Tutorial 11. Authentication and Authorization

Authentication: check if username and password is correct
Authorization: check if this user has permission to do something

There are some approaches:
- Use Passport libraries
- Use Loopback
- **Use JWT: Json Web Token**
- Build our own: Token-based, Basic

https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/

How to Build an Authentication API with JWT Token in Node.js

June 15, 2021

- Topics:
- Node.js

In this tutorial, we will learn how to use *JWT* in *Node.js* to secure endpoints and even authenticate users.

It’s pretty simple to write code and develop applications. Yet, how do we deal with authentication, and most likely, authorization?

What is authentication and authorization

Authentication and authorization are used in security, particularly when it comes to getting access to a system. Yet, there is a significant distinction between gaining entry into a house (authentication) and what you can do while inside (authorization).

Authentication

Authentication is the process of verifying a user’s identification through the acquisition of credentials and using those credentials to confirm the user’s identity. The authorization process begins if the credentials are legitimate. The authorization process always follows the authentication procedure.

You were already aware of the authentication process because we all do it daily, whether at work (logging into your computer) or at home (logging into a website). Yet, the truth is that most “things” connected to the Internet require you to prove your identity by providing credentials.

Authorization

Authorization is the process of allowing authenticated users access to resources by determining whether they have system access permissions. By giving or denying specific licenses to an authenticated user, authorization enables you to control access privileges.

So, authorization occurs after the system authenticates your identity, granting you complete access to resources such as information, files, databases, funds, places, and anything else. That said, authorization affects your capacity to access the system and the extent to which you can do so.

What is JWT

JSON Web Tokens (JWT) are an RFC 7519 open industry standard for representing claims between two parties. For example, you can use jwt.io to decode, verify, and produce JWT.

JWT specifies a compact and self-contained method for communicating information as a JSON object between two parties. Because it is signed, this information can be checked and trusted. JWTs can be signed using a secret (using the HMAC algorithm) or an RSA or ECDSA public/private key combination. In a moment, we'll see some examples of how to use them.

Prerequisites

To follow along with this tutorial, you will need:

- A working knowledge of JavaScript.
- A good understanding of Node.js.
- A basic understanding of MongoDB or any database of your choice.
- Postman and some knowledge on how to use Postman.

API development using JWT token for authentication in Node.js

To get started, we’ll need to set up our project.

Open Visual Studio Code by navigating to a directory of your choice on your machine and opening it on the terminal.

Then execute:

```
code.
```

Note: `code .` won’t work if you don’t have Visual Studio Code installed on your system.

Step 1 - Create a directory and initialize `npm`

Create a directory and initialize `npm` by typing the following command:

- Windows power shell

```
mkdir jwt-project

cd jwt-project

npm init -y
```

- Linux

```
mkdir jwt-project
```

```
cd jwt-project
```

```
npm init -y
```

```
//simple: create package.json
```

Step 2 - Create files and directories

In step 1, we initialized npm with the command `npm init -y`, which automatically created a package.json.

We need to create the `model`, `middleware`, `config` directory and their files, for example `user.js`,`auth.js`,`database.js` using the commands below.

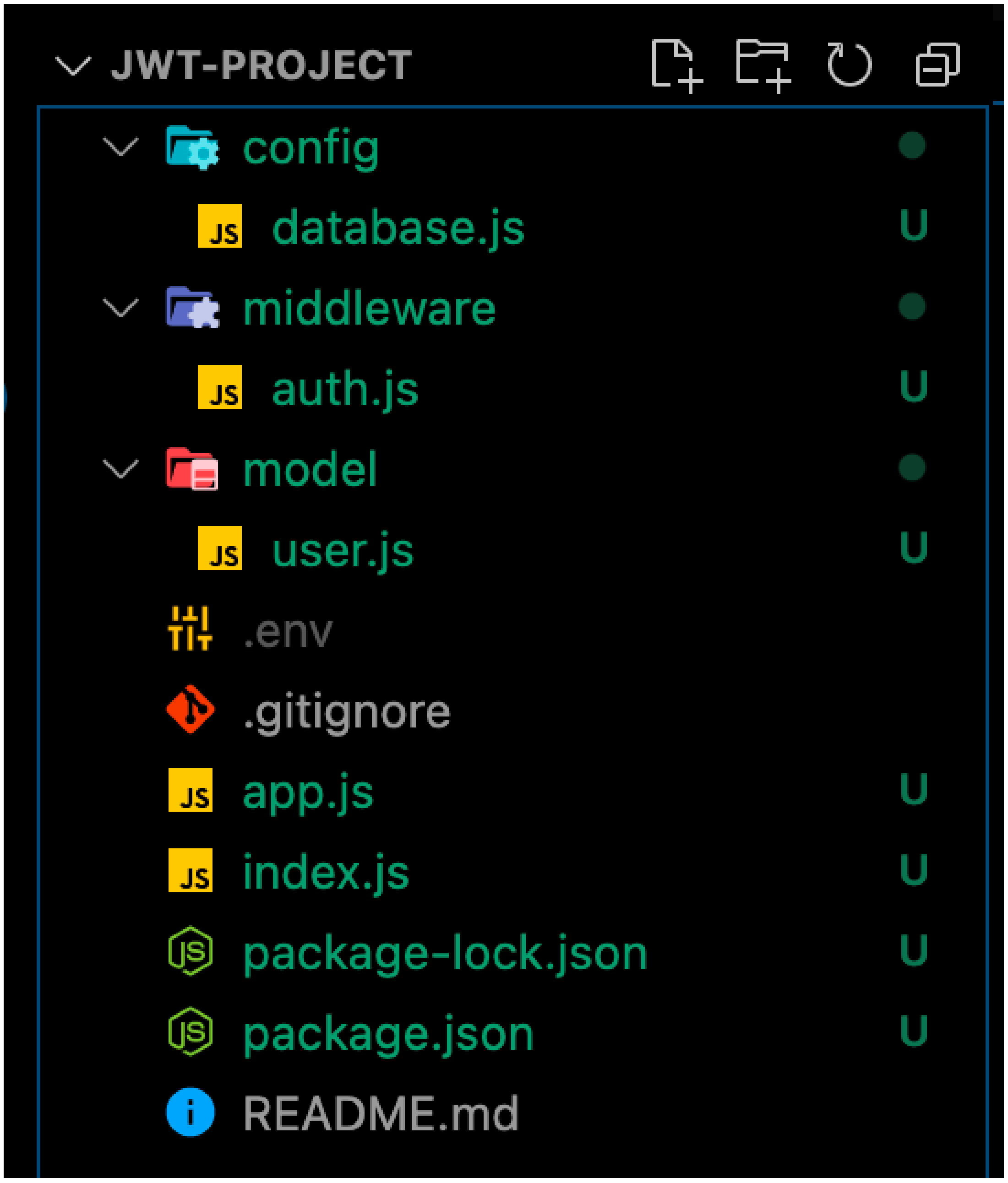
```
mkdir model middleware config
```

```
touch config/database.js middleware/auth.js model/user.js
```

We can now create the `index.js` and `app.js` files in the root directory of our project with the command.

```
touch app.js index.js
```

As shown in the image below:



```
JWT-PROJECT
  config
    database.js        U
  middleware
    auth.js            U
  model
    user.js            U
  .env
  .gitignore
  app.js               U
  index.js             U
  package-lock.json    U
  package.json         U
  README.md
```

Step 3 - Install dependencies

We’ll install several dependencies like `mongoose`, `jsonwebtoken`, `express` `dotenv` `bcryptjs` and development dependency like `nodemon` to restart the server as we make changes automatically.

We will install mongoose because I will be using MongoDB in this tutorial.

We will validate user credentials against what we have in our database. So the whole authentication process is not limited to the database we’ll be using in this article.

```
npm install mongoose express jsonwebtoken dotenv bcryptjs
```

```
npm install nodemon -D
```

Step 4 - Create a Node.js server and connect your database

Now, let’s create our Node.js server and connect our database by adding the following snippets to your `app.js`, `index.js` , `database.js` `.env` in that order.

In our `database.js.`

`config/database.js`:

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

const { MONGO_URI } = process.env;

exports.connect = () => {
  // Connecting to the database
  mongoose
    .connect(MONGO_URI)
[1]    .then(() => {
      console.log("Successfully connected to database");
    })
    .catch((error) => {
      console.log("database connection failed. exiting now...");
      console.error(error);
      process.exit(1);
    });
};
```

In our `app.js`:

`jwt-project/app.js`

```
require("dotenv").config();
require("./config/database").connect();
const express = require("express");
const bcrypt =  require('bcryptjs')

const jwt = require('jsonwebtoken')

const app = express();
```

[1] Please note that this is slightly different from the original tutorials. Please read here: https://stackoverflow.com/questions/68958221/mongoparseerror-options-usecreateindex-usefindandmodify-are-not-supported

```
app.use(express.json());

// Logic goes here

module.exports = app;
```

In our `index.js`:

`jwt-project/index.js`

```
const http = require("http");
const app = require("./app");
const server = http.createServer(app);

const { API_PORT } = process.env;
const port = process.env.PORT || API_PORT;

// server listening
server.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

If you notice, our file needs some environment variables. You can create a new `.env` file if you haven't and add your variables before starting our application.

In our `.env`.

```
API_PORT=4001
```

```
MONGO_URI= //Your database URI here
```

To start our server, edit the scripts object in our `package.json` to look like the one shown below.

```
"scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  }
```

The snippet above has been successfully inserted into `app.js`, `index.js`, and `database.js`. First, we built our node.js server in `index.js` and imported the app.js file with routes configured.

Then, as indicated in `database.js`, we used `mongoose` to create a connection to our database.

Execute the command `npm run dev`.

Both the server and the database should be up and running without crashing.

Step 5 - Create user model and route

We'll define our schema for the user details when signing up for the first time and validate them against the saved credentials when logging in.

Add the following snippet to `user.js` inside the `model` folder.

`model/user.js`

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

const userSchema = new mongoose.Schema({
  first_name: { type: String, default: null },
  last_name: { type: String, default: null },
  email: { type: String, unique: true },
  password: { type: String },
  token: { type: String },
});

module.exports = mongoose.model("user", userSchema);
```

Now let's create the routes for `register` and `login`, respectively.

In `app.js` in the root directory, add the following snippet for the registration and login.

`app.js`

```
// importing user context
const User = require("./model/user");
```

```
// Register
app.post("/register", (req, res) => {
// our register logic goes here...
});

// Login
app.post("/login", (req, res) => {
// our login logic goes here
});
```

Step 6 - Implement register and login functionality

We'll be implementing these two routes in our application. We will be using JWT to sign the credentials and `bycrypt` to encrypt the password before storing them in our database.

From the `/register` route, we will:

- Get user input.
- Validate user input.
- Validate if the user already exists.
- Encrypt the user password.
- Create a user in our database.
- And finally, create a signed JWT token.

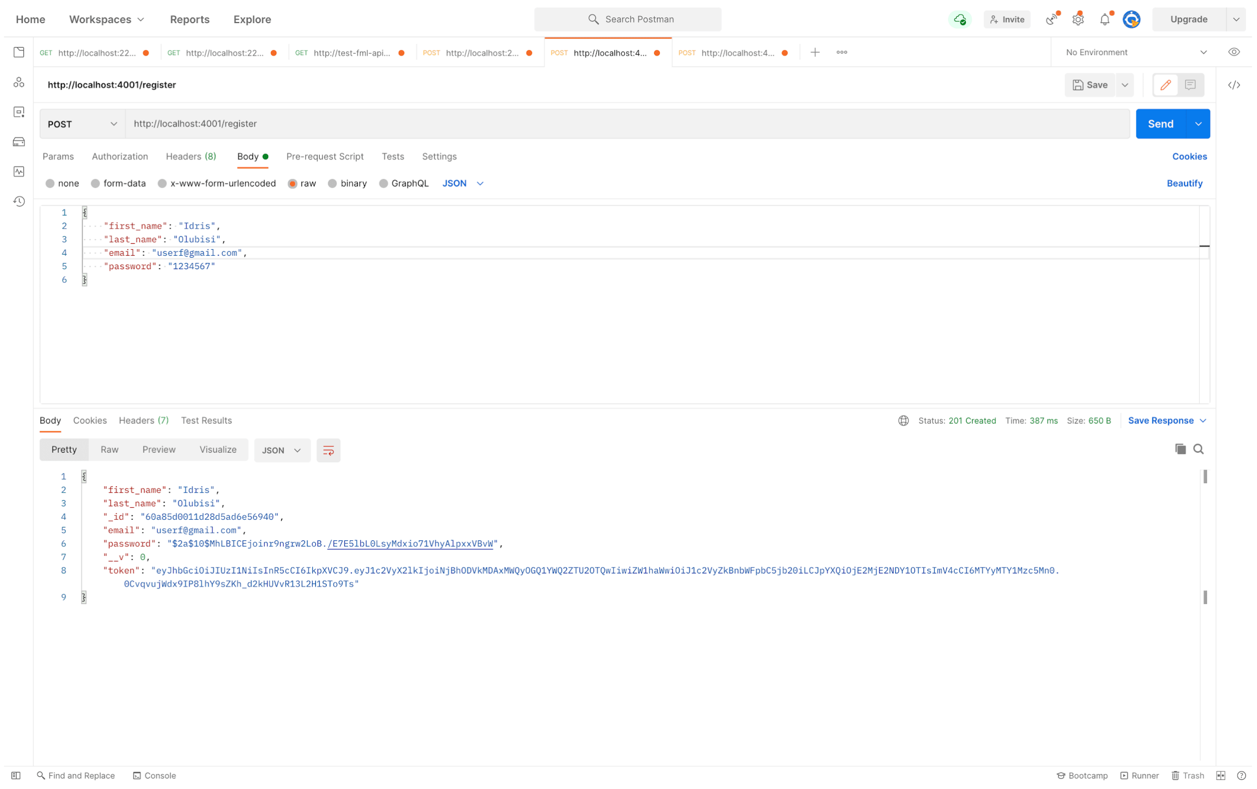
Modify the `/register` route structure we created earlier to look as shown below.

app.js

```
// ...

app.post("/register", async (req, res) => {

  // Our register logic starts here
  try {
    // Get user input
    const { first_name, last_name, email, password } = req.body;

    // Validate user input
    if (!(email && password && first_name && last_name)) {
      res.status(400).send("All input is required");
    }

    // check if user already exist
    // Validate if user exist in our database
    const oldUser = await User.findOne({ email });

    if (oldUser) {
      return res.status(409).send("User Already Exist. Please
Login");
    }

    //Encrypt user password
    encryptedPassword = await bcrypt.hash(password, 10);

    // Create user in our database
    const user = await User.create({
      first_name,
      last_name,
      email: email.toLowerCase(), // sanitize: convert email to
lowercase
      password: encryptedPassword,
    });

    // Create token
    const token = jwt.sign(
      { user_id: user._id, email },
      process.env.TOKEN_KEY,
      {
        expiresIn: "2h",
      }
    );
    // save user token
    user.token = token;

    // return new user
    res.status(201).json(user);
  } catch (err) {
    console.log(err);
  }
  // Our register logic ends here
});

// ...
```

Note: Update your `.env` file with a `TOKEN_KEY`, which can be a random string.

Using Postman to test the endpoint, we'll get the response shown below after successful registration.



POST http://localhost:4001/register

Body (raw, JSON):

```json
{
    "first_name": "Idris",
    "last_name": "Olubisi",
    "email": "userf@gmail.com",
    "password": "1234567"
}
```

Status: 201 Created Time: 387 ms Size: 650 B

```json
{
    "first_name": "Idris",
    "last_name": "Olubisi",
    "_id": "60a85d0011d28d5ad6e56940",
    "email": "userf@gmail.com",
    "password": "$2a$10$MhLBICEjoinr9ngrw2LoB./E7E5lbL0LsyMdxio71VhyAlpxxVBvW",
    "__v": 0,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBhODVkMDAxMWQyOGQ1YWQ2ZTU2OTQwIiwiZW1haWwiOiJ1c2VyZkBnbWFpbC5jb20iLCJpYXQiOjE2MjE2NDY1OTIsImV4cCI6MTYyMTY1Mzc5Mn0.
        0CvqvujWdx9IP8lhY9sZKh_d2kHUVvR13L2H1STo9Ts"
}
```

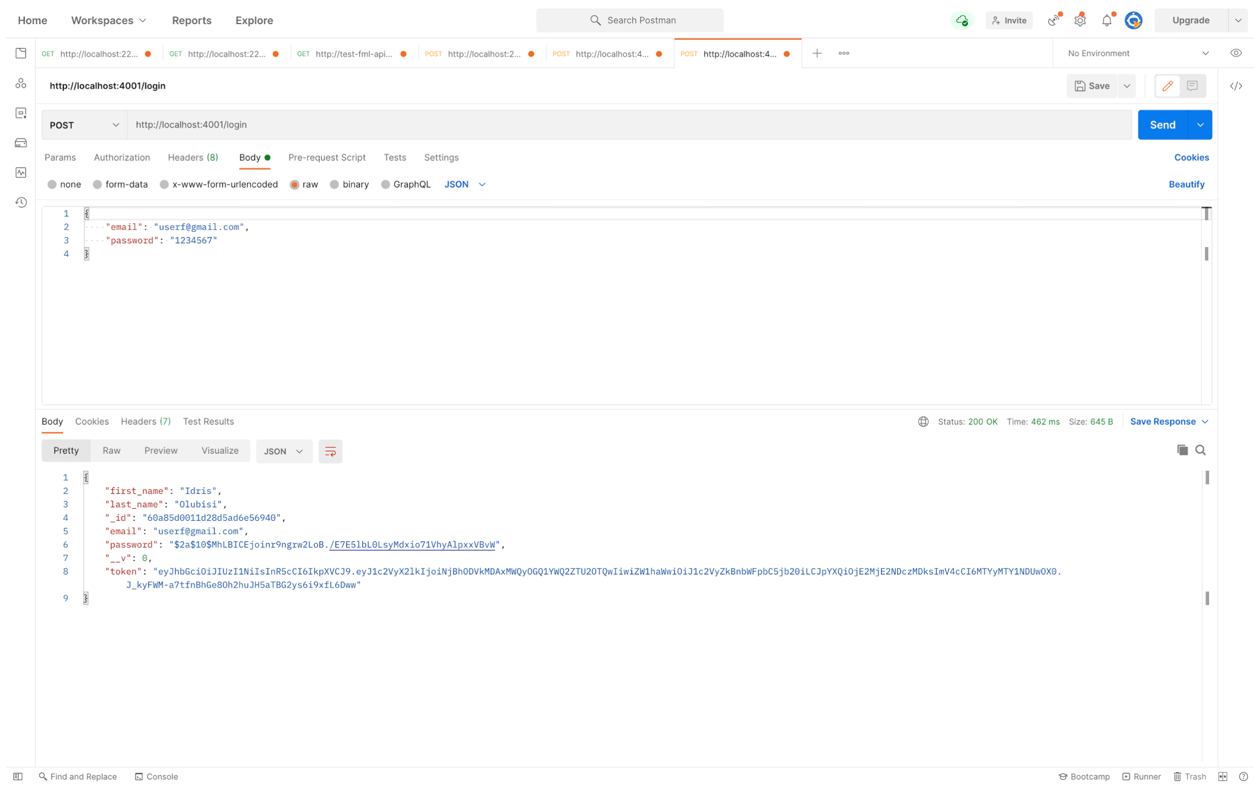
For the `/login` route, we will:

- Get user input.
- Validate user input.
- Validate if the user exists.
- Verify user password against the password we saved earlier in our database.
- And finally, create a signed JWT token.

Modify the `/login` route structure we created earlier to look like shown below.

```
// ...

app.post("/login", async (req, res) => {

  // Our login logic starts here
  try {
    // Get user input
    const { email, password } = req.body;

    // Validate user input
    if (!(email && password)) {
      res.status(400).send("All input is required");
    }
    // Validate if user exist in our database
    const user = await User.findOne({ email });

    if (user && (await bcrypt.compare(password, user.password)))
{
      // Create token
      const token = jwt.sign(
        { user_id: user._id, email },
        process.env.TOKEN_KEY,
        {
          expiresIn: "2h",
        }
      );

      // save user token
      user.token = token;

      // user
      res.status(200).json(user);
    }
    res.status(400).send("Invalid Credentials");
  } catch (err) {
    console.log(err);
  }
  // Our register logic ends here
});

// ...
```

Using Postman to test, we'll get the response shown below after a successful login.



POST http://localhost:4001/login

Body raw JSON

```
{
    "email": "userf@gmail.com",
    "password": "1234567"
}
```

Status: 200 OK Time: 462 ms Size: 645 B

```
{
    "first_name": "Idris",
    "last_name": "Olubisi",
    "_id": "60a85d0011d28d5ad6e56940",
    "email": "userf@gmail.com",
    "password": "$2a$10$MhLBICEjoinr9ngrw2LoB./E7E5lbL0LsyMdxio71VhyAlpxxVBvW",
    "__v": 0,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBhODVkMDAxMWQyOGQ1YWQ2ZTU2OTQwIiwiZW1haWwiOiJ1c2VyZkBnbWFpbC5jb20iLCJpYXQiOjE2MjE2NDczMDksImV4cCI6MTYyMTY1NDUwOX0.
        J_kyFWM-a7tfnBhGe8Oh2huJH5aTBG2ys6i9xfL6Dww"
}
```

Step 7 - Create middleware for authentication

We can successfully create and log in a user. Still, we'll create a route that requires a user token in the header, which is the JWT token we generated earlier.

Add the following snippet inside `auth.js`.

`middleware/auth.js`

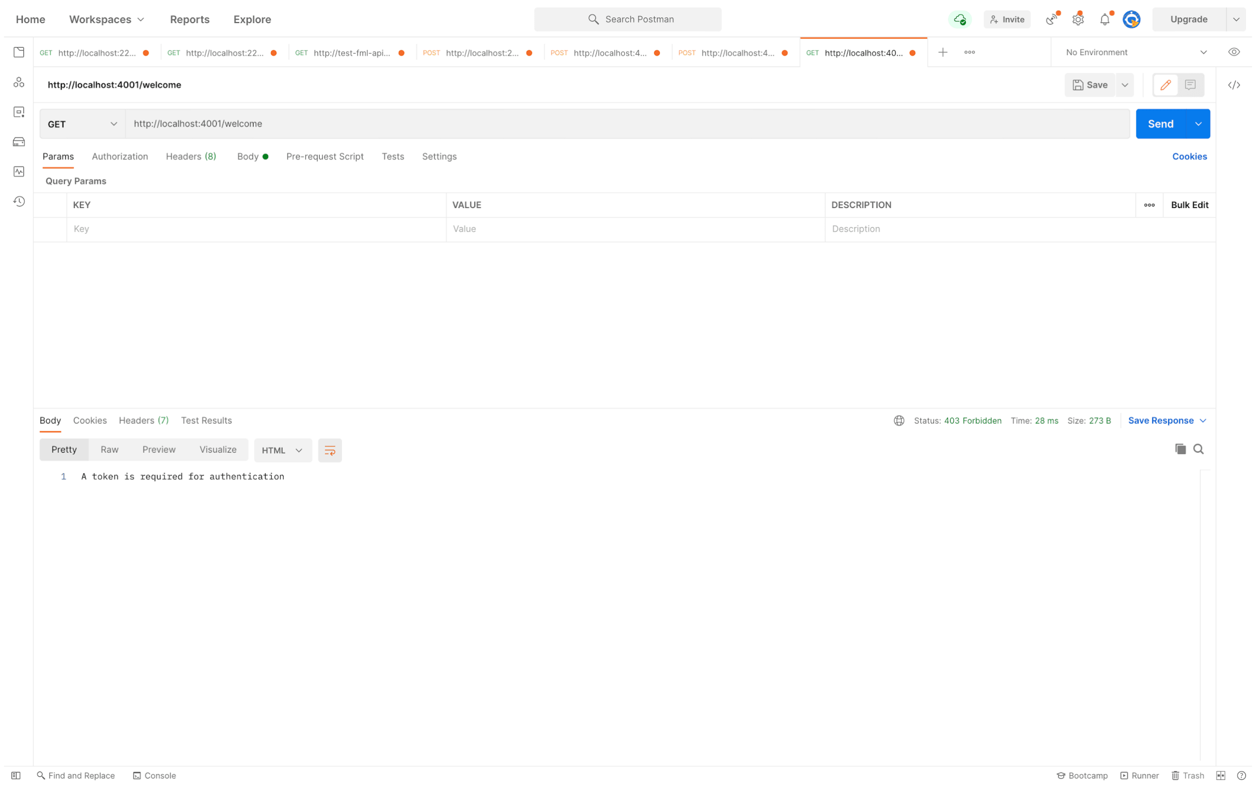
```
const jwt = require("jsonwebtoken");

const config = process.env;

const verifyToken = (req, res, next) => {
  const token =
    req.body.token || req.query.token ||
req.headers["x-access-token"];

  if (!token) {
    return res.status(403).send("A token is required for
authentication");
  }
  try {
    const decoded = jwt.verify(token, config.TOKEN_KEY);
    req.user = decoded;
  } catch (err) {
    return res.status(401).send("Invalid Token");
  }
  return next();
};

module.exports = verifyToken;
```

Now let’s create the `/welcome` route and update `app.js` with the following snippet to test the middleware.

`app.js`

```
const auth = require("./middleware/auth");

app.post("/welcome", auth, (req, res) => {
  res.status(200).send("Welcome 🙌 ");
});
```

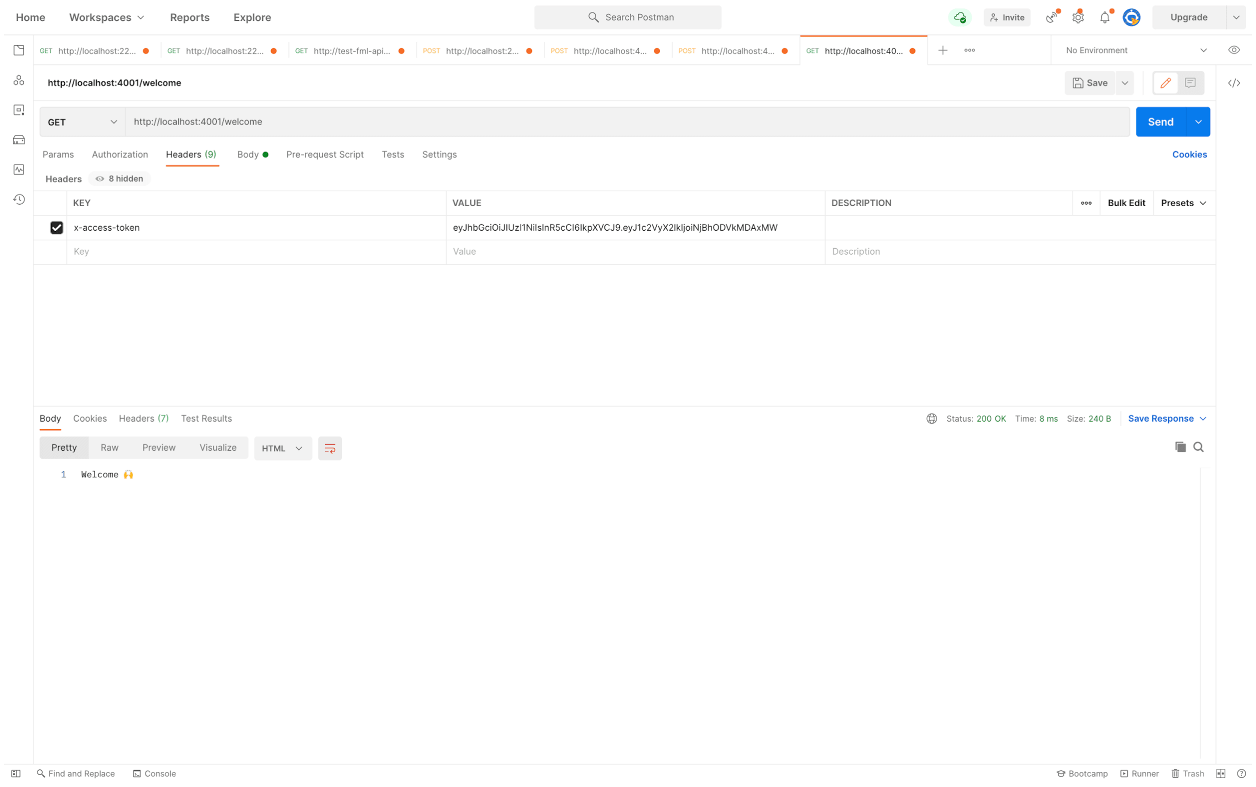
See the result below when we try to access the `/welcome` route we just created without passing a token in the header with the `x-access-token` key.



```
A token is required for authentication
```

We can now add a `token` in the header with the key `x-access-token` and re-test.

See the image below for the response.



GET http://localhost:4001/welcome

	KEY	VALUE	DESCRIPTION
✓	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBhODVkMDAxMW	

Status: 200 OK Time: 8 ms Size: 240 B

```
Welcome 🙌
```

You can [click here](#) to check the complete code on GitHub.

Conclusion

In this tutorial we learned about JWT, authentication, authorization and how to develop an API using JWT token for authentication in Node.js.

Happy coding!

Resources

- JWT
- Node.js
- ExpressJS

Important notes:

https://stackoverflow.com/questions/65305856/no-write-concern-mode-named-majority-found-in-replica-set-configuration-err

Make sure you import required libraries in app.js:

```
// importing user context
const User = require("../model/user")
const express = require('express')
const router = express.Router()
const bcrypt =  require('bcryptjs')
const jwt = require('jsonwebtoken')
```