Understanding JWT Authentication With Node.js

Lesson 6 of 9

By Taha Sufiyan

Last updated on Feb 10, 2023

94840

Previous

Next

Tutorial Playlist

Table of Contents

What is JSON Web Token?

Advantages of Node.js authentication with JWT

View More

Thousands of developers around the world use Node.js to develop I/O-intensive web applications, such as video streaming sites, single-page applications, online chat applications, and other web apps. The open-source (and completely free) platform offers several advantages over other server-side platforms, like Java or PHP.

This tutorial on Node.js authentication with JWT will help you learn how to add a security layer when accessing different routes within a Node.js web application. First, we will discuss the basics of JWT (JSON Web Token) and then cover its implementation within a Node.js application.

What is JSON Web Token?

JSON Web Token (JWT) is a standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. The compact size makes the tokens easy to transfer through an URL, POST parameter, or inside an HTTP header. The information in a JWT is digitally signed using a secret or public/private key pair.

JWTs can be signed using a secret or a public/private key pair.

JWTs are mainly used for authentication. After a user signs in to an application, the application then assigns JWT to that user. Subsequent requests by the user will include the assigned JWT. This token tells the server what routes, services, and resources the user is allowed to access.

Advantages of Node.js authentication with JWT

Node.js authentication with JWT has several advantages over the traditional authentication process, primarily the scalability of stateless applications. And since it's becoming popular among such heavyweights as Facebook and Google, it's adoption across the industry likely will continue to grow.

Other advantages include:

- Provides more trustworthiness than cookies or sessions

The Need for JSON Web Token

There are several reasons why applications use JSON Web Tokens for authentication:

	• JWT is an excellent choice to be passed in HTML and HTTP environments due to its smaller footprint when compared to other types of tokens
	• JSON Web Tokens can be signed using a shared secret and also by using public/private key pairs
	• It is easier to work with JWT as JSON parsers are standard in most programming languages
	• JWT is also suitable for implementing authorization in large-scale web applications

Structure of a JWT

A JSON Web Token consists of:

- **Header** – Consists of two parts: the type of token (i.e., JWT) and the signing algorithm (i.e., HS512)

- **Payload** – Contains the claims that provide information about a user who has been authenticated along with other information such as token expiration time

- **Signature** – Final part of a token that wraps in the encoded header and payload, along with the algorithm and a secret

Learn to build network applications quickly and efficiently using JavaScript with the Node.js Training. Click to enroll now!

JWT Use Cases

Some scenarios where JSON Web Tokens are useful:

	• **Authorization** - This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.
	• **Information Exchange** - JSON Web Tokens are a good way of securely transmitting information between parties.

Next, we'll look at some of the node applications using JWT.

Node.js Application with JWT

The following application uses JWT authentication allowing users to access routes by logging in.

Prerequisites

Node.js installation

1. Download the Node.js. Select the installer according to your operating system and

1. Download the Node.js. Select the installer according to your operating system and environment

2. Run the Node.js installer. Accept the license agreement. You can leave other settings as default. The installer will install Node.js and prompt you to click on the finish button.

3. Verify that Node.js was properly installed by opening the command prompt and typing this command: **node --version**

4. When we install Node.js, NPM (Node Package Manager) is also installed. NPM includes many libraries that are used in web applications, such as React. Verify whether it is installed with the following command in CMD: **npm --version**

Text editor

Install a text editor of your choice. We are using Visual Studio Code in this tutorial, but you can use other editors, such as Atom or Sublime Text, if you are more comfortable with those.

Postman

We are using the Postman application to check the output of the application. We send and receive API calls and check if the JWT is working correctly.

To download the Postman application, go to its official website.

Project Setup

1. Create an empty folder and name it **mongodb_crud**.

2. Open the newly created directory in VS Code and (inside the terminal) type **npm init** to initialize the project. Press the Enter key to leave the default settings as they are.

Let’s Code!

We are going to create our own Node.js application and include JWT, thus adding a security mechanism to each route that we define in this application. This prevents unauthorized access and allows the logged in users to access the routes as long as they have the JWT token included in the header request.

Let’s first take a look at how the project directory should be at the end of this tutorial. We are working only on a single file called **index.js**, so the project directory is super simple.

index.js

Create a file called **index.js** in the project directory. This is the only file that we create and work on in this project in order to keep things simple. Our main goal is to understand how JWTs work in a Node.js application.

So, let’s go ahead and understand what each snippet of code does in this file.

- We are importing a couple of modules that we will be needing in the application: **express**, and **jwt**.

```
const express = require("express");

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

- We can add these modules using the terminal inside VSCode.
- After that, we create an express application variable.

```
const app = express();
```

- We then define an **app.get()** method to create a json string with the desired message.

```
app.get("/api", (req, res) => {
```

```
app.get("/api", (req, res) => {

  res.json({

    message: "Hey, there! Welcome to this API service"

  });

});
```

- app.post() is a POST request and in the method parameters, we add URL, verifyToken, and request and response values.

verify() method then takes the request token as input and verifies whether it is correct. We set it to print an error message if it doesn’t match; otherwise, we print a message on the screen stating that the post was created.

```
app.post("/api/posts", verifyToken, (req, res) => {

  jwt.verify(req.token, "secretkey", (err, authData) => {

    if (err) {

      res.sendStatus(403);

    } else {

      res.json({

        message: "POST created...",

        authData

      });

    }

  });

});
```

- We define another POST method that creates a route for user login at the specified URL.

JWT then uses the **sign()** method to create a JSON Web Token for that user and returns the token in the form of a JSON string.

```
app.post("/api/login", (req, res) => {
  const user = {
    id: 1,
    username: "john",
    email: "john@gmail.com"
  };
  jwt.sign({ user: user }, "secretkey", (err, token) => {
    res.json({
      token
    });
  });
});
```

- Finally, we define the method **verifyToken()** that takes care of the token verification process.
- bearerHeader variable contains the token that is passed in the authorization field of request header.
- We add an if condition that checks whether the token exists in the authorization field; if not, we send an error status to the user.

```
function verifyToken(req, res, next) {
  const bearerHeader = req.headers["authorization"];
  if (typeof bearerHeader !== "undefined") {
```

```
  if (typeof bearerHeader !== "undefined") {

    const bearerToken = bearerHeader.split(" ")[1];

    req.token = bearerToken;

    next();

  } else {

    res.sendStatus(403);

  }

}
```

- In the end, we set the server to listen to Port 3000.

```
app.listen(3000, () => console.log("Server started"));
```

That’s all the code we are going to write to develop this application. Next, we’ll run the application by using the command **node index.js** in the terminal of the VS Code.

The terminal should look like this after using the above command:

If your terminal also displays Server started, it means that the server is running properly.

If you're eager to gain the skills required to work in a challenging, rewarding, and dynamic IT role - we've got your back! Discover the endless opportunities through this innovative Post Graduate Program in Full Stack Web Development course designed by our partners at Caltech CTME. Enroll today!

Get Ahead of the Curve and Master Node.js Today

Now that you have learned about the value of Node.js authentication with JWT, you may be wondering how you can obtain the skills necessary to take advantage of its rising popularity. Fortunately, there are some great options to learn these skills at your own pace. Simplilearn’s Certification training course will give you a great, foundational understanding of this popular platform, combining live, instructor-led training, self-paced tutorials, and hands-on projects to help you become career-ready upon completion. Node.js authentication is one of the commonly asked topics in the Node.js interview questions. So, get started today and seize your future!

Find our Post Graduate Program in Full Stack Web Development Online Bootcamp in top cities:

Name	Date	Place
Post Graduate Program in Full Stack Web Development	Cohort starts on 1st Mar 2023, Weekend batch	Your City
Post Graduate Program in Full Stack Web Development, Hyderabad	Cohort starts on 2nd Mar 2023, Weekend batch	Hyderabad
Post Graduate Program in Full Stack Web Development, Pune	Cohort starts on 15th Mar 2023, Weekend batch	Pune

About the Author

Taha Sufiyan

Taha is a Research Analyst at Simplilearn. He is passionate about building great user interfaces and keeps himself updated on the world of Artificial Intelligence. Taha is also interested in ga…

View More

Recommended Resources

An Ultimate Guide to Learn the Importance o…

Getting Started With Low-Code and No-Cod…

2 Comments

1 Login ▾

G

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Share

Best Newest Oldest

Prudhvi Raj Civil
2 years ago

You just sent the token in json after login in and manually entered that token in Postman. How to send the token to headers.["authorization"] after user login.

0 0 Reply • Share ›

Team Simplilearn Mod → Prudhvi Raj Civil
2 years ago

Thanks for your feedback, Prudhvi. We'll soon come up with an article explaining the topic.

0 0 Reply • Share ›

Subscribe Privacy Do Not Sell My Data