**Autenticando un API Rest con NodeJS y JWT (JSON Web Tokens)**

Hoy en día las APIs son parte importantísima de nuestro ecosistema de desarrollo, ya que gracias a ellas tenemos acceso a miles de millones de datos en distintas empresas u organismos, ya que nos permiten mandar pedir información, o utilizar servicios de terceros de forma sencilla (normalmente…).

Sin embargo, dejar expuesto un servicio que el punto es que sea bajo demanda, es decir, que para tener acceso a este haya que pagar por petición, por mes o cosas así, nos hace que estas APIs deban protegerse, así no cualquiera puede realizar peticiones a nuestros servicios.

Hay distintos métodos para poder dar acceso a un API, por ejemplo la autenticación a través de tokens y es la que veremos usando JWT.

**Autenticación por tokens**

Primero, ¿qué demonios es un servicio basado en autenticación por tokens?, en términos simples es que a través de una cadena de caracteres enviadas desde el cliente hacia nuestro API, algo así como una “contraseña”, entonces nosotros detectamos la token y podemos revisar a qué usuario pertenece, si es válida o no, mantener registros de peticiones, y mucho más.

**¿Cómo funciona?**

El funcionamiento de JWT es bastante simple y se basa en 6 pasos:

* Primero: autentica usando credenciales regulares (usuario-contraseña normalmente)
* Segundo: Una vez que ya autenticó en el servidor, genera una cadena de caracteres que contiene la token de JWT integrada.
* Tercero: Esa token es enviada al cliente
* Cuarto: La token se almacena en el lado del cliente
* Quinto: La token se manda al lado del servidor en cada petición que se realiza.
* Sexto: El servidor valida la token y otorga (o no) acceso al recurso que el cliente solicita.

Y listo, este es el proceso a forma de texto. Aquí lo vemos a forma de diagrama:

Diagram

Description automatically generated

**Construyendo nuestra API**

En esta ocasión no voy a explicar qué es un API, cómo funciona, qué métodos usar y demás, solamente haremos una demo, si quieres más información de esto puedes consultar mi otra guía:

**[Desarrollando una sencilla API REST con NodeJS y Express](https://medium.com/@asfo/desarrollando-una-sencilla-api-rest-con-nodejs-y-express-cab0813f7e4b" \t "_blank)**

[Actualmente aún hay muchas empresas que siguen utilizando el clásico WebService viejo con XML (te estoy viendo a ti…](https://medium.com/@asfo/desarrollando-una-sencilla-api-rest-con-nodejs-y-express-cab0813f7e4b" \t "_blank)

[medium.com](https://medium.com/@asfo/desarrollando-una-sencilla-api-rest-con-nodejs-y-express-cab0813f7e4b" \t "_blank)

Vamos a instalar unas dependencias para poder trabajar con esta guía en nuestro nuevo proyecto:

npm install --save express jsonwebtoken

Como dato, body-parser ya viene integrado en express, así que ya no se ocupa, porque habrán visto que en algunas cosas como mi guía pasada se ocupaba.

Vamos a crear una carpeta llamada configs y dentro un archivo llamado config.js . Su contenido es súper sencillo:

module.exports = {  
 llave: "miclaveultrasecreta123\*"  
}

¿Por qué creamos esto aparte?, para no encimar esta sección al código principal y si les es útil, si no pues, pueden integrar esto dentro del index.js sin problemas.

¿Y qué significa?, bueno, básicamente JWT necesita una “contraseña maestra” por así llamarlo que usará para encriptar la información y aquí la pondremos, ustedes deberán usar una clave segura que nadie pueda conocer fácilmente.

Ahora sí, vamos a nuestro archivo index.js y le pondremos un código simple:

const express = require('express'),  
 bodyParser = require('body-parser'),  
 jwt = require('jsonwebtoken'),  
 config = require('./configs/config'),  
 app = express();// 1  
app.set('llave', config.llave);// 2  
app.use(bodyParser.urlencoded({ extended: true }));// 3  
app.use(bodyParser.json());// 4  
app.listen(3000,()=>{  
 console.log('Servidor iniciado en el puerto 3000')   
});// 5  
app.get('/', function(req, res) {  
 res.send('Inicio');  
});

¿Qué hace este código?, bueno a nivel básico generamos las constantes requeridas, después, indicamos la configuración de nuestra clave ultra secreta (1), seteamos para que el body-parser nos convierta lo que viene del cliente (2), lo pasamos a JSON (3), arrancamos el servidor (4) y generamos un “punto de inicio” (5).

Si ejecutamos: node index.js veremos en el navegador en la URL [**http://localhost:3000/**](http://localhost:3000/)la palabra “Inicio”.

**Agregando el sistema de autenticación**

Ya tenemos todo casi listo, ahora debemos agregar un método por POST que nos permita autenticar, en este caso no usaremos bases de datos ni nada, será simplemente texto plano que autenticará por una condicional:

app.post('/autenticar', (req, res) => {  
 if(req.body.usuario === "asfo" && req.body.contrasena === "holamundo") {  
 const payload = {  
 check: true  
 };  
 const token = jwt.sign(payload, app.get('llave'), {  
 expiresIn: 1440  
 });  
 res.json({  
 mensaje: 'Autenticación correcta',  
 token: token  
 });  
 } else {  
 res.json({ mensaje: "Usuario o contraseña incorrectos"})  
 }  
})

Esta parte se agrega al fondo del archivo index.js lo que haremos será que si el usuario no es “asfo” y la contraseña no hace empate con “holamundo” le diremos que hay algo mal, pero si empatan, entonces estará correcto y generaremos la token con hasta 24 horas de tiempo de vida (es posible hacerlo sin expiración).

Haciendo la petición tendríamos una respuesta como la siguiente:

{"message": "Autenticación correcta","token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjaGVjayI6dHJ1ZSwiaWF0IjoxNTY2MjQ4MTYwLCJleHAiOjE1NjYyNDk2MDB9.1olkcT6F\_e2y2lN9zzHgFcBYnofnD8cmOb7EEKVjSdQ"}

De esta forma como podemos ver se nos envió una token, esta deberemos guardarla y con ella haremos las peticiones subsecuentes.

**Creando un middleware**

A grandes rasgos un middleware es algo así como “código intermedio”, que estará en este caso entre la petición del cliente y nuestro código del API para que nos proteja de peticiones indebidas (que no traen una token o la token es inválida). El código luce más o menos así:

const rutasProtegidas = express.Router();   
rutasProtegidas.use((req, res, next) => {  
 const token = req.headers['access-token'];  
   
 if (token) {  
 jwt.verify(token, app.get('llave'), (err, decoded) => {   
 if (err) {  
 return res.json({ mensaje: 'Token inválida' });   
 } else {  
 req.decoded = decoded;   
 next();  
 }  
 });  
 } else {  
 res.send({   
 mensaje: 'Token no proveída.'   
 });  
 }  
 });

Lo que hace será revisar si viene la header “access-token” en las cabeceras y de ahí verifica la JWT contra el servicio. En caso de que no se envíe o sea inválida retornamos un mensaje con error. En caso contrario, le permite a nuestro API seguir su camino correcto.

Ahora que tenemos nuestro Middleware, vamos a crear un nuevo endpoint que requiera de la token, le nombraremos: datos y será a través de GET :

app.get('/datos', rutasProtegidas, (req, res) => {  
 const datos = [  
 { id: 1, nombre: "Asfo" },  
 { id: 2, nombre: "Denisse" },  
 { id: 3, nombre: "Carlos" }  
 ];  
   
 res.json(datos);  
});

Ahora, si reiniciamos nuestro servidor, hagamos una petición a este endpoint sin la token:

Graphical user interface, text, application, email

Description automatically generated

Si proveemos la token a través de las headers:

Graphical user interface, text, application, email

Description automatically generated

Y si la token es inválida:

Graphical user interface, application, email

Description automatically generated

**Conclusión y extras**

Como podemos observar de forma muy sencilla es posible evitar que usuarios no autorizados obtengan información de nuestras APIs, el uso de JWT nos da muchísimas facilidades, en este caso para dar tiempo de vida a las sesiones, puede incluso llevar datos como el user id integrado y al obtenerlo nos pueda ahorrar una solicitud a la base de datos, o muchas otras ventajas pero la principal, es la seguridad que nos otorgará a nuestro API.

Debido a que no me es viable generar un repo para este tutorial, te dejo un Gist en su lugar de nuestro index.js: