Developer Circles Kano from facebook

Sign in

Get started

Top highlight

Basics of Authentication using Passport and JWT with Sequelize and MySQL Database

Mustapha Issa Toyin

Follow

Jan 29, 2019 · 7 min read

Hello there, welcome to my first medium post.

It so happened that I was looking for resources to guide me as a beginner

MySQL database with Sequelize. So after surfing the internet for a while, I was able to come across some useful resources like this and a bunch of others, I'm still new to back-end, so it was a bit stressful to combine all the pieces together.

I thought I should put together all these pieces, to help someone who is also struggling with it or would soon start to look for it just like I was.

Before we dive in, let's explain what each of these technologies are.

Passport is authentication middleware for Node.js. extremely flexible and modular.

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as

Disclaimer: This post is introduction and is focused at someone that just want to see how these technologies work together.

We are going to create a simple project, very simple, just to see how to use the mentioned technologies together.

Let's create our project directory

```
mkdir express-jwt-sequelize-mysql
```

That's a funny name right?

Then

```
cd express-jwt-sequelize-mysql
```

Let's get our package.json with default values,

Let's set up our express server, to do that, we need to install some

Responses (8)

What are your thoughts?

Boris Tuman
over 1 year ago

It looks like you are simply storing your passwords as plaintext. As this is a tutorial related to security, I would strongly that you at least mention that this is not good practice.
Passwords should be hashed at a minimum.

9

Reply Hide replies

Mustapha Issa Toyin
AUTHOR
about 1 year ago

Yeah, thanks Boris Tuman, but the article was focused on introducing readers to barest minimum setup required as It was also recommended to the readers not to use plain text for password here https://medium.com/devc-kano/basics-of-authentication-using-passport-and-jwt-with-sequelize-and-mysql-database-748e09d01bab#ad90

Robin Ferrari
about 1 year ago

Nice article, thank you!
You just missing one step: after installing passport-jsonwebtoken, also need to require jsonwebtoken.
We need to add this line:
const jwt = require("jsonwebtoken");

1

Reply 1 reply

Jarupong Pajakgo
almost 2 years ago

8

LeoGouveia I personally suggest you use const/let instead of var because var has trouble about Lexical scope. So you MUST use const/let instead.
var is old. You can take a short time to learn const/let

LeoGouveia
almost 2 years ago

1

1

Why didn't you use var instead of const/let? I see that sometimes you use destructuring so there's already es6 in there...
But I liked your article very much. Stateless authentication is something that continuous bothering me...

Franco Zenatti
almost 2 years ago

since I am a beginner, I wanted to know if it is possible to continue the article with the front end part with vue.js

Reply 1 reply

Franco Zenatti
almost 2 years ago

also is it possible to know if there is a repository from which to download the code?

Reply 1 reply

Elumalai G
almost 2 years ago

1

Hi Mustapha Issa Toyin,
Awesome startup of basic authentication using a passport and jwt with sequalize and mysql.

Isaacmichael

about 1 year ago

req.body.name or req.body.password is undefined

1 reply

Reply

dependencies,

```
npm install --save express body-parser
```

By now, package.json should look like this:

Let’s create a file, index.js, with the following content to get the app running and run a simple page.

```
// index.js

const express = require('express');
const bodyParser = require('body-parser');

const app = express();

// parse application/json
app.use(bodyParser.json());
//parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// add a basic route
app.get('/', function(req, res) {
  res.json({ message: 'Express is up!' });
});

// start the app
app.listen(3000, function() {
  console.log(‘Express is running on port 3000’);
});
```

To start the app, we run the following on the command prompt,

```
node index.js
```

Actually, for development, I would recommend we use nodemon, this would always restart our app as soon as we make any change(s) to the file(s), so let’s install it and save it as a dev dependency.

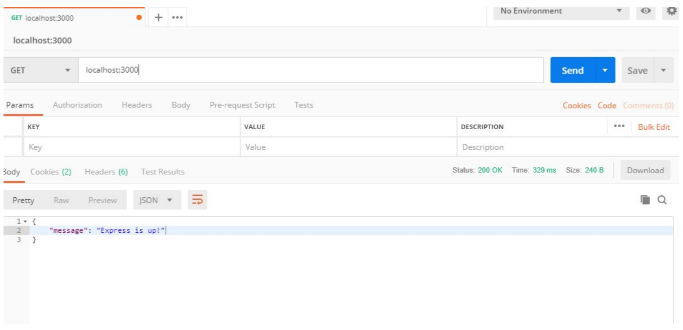
```
npm install nodemon --save-dev
```

After then, we can run the app like so:

```
nodemon index.js
```

We can test our app using Postman by sending a GET request to http://localhost:3000. We also do this using our browser.

The result should look like so:



Status: 200 OK Time: 329 ms Size: 240 B

```
{
    "message": "Express is up!"
}
```

After then, we can now create our database and connect it to the app using Sequelize ORM.

Firstly, create a database, let’s call it users_db.

Then we can go ahead and install the some dependencies to facilitate connection to the database,

```
npm install --save sequelize mysql2
```

And then connect to the database

```
// parse application/json

app.use(bodyParser.json());

//parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: true }));

const Sequelize = require('sequelize');

// initialize an instance of Sequelize

const sequelize = new Sequelize({
  database: 'users_db',
  username: 'root',
  password: '',
  dialect: 'mysql',
});

// check the databse connection

sequelize
  .authenticate()
  .then(() => console.log('Connection has been established
successfully.'))
  .catch(err => console.error('Unable to connect to the
database:', err));
```

If everything went well, when you check your console, you should see a message like “Connection has been established successfully.”

We then need to create a simple model for user, let’s modify our index.js file by adding the following after the code for database connection.

```
// create user model

const User = sequelize.define('user', {
  name: {
    type: Sequelize.STRING,
  },
  password: {
    type: Sequelize.STRING,
  },
});

// create table with user model

User.sync()
  .then(() => console.log('Oh yeah! User table created
successfully'))
  .catch(err => console.log('BTW, did you enter wrong database
credentials?'));
```

By now, if the app is still running, we’ll see the message “Oh yeah! User table created successfully” on the console.

Now, let’s create some helper functions to help us retrieve/submit data to/from the database.

```
// create some helper functions to work on the database

const createUser = async ({ name, password }) => {
  return await User.create({ name, password });
};

const getAllUsers = async () => {
  return await User.findAll();
};

const getUser = async obj => {
  return await User.findOne({
  where: obj,
});
};
```

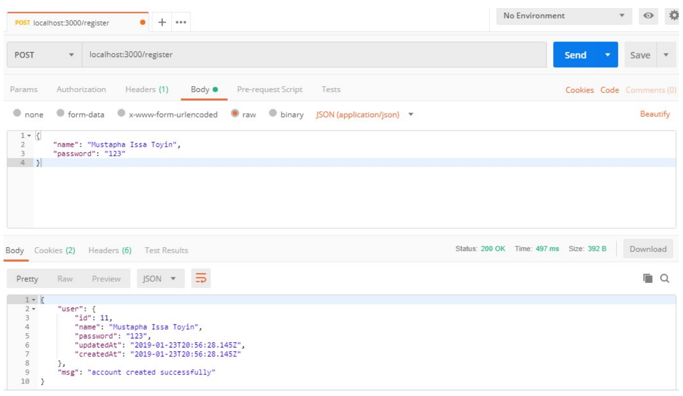
At this point, we can add some more routes, so we can be able to add users to our table and also be able to fetch users list. Let’s add the routes first, and then test them with Postman.

```
// get all users
```

```
app.get('/users', function(req, res) {
  getAllUsers().then(user => res.json(user));
});

// register route
app.post('/register', function(req, res, next) {
  const { name, password } = req.body;
  createUser({ name, password }).then(user =>
    res.json({ user, msg: 'account created successfully' })
  );
});
```

Then we can test the register route, I'll advise you encrypt your password before saving it to the database, you can do this using bcrypt or any other similar tools. That wasn't shown here because the post was meant to be an introductory one but you can check that on your own. So, sending POST request to localhost:3000/register with body content of type raw JSON as shown below, notice the message indicating that the account has been created successfully.



localhost:3000/register

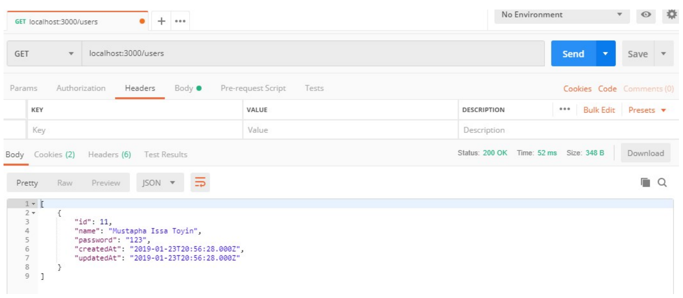
raw: JSON (application/json)

```
{
    "name": "Mustapha Issa Toyin",
    "password": "123"
}
```

Status: 200 OK Time: 497 ms Size: 392 B

```
{
    "user": {
        "id": 11,
        "name": "Mustapha Issa Toyin",
        "password": "123",
        "updatedAt": "2019-01-23T20:56:28.145Z",
        "createdAt": "2019-01-23T20:56:28.145Z"
    },
    "msg": "account created successfully"
}
```

And also GET request using users route like shown below, we can see that the user's details are returned.



localhost:3000/users

Status: 200 OK Time: 52 ms Size: 348 B

```
[
    {
        "id": 11,
        "name": "Mustapha Issa Toyin",
        "password": "123",
        "createdAt": "2019-01-23T20:56:28.000Z",
        "updatedAt": "2019-01-23T20:56:28.000Z"
    }
]
```

. . .

Authentication using Passport.js and JSON Web Tokens

Now, let's setup our authentication using passport, we start by installing the necessary dependencies,

```
npm install passport jsonwebtoken passport-jwt --save
```

Then we can require the installed modules, let's modify the top of index.js like so:

```
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');

// import passport and passport-jwt modules
const passport = require('passport');
const passportJWT = require('passport-jwt');

// ExtractJwt to help extract the token
let ExtractJwt = passportJWT.ExtractJwt;

// JwtStrategy which is the strategy for the authentication
let JwtStrategy = passportJWT.Strategy;
let jwtOptions = {};

jwtOptions.jwtFromRequest =
ExtractJwt.fromAuthHeaderAsBearerToken();
```

```
jwtOptions.secretOrKey = 'wowwow';
```

passport.js works with the concept of strategies. They basically are a middleware function that a requests runs through before getting to the actual route. If your defined authentication strategy fails, which means that the callback will be called with an error that is not null or false as the second argument, the route will not be called, but an error 401 unauthorized response will be sent.

Let's implement the jwt strategy by adding this below the above code,

```
// lets create our strategy for web token
let strategy = new JwtStrategy(jwtOptions, function(jwt_payload,
next) {
  console.log('payload received', jwt_payload);
  let user = getUser({ id: jwt_payload.id });
  if (user) {
    next(null, user);
  } else {
    next(null, false);
  }
});
// use the strategy
passport.use(strategy);
```

Then we initialize passport by placing this below where we created our app,

```
app.use(passport.initialize());
```

Now, let's create our login route using the helper function we created above,

```
// login route

app.post('/login', async function(req, res, next) {
  const { name, password } = req.body;
  if (name && password) {
    // we get the user with the name and save the resolved
promise
    returned
    let user = await getUser({ name });
    if (!user) {
      res.status(401).json({ msg: 'No such user found', user });
    }
   if (user.password === password) {
      // from now on we'll identify the user by the id and the id
is

// the only personalized value that goes into our token
      let payload = { id: user.id };
      let token = jwt.sign(payload, jwtOptions.secretOrKey);
      res.json({ msg: 'ok', token: token });
    } else {
      res.status(401).json({ msg: 'Password is incorrect' });
    }
  }
});
```

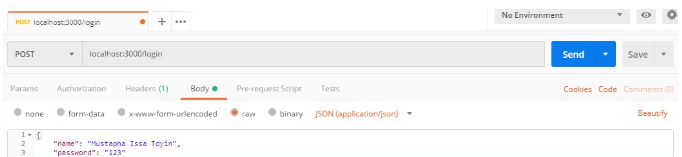
Now, let's test our login route with a user that is not registered, setting the http method to POST and type to raw JSON (application/json) we should get a response like so,

```
{
  "msg": "No such user found",
  "user": null
}
```

And if we try with wrong password, we get this response,

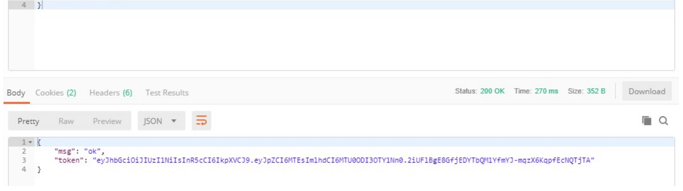
```
{
  "msg": "Password is incorrect"
}
```

Whereas the correct credentials would give us the following result,



POST localhost:3000/login

```
{
    "name": "Mustapha Issa Toyin",
    "password": "123"
```



```
}
```

Body | Cookies (2) | Headers (6) | Test Results — Status: 200 OK Time: 270 ms Size: 352 B

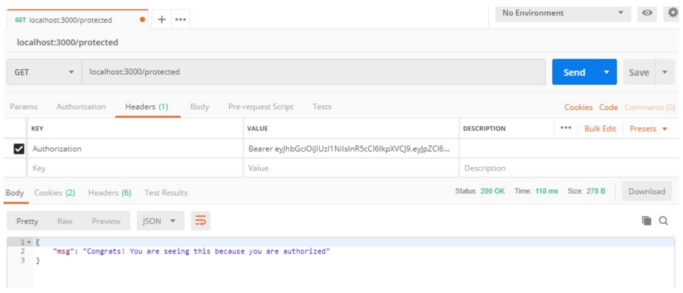
```
{
    "msg": "ok",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTEsImlhdCI6MTU0ODI3OTY1Nn0.2iUFlBgE8GfjEDYTbQM1YfmYJ-mqzX6KqpfEcNQTjTA"
}
```

The token in the response object can now be used for authorization, let's say any request sent to a protected route cannot be allowed unless it has the token attached to it. We would demonstrate this next. First, let's create a protected route,

```
// protected route
app.get('/protected', passport.authenticate('jwt', { session:
false }), function(req, res) {
res.json({ msg: 'Congrats! You are seeing this because you are
authorized'});


});
```

Then, we can test it by trying to access the protected route, we would receive a response of status 401 Unauthorized. As earlier mentioned, the token needs to be attached to the request, we can do this adding key Authorization to the headers with value "Bearer " appended to the token we want to attach to the request. You can check the image below for clarity,



GET localhost:3000/protected

	KEY	VALUE	DESCRIPTION
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6...	
	Key	Value	Description

Status: 200 OK Time: 118 ms Size: 278 B

```
{
    "msg": "Congrats! You are seeing this because you are authorized"
}
```

You can find the full code below. Was this helpful? If it was, please leave your review. You can also check the full project here.

I also built a boilerplate repository on top of this article which is enough to help you kick-start your project in the shortest time, you can check it out here https://github.com/emaitee/node-express-boilerplate. It contains additional features such as password encryption, user access level control, ES6 syntax etc.

Nodejs Passport

1.1K claps

8 responses

WRITTEN BY

Mustapha Issa Toyin

Follow

Building for Server, Web, Mobile using JavaScript

DevC Kano

Follow

Medium Publication for Members of DevC Kano

More From Medium

JavaScript Array Functional Methods like map( ), filter( ) and reduce( )

Ishu Singhal in The Startup

Angular 9 Has Arrived: Know What's New And Changed

JSPanther

Leveraging JS File Uploader Libraries in PHP Forms: Example using FilePond

Jim Dee in Web Designer / Web Developer Magazine

Must-Know Ref vs Reactive Differences in Vue 3 Composition API

Raja Tamil in The Startup

Day 16: Become a self-taught blockchain developer with zero knowledge in 365 days

Ximena Han

These 5 Tips Will Change The Way You Build React Apps

Mackenzie Kieran in dev@red

Vue.JS

Ahmet Suhan Oka

How To Combine And Use Multiple NextJS Plugins

Malcolm L in Frontend Digest