Ultimate PHP String Functions

## List



## Iterators In ES6




## CSS Selectors

### Tutorial




## Laravel Cache

### Tutorial

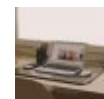


# How To Add Routes and Models To Node Rest API



# Getting Your Database Set Up

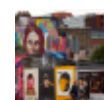
## For Use With Laravel



## PHP Code Structure



## How To Favorite A Model



## How Do Linux Permissions Work?



## Render HTML In Node.js

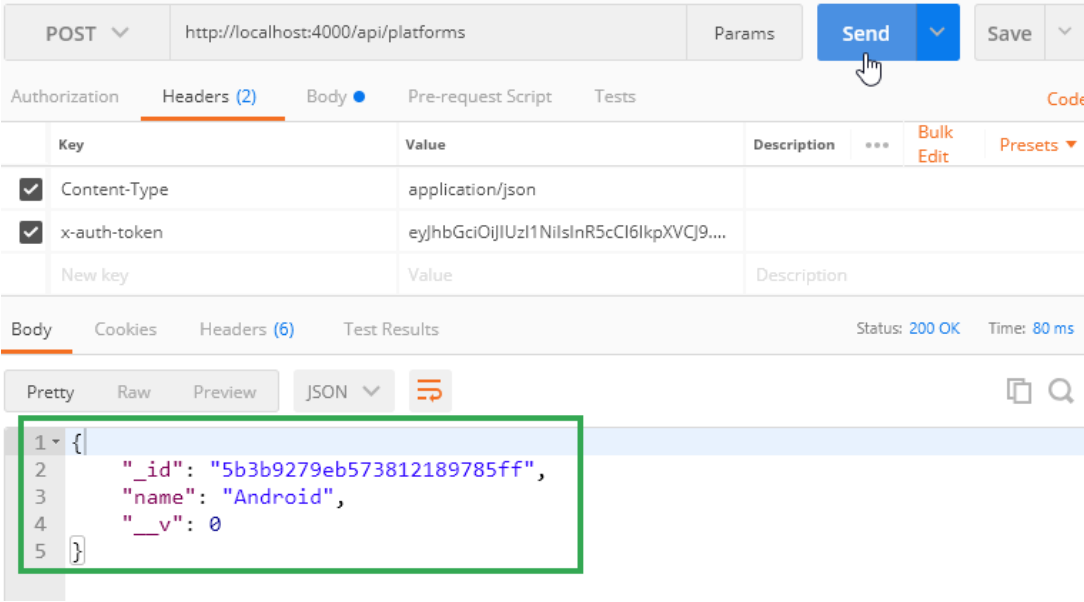


## Vue.js Tutorial

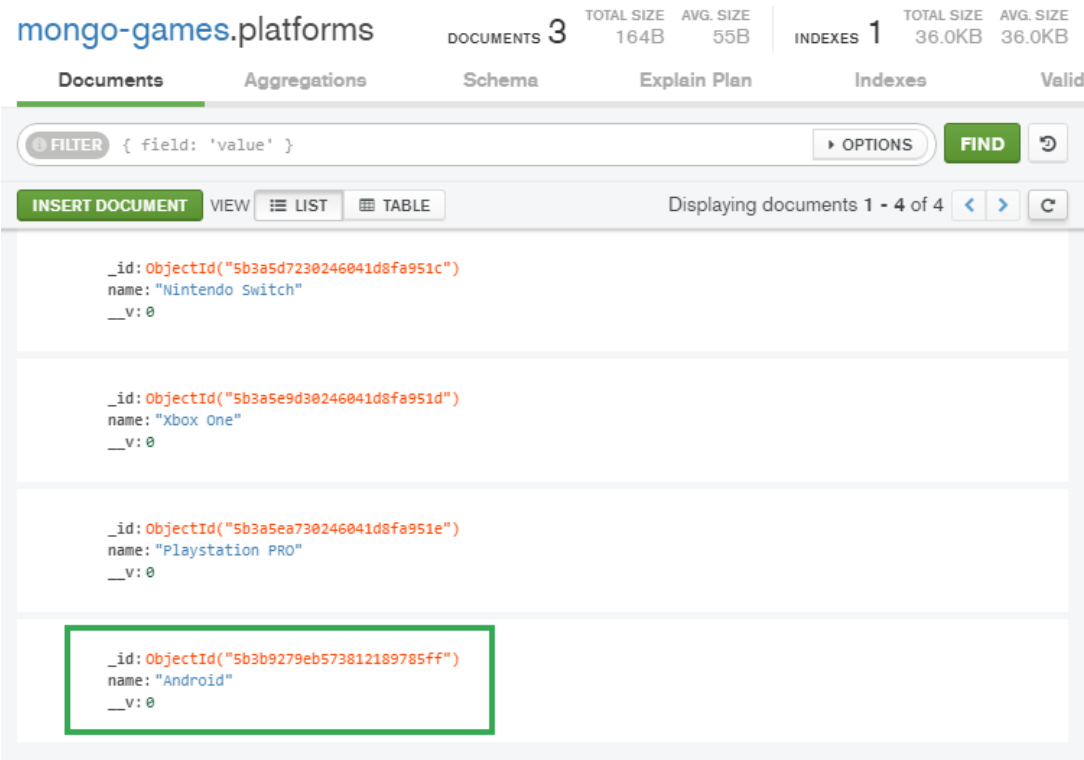


## D3 DOM Selection With D3 JavaScript

Now for the moment of truth! Send that POST request using both the JWT and the JSON payload to create a new platform.



We got a response back with the JSON object specifying “Android”. What does this mean? It means it worked! Don’t believe me? Let’s also look in [Compass](#).



## Apply auth Middleware as you see fit

Here is the great thing. We got this middleware working, and applied it to one specific route or endpoint.




Recall it was as easy as using a require to pull in the middleware, like so:

```
1 const auth = require('../middleware/auth');
```

Then, we can use it as the second argument to the router.post() function like so.

```
1 router.post('/', auth, async (req, res) => {})
```

This means you can now use it anywhere you like. For example, no user should be able to delete a platform without being authenticated. Want to add protection to that route? Simple.

- [Developing With VueJS & PHP](#)
- [Vue JS Directives](#)
- [JavaScript Types and Objects Tutorial](#)
- [Vue-cli Webpack-Simple Tutorial](#)

```
1 router.delete('/:id', auth, async (req, res) => {  
2   const platform = await Platform.findByIdAndRemove(re  
3  
4   if (!platform) {  
5     return res.status(404).send('That platform ID was n  
6   }  
7  
8   res.send(platform);  
9 });
```

## JSON Web Token Authentication With Node.js Summary

With that, we can see how it is pretty straight forward to implement a middleware to protect various routes by making use of JSON Web Tokens. If a user then tries to make a request without sending a token, the api should respond with a 401 status code indicating unauthorized. If the user does provide a valid token, but does not have the correct privileges, the api can respond with a 403 status code indicating forbidden. Also, do not store the JWT in plain text in the database. It is better for the client to store the JWT. In this tutorial, the JWT was simply stored on the Postman application for testing. Postman itself was acting as the client. In a real application, there would be some type of client built in whatever framework chosen, and the JWT could be stored there.

[#javascript](#)[#nodejs](#)

← [How To Add Routes and Models To Node Rest API](#) | [Testing JavaScript With Jest](#) →

[About](#) [Privacy](#) [Terms](#) [User Site Map](#) © 2013 - 2019 Vegibit.com