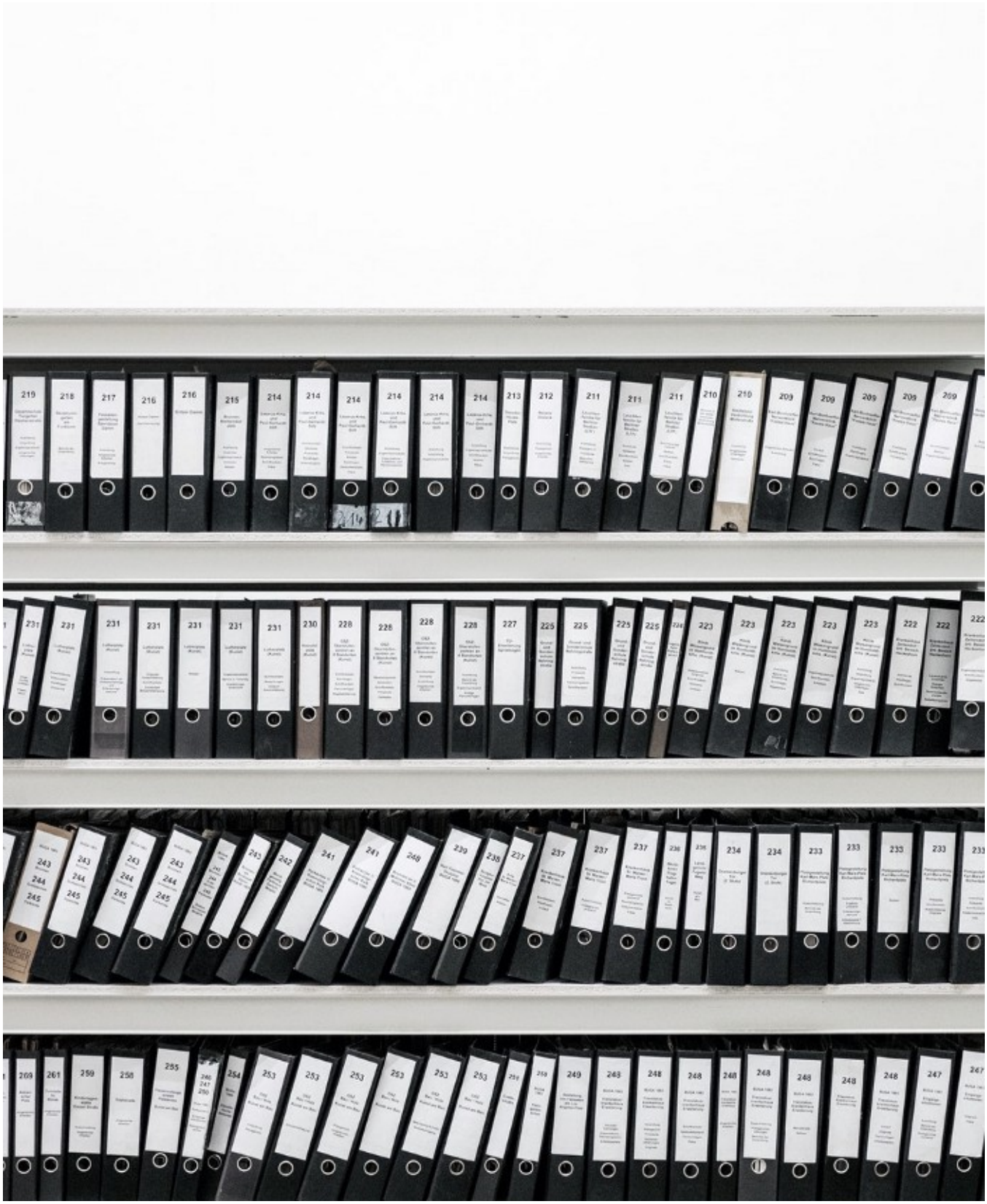


Image from Unsplash

Handling Authentication and Authorization with Node

Bilguun Batbold Follow
Mar 18 · 5 min read

In this article you will learn about handling Authentication and Authorization for your Node server.

Introduction

Let's assume you have some routes that provide resources to users. Without proper authentication and authorization, sensitive information can be compromised. Therefore, it is very important to authenticate users and provide different access rights depending on the user type.

What will be used in this tutorial

- Node.JS
- Express
- MongoDB (to store user information)
- JWT (JSON Web Tokens)
- Creating simple routes
- Creating Models / Schema
- Writing custom authentication middleware
- VS Code with Powershell as default terminal

Feel free to read my previous articles to get learn more about basics of Node / Express / MongoDB / Mongoose.

JWT

> *JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.*
> *JWT.IO allows you to decode, verify and generate JWT.*

Essentially, JWT are strings of data that can be used to authenticate and exchange information between a server and a client.
The flow of information is as follows:

- Client sends credentials to the server
- Server verifies the credentials, generates a JWT and sends it back as a response
- Subsequent requests from the client have a JWT in the request headers
- Server validates the token and if valid, provide the requested response.

This is a rough idea of how JWT are used, you can read more about them here.

Tutorial

Let's first create a new folder called `nodejs-authentication` and setup the basic app template

1. Config — Uued to retrieve jwtPrivateKey
2. joi — validation function
3. express — web framework
4. mongoose — create models and schema
5. jsonwebtoken — generate and verify JWT
6. bcrypt — hashing the password to store in the database

Install the following dependencies:

```
npm i config joi express mongoose jsonwebtoken bcrypt
```

Our project tree structure would look like below so go ahead and create the necessary folders and files.

```
-- config
--- default.json
-- middleware
--- auth.js
-- models
--- user.model.js
-- routes
--- auth.route.js
--- users.route.js
-- index.js
-- package-lock.json
-- package.json
```

In the `default.json`, add the following:

```
{
 "myprivatekey": "myprivatekey"
}
```

We will set an environment variable `myprivatekey` and use it in our application.

Creating User Model

```
const config = require('config');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const mongoose = require('mongoose');

//simple schema
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    minlength: 3,
    maxlength: 50
```

```
    maxlength: 255,
    unique: true
  },
  password: {
    type: String,
    required: true,
    minlength: 3,
    maxlength: 255
  },
  //give different access rights if admin or not
  isAdmin: Boolean
});


//custom method to generate authToken
UserSchema.methods.generateAuthToken = function() {
  const token = jwt.sign({ _id: this._id, isAdmin: this.isAdmin }, config.get('myprivatekey'));
  return token;
}

const User = mongoose.model('User', UserSchema);

//function to validate user
function validateUser(user) {
  const schema = {
    name: Joi.string().min(3).max(50).required(),
    email: Joi.string().min(5).max(255).required().email(),
    password: Joi.string().min(3).max(255).required()
  };

  return Joi.validate(user, schema);
}

exports.User = User;
exports.validate = validateUser;
```

user.model.js hosted with ❤ by GitHub view raw

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

module.exports = function(req, res, next) {
  //get the token from the header if present
  const token = req.headers["x-access-token"] || req.headers["authorization"];
  //if no token found, return response (without going to the next middelware)
  if (!token) return res.status(401).send("Access denied. No token provided.");

  try {
    //if can verify the token, set req.user and pass to next middleware
    const decoded = jwt.verify(token, config.get("myprivatekey"));
    req.user = decoded;
    next();
  } catch (ex) {
    //if invalid token
    res.status(400).send("Invalid token.");
  }
};
```

auth.js hosted with ❤ by GitHub view raw

This is our custom middleware that will be used to check if there is an existing and valid JWT present in the request headers. This is how we will be identifying the user.

Handling User Routes

```
const auth = require("../middleware/auth");
const bcrypt = require("bcrypt");
const { User, validate } = require("../models/user.model");
const express = require("express");
const router = express.Router();

router.get("/current", auth, async (req, res) => {
  const user = await User.findById(req.user._id).select("-password");
  res.send(user);
});

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

  //find an existing user
```

```
    email: req.body.email
  });
  user.password = await bcrypt.hash(user.password, 10);
  await user.save();

  const token = user.generateAuthToken();
  res.header("x-auth-token", token).send({
    _id: user._id,
    name: user.name,
    email: user.email
  });
});

module.exports = router;
```

user.route.js hosted with ❤ by GitHub view raw

Here we have defined 2 routes, one is to get the current user using the `auth` middelware we have created earlier. Seconds one is to register to user. We will use `bcrypt` to hash the password, for more information please seethe official documentation here.

Entrance to the app — Index.js

```
const config = require("config");
const mongoose = require("mongoose");
const usersRoute = require("./routes/user.route");
const express = require("express");
const app = express();


//use config module to get the privatekey, if no private key set, end the application
if (!config.get("myprivatekey")) {
  console.error("FATAL ERROR: myprivatekey is not defined.");
  process.exit(1);
}

//connect to mongodb
mongoose
  .connect("mongodb://localhost/nodejsauth", { useNewUrlParser: true })
  .then(() => console.log("Connected to MongoDB..."))
  .catch(err => console.error("Could not connect to MongoDB..."));

```

```
26  app.listen(port, () => console.log(`Listening on port ${port}...`));
```

index.js hosted with ❤ by GitHub view raw

This will be the file we will use to run the app. Let's go through how it exactly works!

Running the application

```
node index.js
```

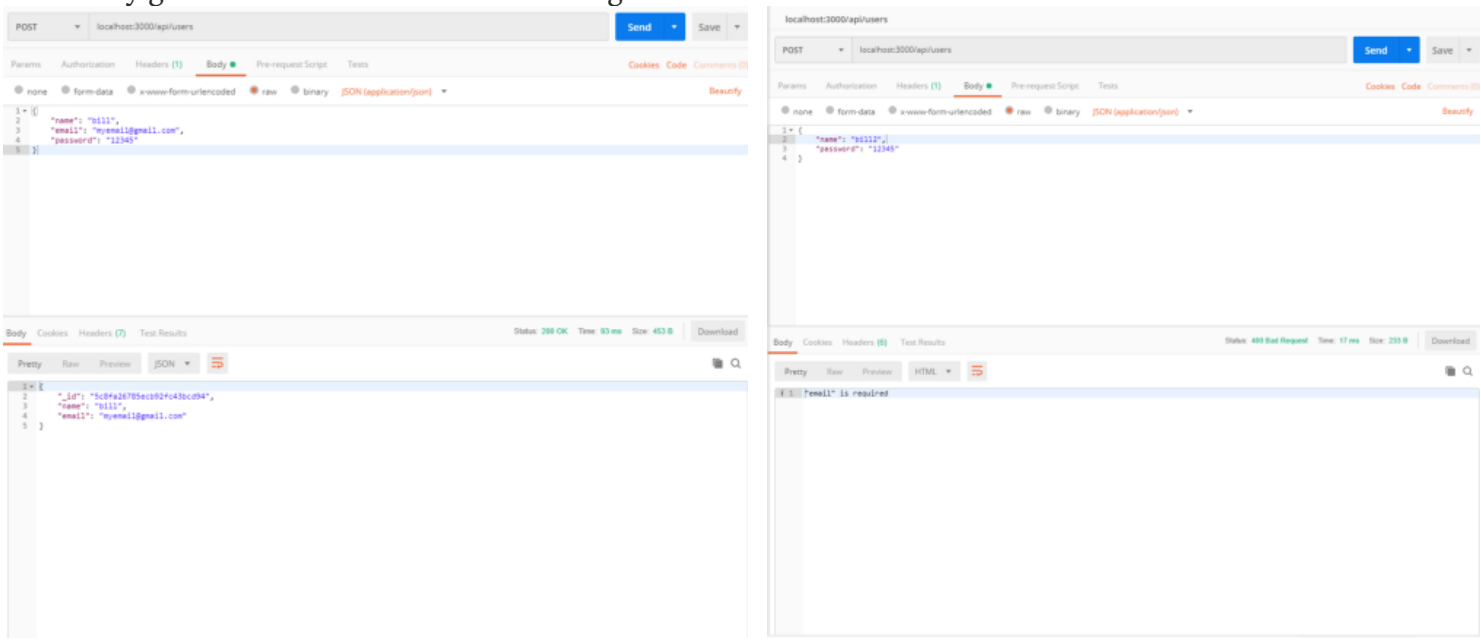
When you try to run the application, you will get `FATAL ERROR: myprivatekey is not defined`. That is because we have not set the environment variable that will be used to sign our tokens

```
$env:myprivatekey = "myprivatekey"
node index.js
```

Now when you run the app, it should print:

```
Listening on port 3000...
Connected to MongoDB...
```

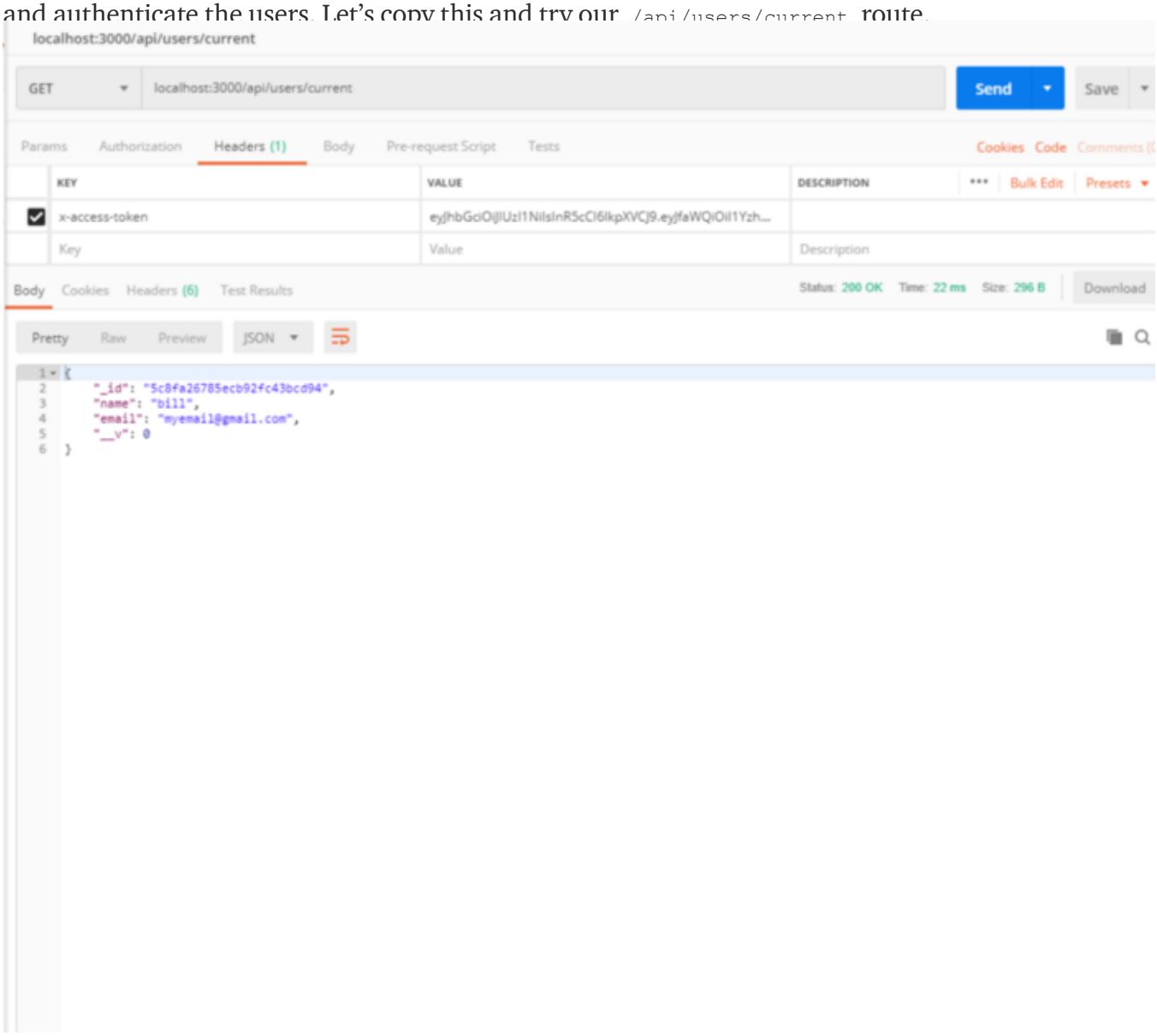
Okay great! We can now test our user registration route.



Registration works and returns the information stored on the database (without the password field)

Here we can see that when the required fields are not passed into the request body, our `validate` function returns a response with the required field

In the headers section, you can also see the generated JWT. This is what will be used to authorize and authenticate the users. Let's copy this and try our `/api/users/current` route.



Pass the JWT into the request headers

Great! We can use the JWT to identify which user is making the request.
In our JWT, we have simply used the `user id` as the payload, but we can use any other information such as *access rights*, *admin rights* etc. These information can then be used to check whether the

means that, while parties with access to the token cannot tamper with it, all payload information is publicly visible. JWT can also be used as an ***encrypted*** token on top of just signing it, which is not covered in this tutorial.
If you have had any problems following this tutorial, please do let me know below. Feel free to check out the source code here.

Bilguun132/nodejs-authentication

Contribute to Bilguun132/nodejs-authentication development by creating an account on GitHub.

github.com

. . .

If anyone finds these useful, feel free to share this or let me know should there be an error / bad practice / implementations.
Have fun coding!

Nodejs Express Authentication Jwt Mongodb

157 claps

WRITTEN BY

Bilguun Batbold

Software Engineer

Follow

Quick Code

Find the best tutorials and courses for the web, mobile, chatbot, AR/VR development, database management, data

Follow

See responses (4)

More From Medium

More from Quick Code

More from Quick Code

More from Quick Code

Advanced Python made easy

Ravin... Sep 6,... 1.7K

10 Reasons Why Python Beats PHP for Web Development

myTec... Feb 27 ... 1.2K

Local Notifications with Swift 4

Ali Fak... Jan 15 ... 613