Auth with NodeJS, Express, Mongoose and JWT

Matteo Lobello · Follow
Published in ITNEXT · 3 min read · Sep 25, 2019

99

Security of your database is one of the most important factor to consider when building a new project.

There are several ways to implement a solid **authentication** system.
In this article I'll show you how to make one using JWT, completely from scratch and without the need to use third party services.

What technologies are we going to use?

- **NodeJS**: a tool to server-side execute JavaScript code.
- **Express**: a micro-framework to make server development faster.
- **Mongoose**: a library that helps us connect to our MongoDB instance.
- **JWT** (JSON Web Token): an encrypted string that gets generated by the server and is stored by the client. On every request, that string will be sent by the client and will be verified by the server.
- **Bcrypt**: a library we will use to hash our passwords.
- **BodyParser**: will help us to retrieve body parameters when handling a request.

Setup our environment

First of all, be sure to have NodeJS installed. If you haven't, check out this link. Then, create a new MongoDB instance. You can create one here, for free.
Finally, install NPM packages in our project folder via command line.

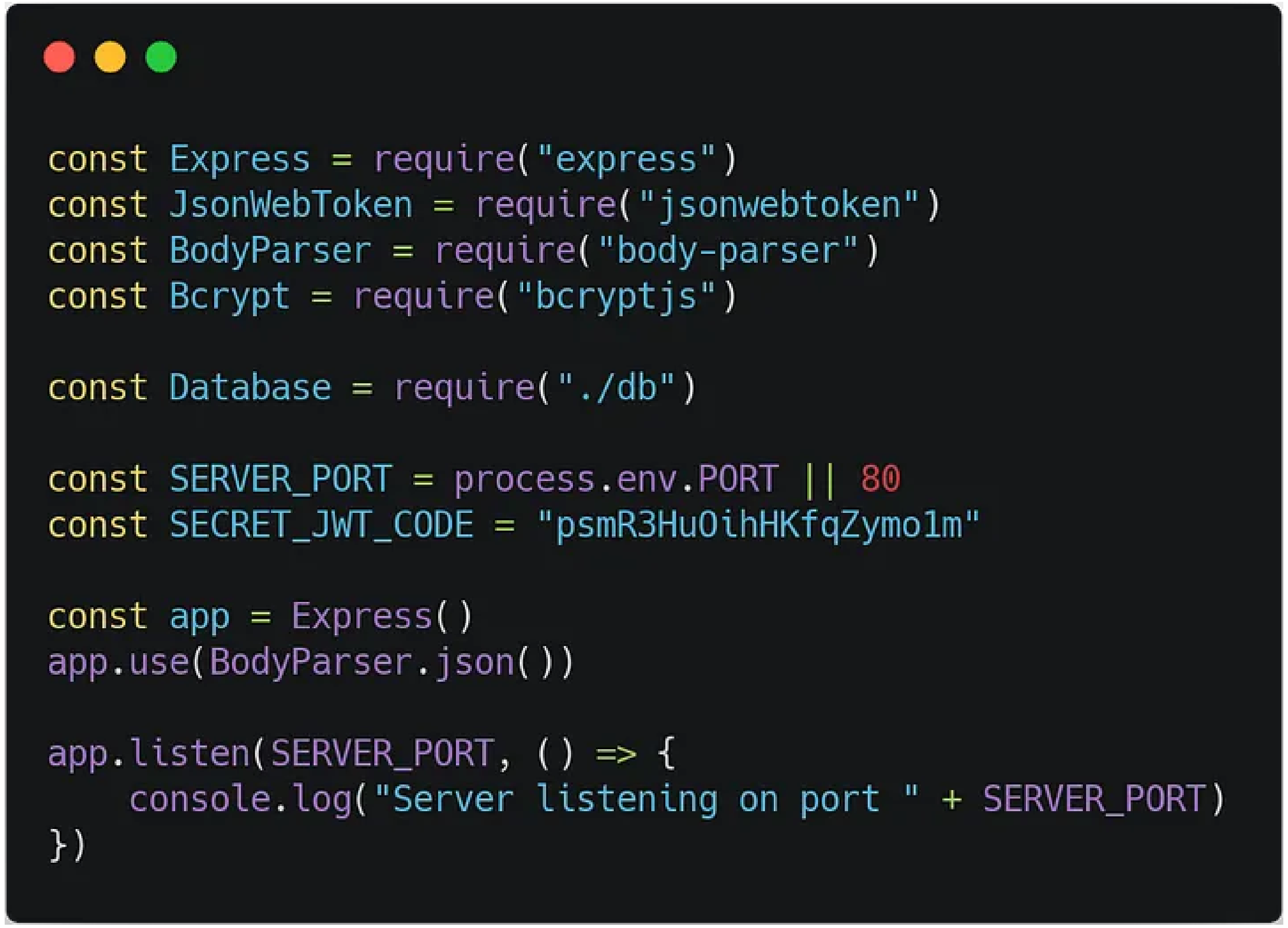
```
npm install express jsonwebtoken body-parser bcryptjs
```

Ready to code!

Now, we need to create our server scripts:

- **app.js**, in which we'll handle all our server requests.
- **db.js**, which will contain our Database models.

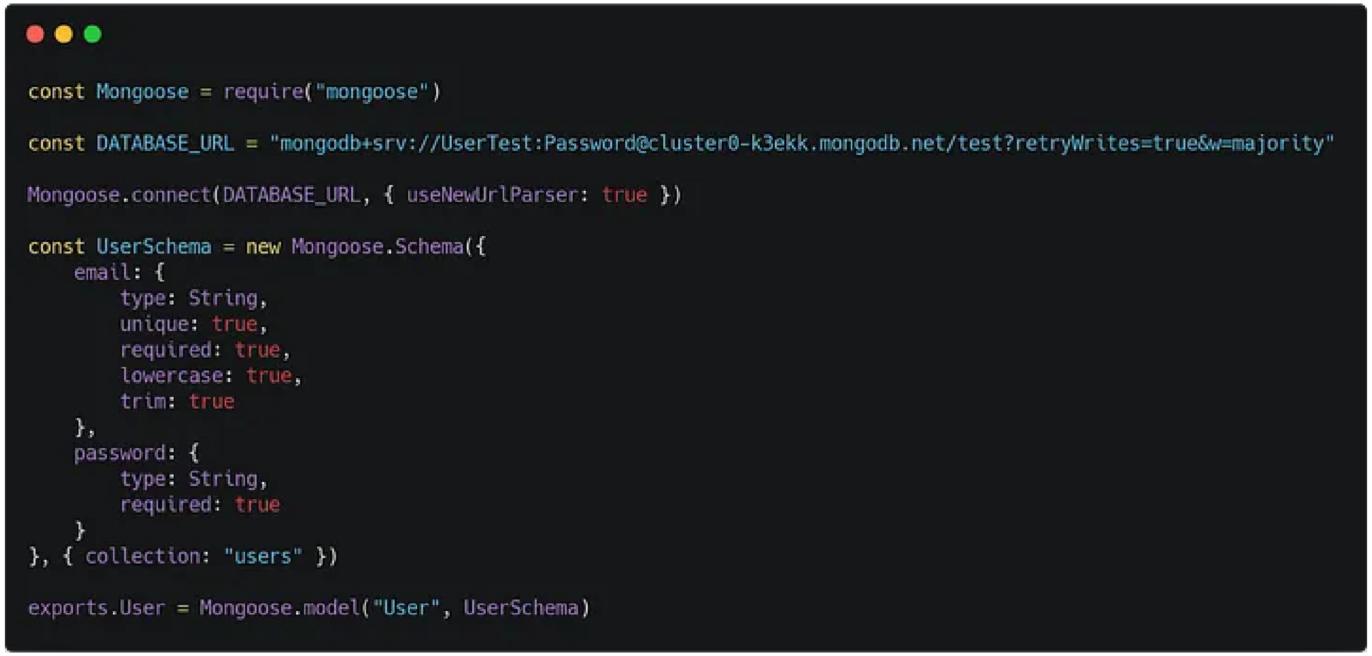
Let's create a very simple **Express** application: it will be the basic structure of our app.



```
const Express = require("express")
const JsonWebToken = require("jsonwebtoken")
const BodyParser = require("body-parser")
const Bcrypt = require("bcryptjs")

const Database = require("./db")

const SERVER_PORT = process.env.PORT || 80
const SECRET_JWT_CODE = "psmR3Hu0ihHKfqZymo1m"

const app = Express()
app.use(BodyParser.json())

app.listen(SERVER_PORT, () => {
    console.log("Server listening on port " + SERVER_PORT)
})
```

The “SECRET_JWT_CODE” is the encryption key we are going to use when generating our tokens.

Before proceeding implementing our endpoints, we should get a look at the **db.js** file.

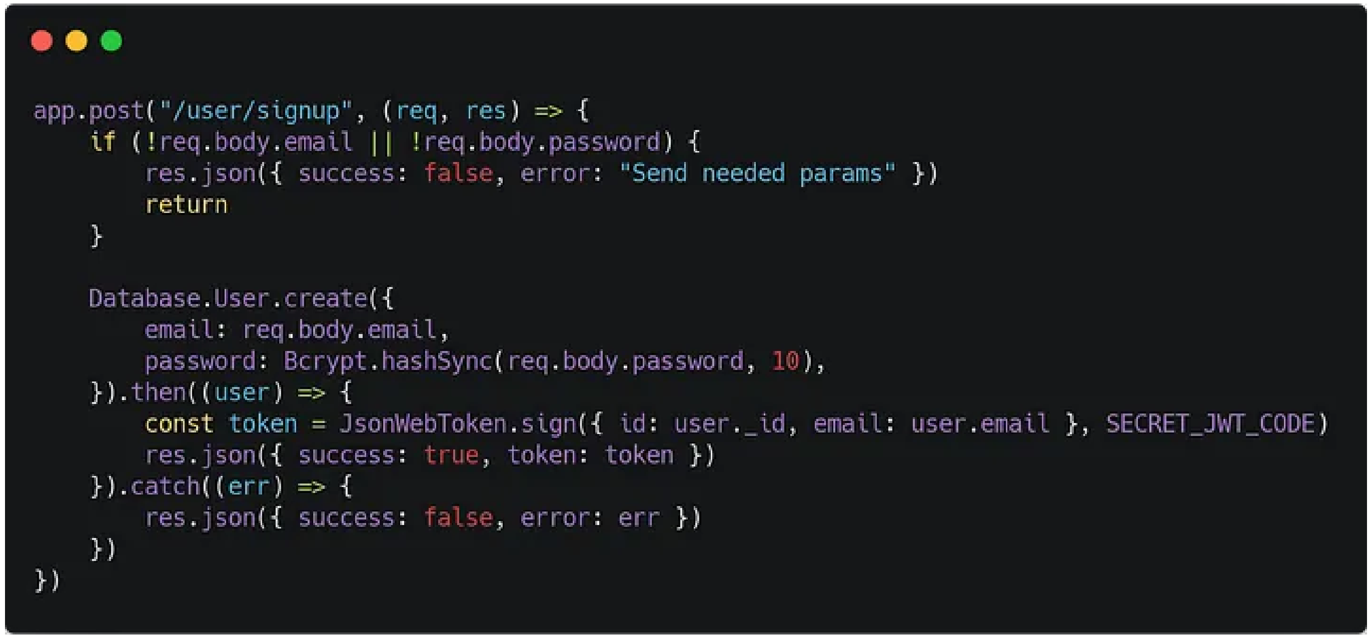


```
const Mongoose = require("mongoose")

const DATABASE_URL = "mongodb+srv://UserTest:Password@cluster0-k3ekk.mongodb.net/test?retryWrites=true&w=majority"

Mongoose.connect(DATABASE_URL, { useNewUrlParser: true })

const UserSchema = new Mongoose.Schema({
    email: {
        type: String,
        unique: true,
        required: true,
        lowercase: true,
        trim: true
    },
    password: {
        type: String,
        required: true
    }
}, { collection: "users" })

exports.User = Mongoose.model("User", UserSchema)
```

db.js

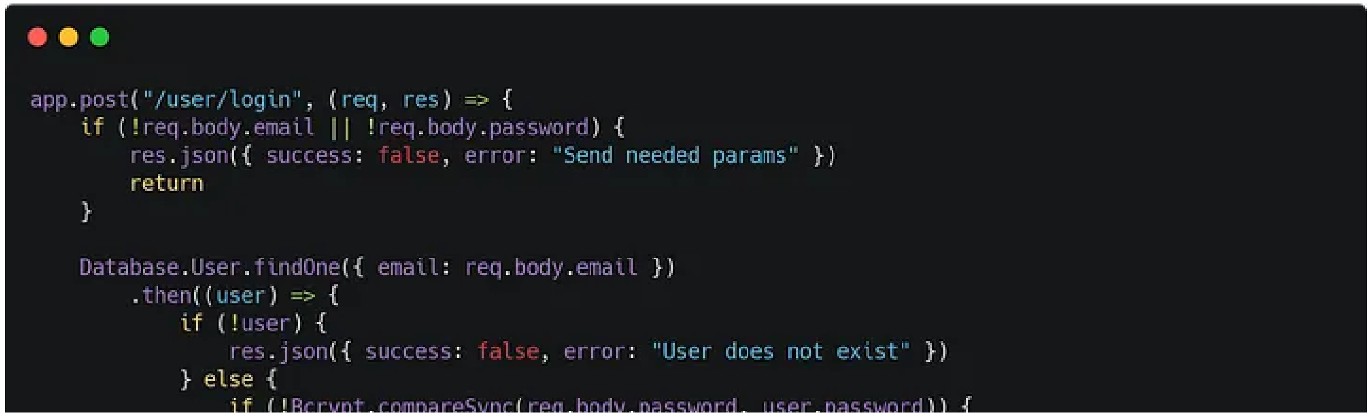
This file essentially creates our DB model schemas. This means that each user related data is going to be saved under a document (collection) called “users” and will have two attributes, “email” and “password”. Of course, you can create as many *models* as you want, with as many *attributes* as you want.

Now we need to implement our **routing** system. The app will essentially have two endpoints, the first one for the **sign up** and the second one for the **login**.

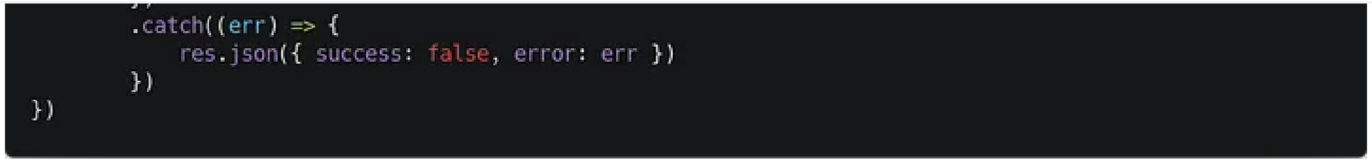


```
app.post("/user/signup", (req, res) => {
    if (!req.body.email || !req.body.password) {
        res.json({ success: false, error: "Send needed params" })
        return
    }

    Database.User.create({
        email: req.body.email,
        password: Bcrypt.hashSync(req.body.password, 10),
    }).then((user) => {
        const token = JsonWebToken.sign({ id: user._id, email: user.email }, SECRET_JWT_CODE)
        res.json({ success: true, token: token })
    }).catch((err) => {
        res.json({ success: false, error: err })
    })
})
```

app.js — sign up



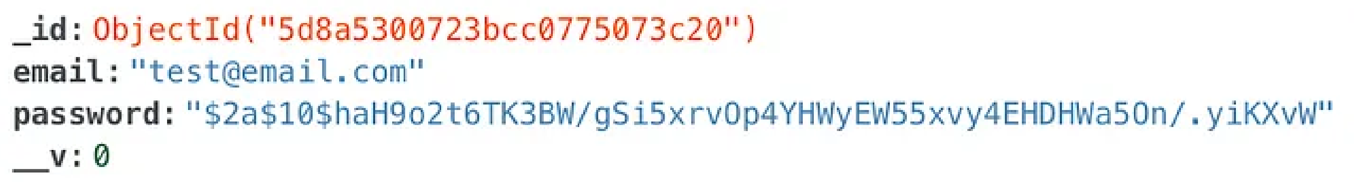
```
app.post("/user/login", (req, res) => {
    if (!req.body.email || !req.body.password) {
        res.json({ success: false, error: "Send needed params" })
        return
    }

    Database.User.findOne({ email: req.body.email })
        .then((user) => {
            if (!user) {
                res.json({ success: false, error: "User does not exist" })
            } else {
                if (!Bcrypt.compareSync(req.body.password, user.password)) {
```



```
        .catch((err) => {
            res.json({ success: false, error: err })
        })
})
```

app.js — login

It works!



```
_id: ObjectId("5d8a5300723bcc0775073c20")
email: "test@email.com"
password: "$2a$10$haH9o2t6TK3BW/gSi5xrv0p4YHWyEW55xvy4EHDHWa50n/.yiKXvW"
__v: 0
```

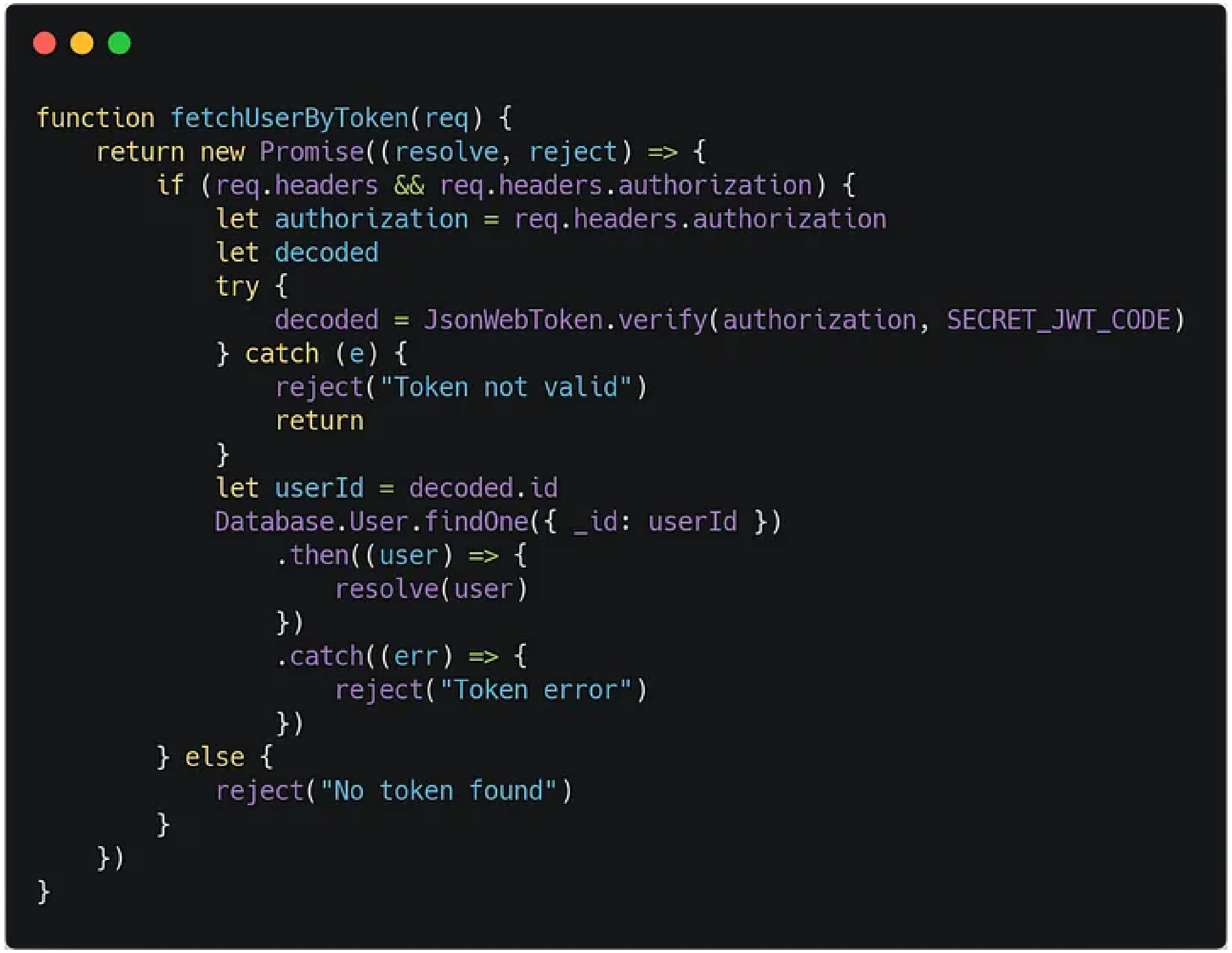
Screenshot of the data saved on our DB after having made a call to our /user/signup endpoint

Final steps

We successfully implemented our authentication system. Now, the client should **save** the token after the login and **send it back** every time it makes a new request.

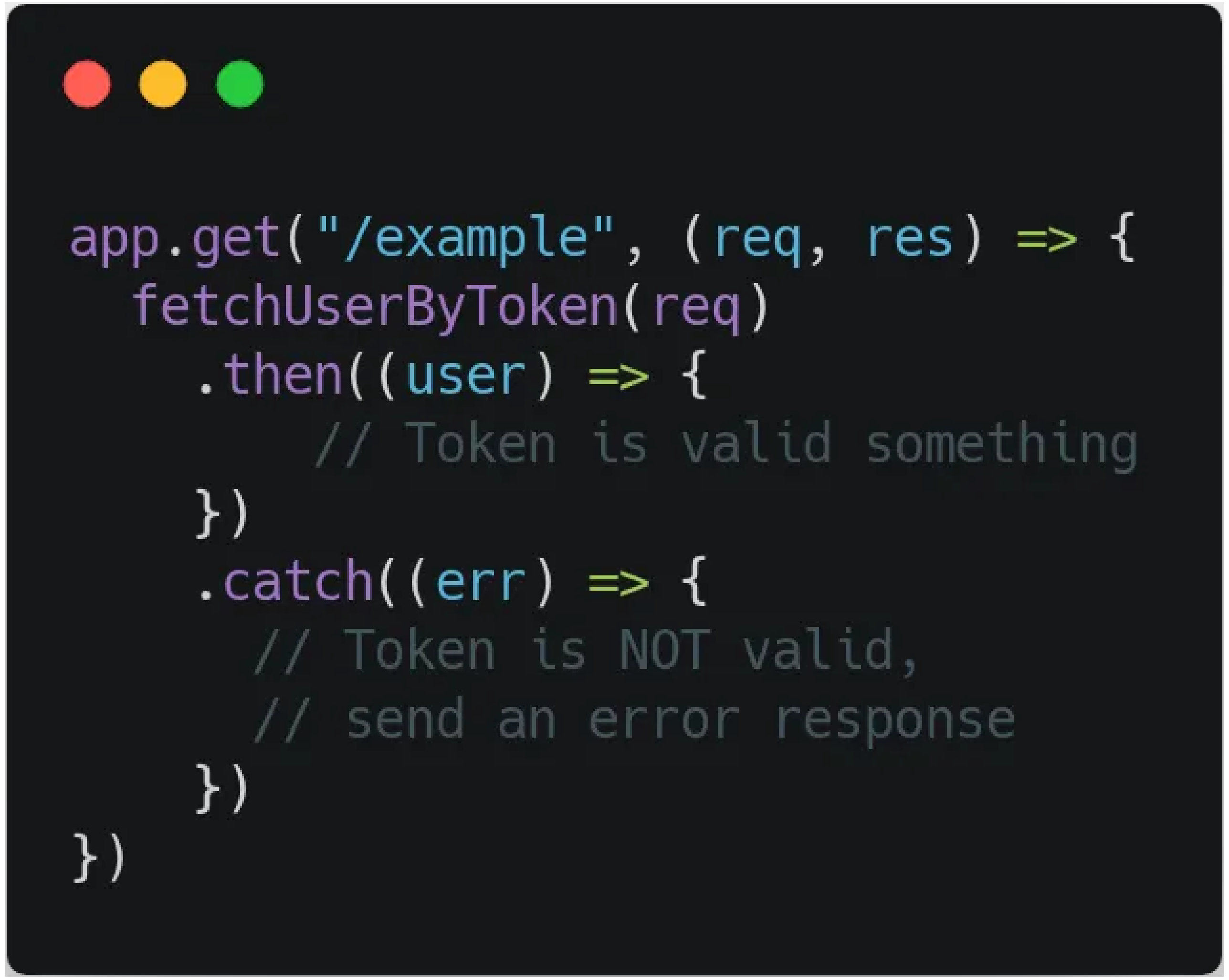
Remember that Json Web Tokens have an **expiration** time. There are many ways to stay the user logged in. For example, whenever it happens, the **client** could **refresh** the token making a new login call.

On the **backend side**, this is how you could check if the token is **valid**:



```
function fetchUserByToken(req) {
    return new Promise((resolve, reject) => {
        if (req.headers && req.headers.authorization) {
            let authorization = req.headers.authorization
            let decoded
            try {
                decoded = JsonWebToken.verify(authorization, SECRET_JWT_CODE)
            } catch (e) {
                reject("Token not valid")
                return
            }
            let userId = decoded.id
            Database.User.findOne({ _id: userId })
                .then((user) => {
                    resolve(user)
                })
                .catch((err) => {
                    reject("Token error")
                })
        } else {
            reject("No token found")
        }
    })
}
```

app.js



```
app.get("/example", (req, res) => {
  fetchUserByToken(req)
    .then((user) => {
        // Token is valid something
    })
    .catch((err) => {
      // Token is NOT valid,
      // send an error response
    })
})
```

app.js

You can find the Github gist here!😊

JavaScript Authentication Nodejs Expressjs Jwt

Written by Matteo Lobello

Follow

16 Followers · Writer for ITNEXT

matteolobello.com

More from Matteo Lobello and ITNEXT

```
<!DOCTYPE html>
<html>
<head>
  <title>Our first web component</title>
  <script src="clock-element.js"></script>
</head>
<body>
  <h1>This is our first component!</h1>
  <clock-element></clock-element>
</body>
```

Matteo Lobello in ITNEXT

Deep dive into WebComponents

The native way to make dynamic component-based web apps.

5 min read · Jan 20, 2020

58 1

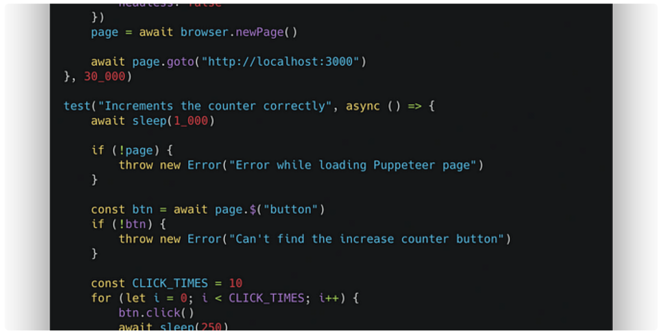
Patrick Kalkman in ITNEXT

Dependency Injection in Python

Building flexible and testable architectures in Python

· 13 min read · Apr 14

632 5



```
    })
    page = await browser.newPage()

    await page.goto("http://localhost:3000")
}, 30_000)

test("Increments the counter correctly", async () => {
    await sleep(1_000)

    if (!page) {
        throw new Error("Error while loading Puppeteer page")
    }

    const btn = await page.$("button")
    if (!btn) {
        throw new Error("Can't find the increase counter button")
    }

    const CLICK_TIMES = 10
    for (let i = 0; i < CLICK_TIMES; i++) {
        btn.click()
        await sleep(250)
```

Gaurav Mukherjee in ITNEXT

Angular 16 is huge

Matteo Lobello in ITNEXT

UI-Testing TypeScript React apps with Puppeteer and Jest

At the time of writing this article, angular just published its first release candidate version ...

5 min read · Apr 16

972 8

Configuring Puppeteer in your React project.

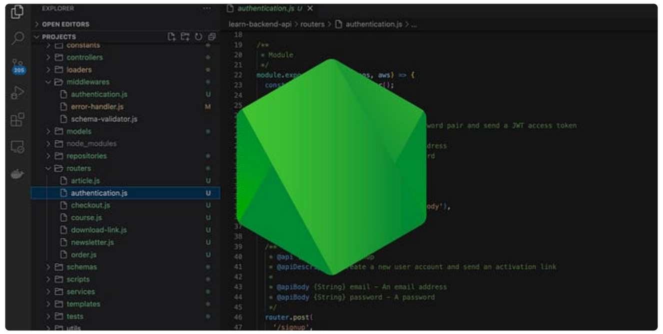
2 min read · May 4, 2021

69 1

See all from Matteo Lobello

See all from ITNEXT

Recommended from Medium



Melih Yumak in JavaScript in Plain English

Razvan L in JavaScript in Plain English

Nodejs Developer Roadmap 2023

Explore nodejs developer roadmap for 2023. A step-by-step guide to how to become...

· 7 min read · Jan 30

599 12

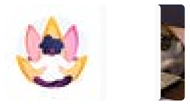
Build a Custom Payload Validation Middleware for ExpressJS

The second and probably most important part of the data handling process after...

· 4 min read · Mar 17

129

Lists



Stories to Help You Grow as a Software Developer

19 stories · 22 saves

Staff Picks

300 stories · 62 saves

Lukas Wimhofer in Level Up Coding

Ibrahim Ahmed in Bootcamp

API development with nodejs, express and typescript from…

This is the fifth part of the series "API development with nodejs, express and…

✦ · 17 min read · Feb 15

57

How I Optimized An API Endpoint To Make It 10x Faster

When it comes to building web applications, performance is one of the most important…

✦ · 3 min read · Jan 12

275 7

Mohammad Faisal in JavaScript in Plain English

A Comprehensive Guide to NodeJS Security Best Practices

Create a Secured NodeJS application

· 7 min read · Jan 16

18 3

Elson Correia in Before Semicolon

How to Set up a TypeScript + NodeJs Server (2023)

With new releases and tools, setting up a node server has become super simple and...

· 7 min read · Feb 18

121 2

See more recommendations