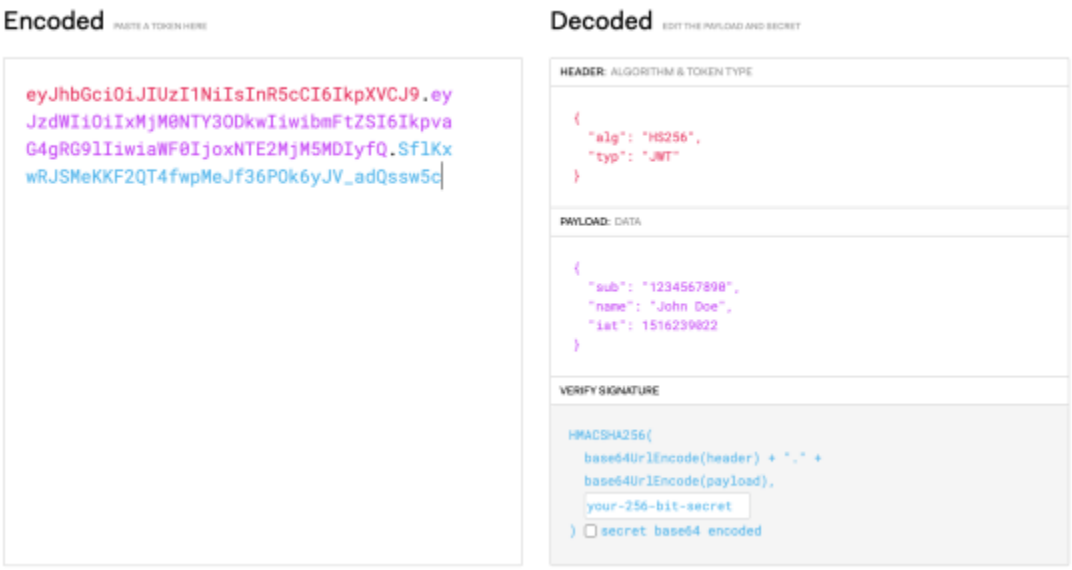
Objetivo

En esta guía aprenderemos a crear un API REST añadiendo una capa de seguridad usando el Estándar JWT

Introducción

Json Web Token (JWT) es un estándar abierto basado en JSON, diseñado para proteger el acceso a API REST. Para este propósito JWT genera un token o cadena de texto codificado en Base64 . Esta cadena esta conformado por 3 partes separados por un punto:

- El encabezado (header)
- payload
- firma de verificación



El encabezado (header)

En header consta a su vez de dos partes: el tipo de token (JWT) y el algoritmo (HS256, HMAC, SHA256 o RSA).

Ejemplo:

```
{
"alg":"HS256",
"typ":"JWT"

}
```

payload

Se utiliza para almacenar un conjunto de campos adicionales que se va a incluir en el token. Un ejemplo de uso puede ser para almacenar los detalles de identificación del usuario. Ejemplo:

```
{
  user_id: 303,
  account_id: 2341,
  email: 'maria@uajms.edu.bo',
  full_name: 'Maria Gonzales',
  default_language: 'es_ES'
}
```

En este campo no se debe guardar información sensible como contraseñas, claves secretas, numero de tarjetas, etc. ya que esta información no esta cifrada. Solo se debe incluir datos esenciales.

Firma de verificación

Se uso es para comprobar que el mensaje no se haya alterado en el camino. Para generar este campo se debe tomar el header, el payload, un secret y el algoritmo. Por ejemplo si se desea utilizar el algoritmo HMAC SHA256, se debe generar de la siguiente forma:

```
HMACSHA256(
  base64urlencode(header)+"."+base64urlencode(payload)
   ,secreto
 )
```

Para poner en práctica estos conceptos puede revisar el Depurador JWT que permite codificar, decodificar, verificar y generar tokens JWT.

¿Cómo debo implementar JWT?

En el siguiente diagrama se muestra cómo se obtiene el token JWT para tener acceso a API o recursos protegidos.

Figure 1:

Los pasos se resumen en:

1. Se debe contar con una clave secreta para generar los tokens
2. La aplicación o cliente realiza una petición POST con el usuario y contraseña para realizar el proceso de login.
3. Se comprueba que el usuario y contraseña son correctos, si son correctos se genera el token JWT para devolverlo al usuario.
4. A partir de ahí la aplicación cliente, solicita recursos protegidos con el token JWT proporcionado.
5. En el servidor se verifica el token JWT con la clave secreta, si el token es válido.

JWT y NODEJS

Para poder trabajar con JWT en NODEjs debemos instalar la dependencia `jsonwebtoken`, esta librería puede trabajar conjuntamente con `express` y `body-parser`. Una vez iniciado el proyecto instalamos la dependencia:

```
npm install express jsonwebtoken
```

Añadimos las dependencias en el siguiente orden:

app.js

```
const express = require('express'),
      bodyParser = require('body-parser'),
      jwt = require('jsonwebtoken'),
      config = require('./configs/config'),
      app = express();
// 1
app.set('llave', "mi llave de crifrado para el token");
// 2
app.use(bodyParser.urlencoded({ extended: true }));
// 3
app.use(bodyParser.json());

// 4
app.get('/', function(req, res) {
    res.send('Inicio');
});

app.get('/public',(req, res)=>{
    res.send("Ruta publica");
});

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

     res.json(productos);
});

app.post('/login',(req, res)=>{
```

```
    res.send("Formulario de login");
});

// 5
app.listen(3000,()=>{
    console.log('Servidor iniciado en el puerto 3000')
});
```

Donde:

1. Indicamos la configuración de nuestra clave ultra secreta,
2. Seteamos para que el body-parser nos convierta lo que viene del cliente (2)
3. Lo pasamos a JSON las peticiones
4. Arrancamos el servidor
5. Generamos un “punto de inicio” de nuestro servidor

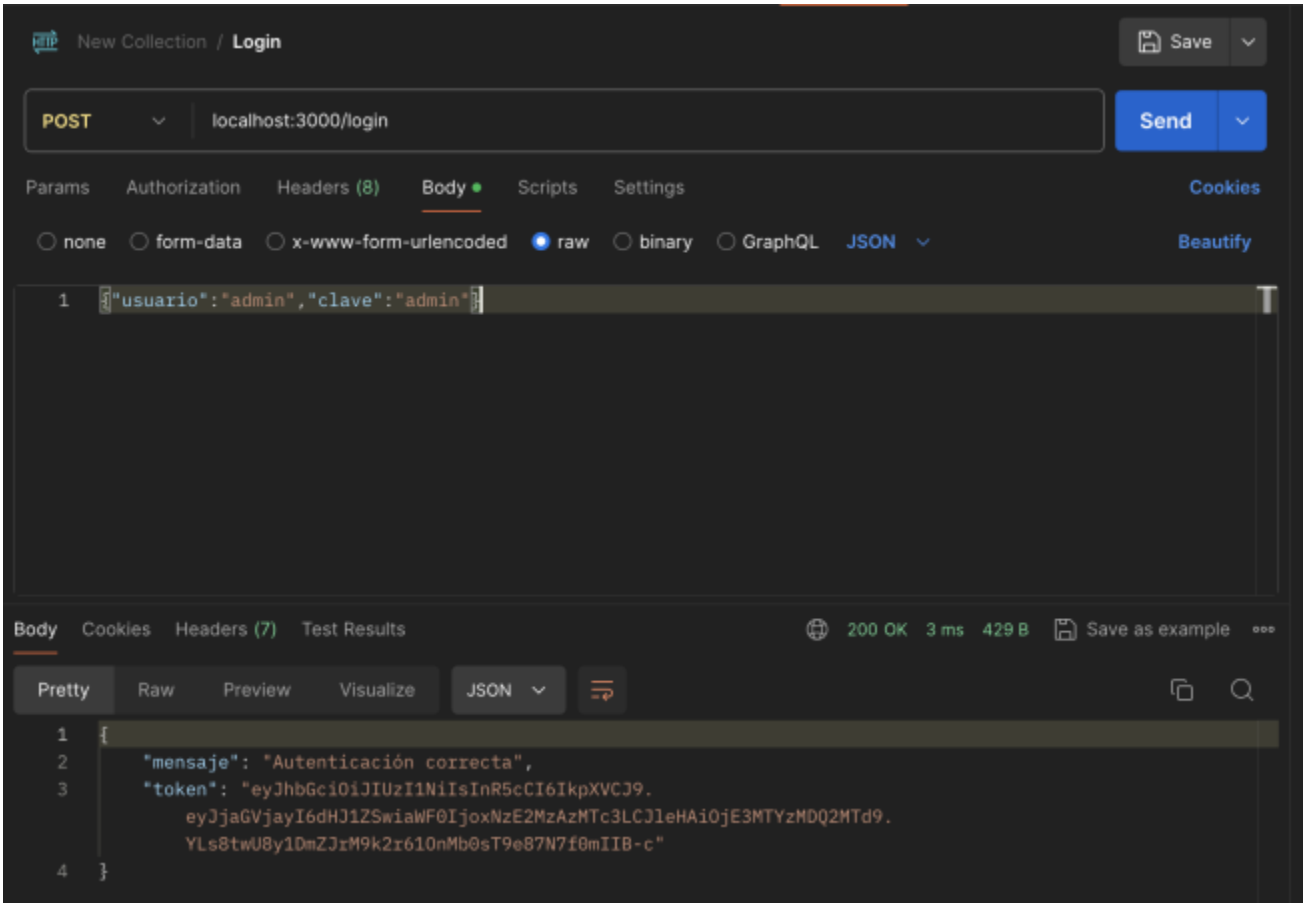
Si ejecutamos `node app.js` y abrimos desde el navegador http://localhost:5000/ veremos la palabra inicio

Generando el token

Agregamos un método POST que simula la autenticación de un usuario en la Base de Datos y si el usuario es valido generamos el token:

```
app.post('/login', (req, res) => {
    if(req.body.usuario === "admin" && req.body.clave === "admin") {
      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"})
    }
})
```

Haciendo una petición POST al endpoint /login con la aplicación POSTMAN



Una vez obtenida la respuesta con el token jwt, este será utilizado para las futuras peticiones donde se requiere validar un token:

```
{
"mensaje": "Autenticación correcta",
"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjaGVjayI6dHJ1ZSwiaWF0IjoxNzE2MzAzMTc3LCJleHAi
OjE3MTYzMDQ2MTd9.YLs8twU8y1DmZJrM9k2r61OnMb0sT9e87N7f0mIIB-c"
}
```

Verificar Token

Para la verificación de tokens debemos añadir una función intermedia, el cual se encargará de realizar la verificación del token:

```
const rutasProtegidas = express.Router();
rutasProtegidas.use((req, res, next) => {
    const token = req.headers['token-de-acceso'];

    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.'
        });
    }
 });
```

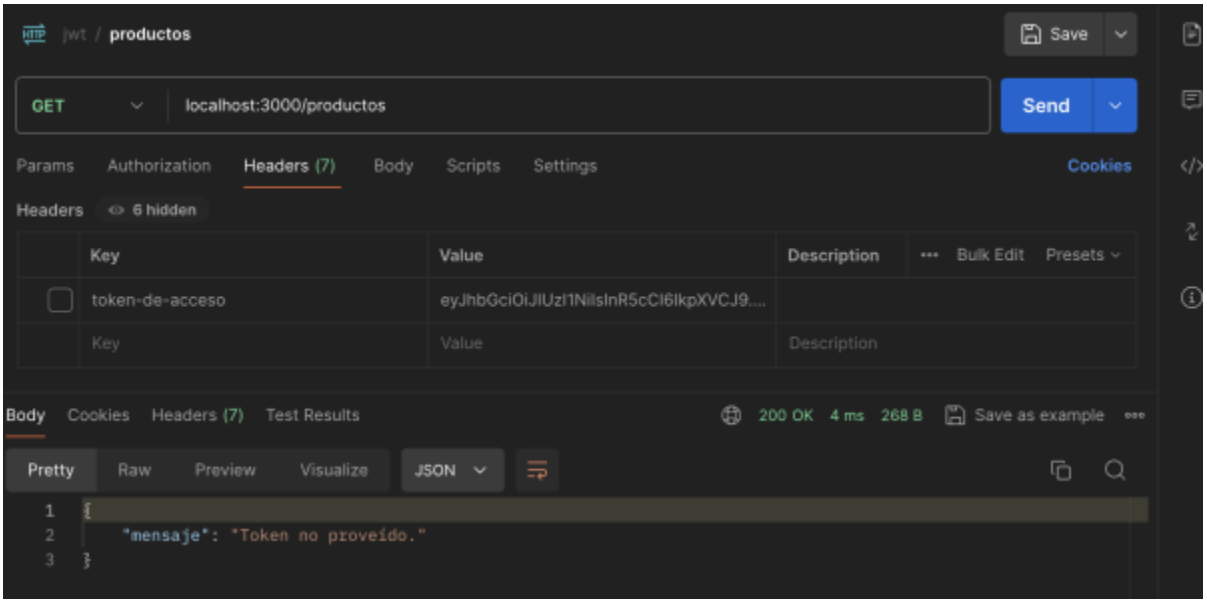
Dentro del POSTMAN debemos enviar un “token-de-acceso” con el valor del token en la cabecera.

Finalmente añadimos nuestra función intermedia a cada endpoint que requiera un token de verificación:

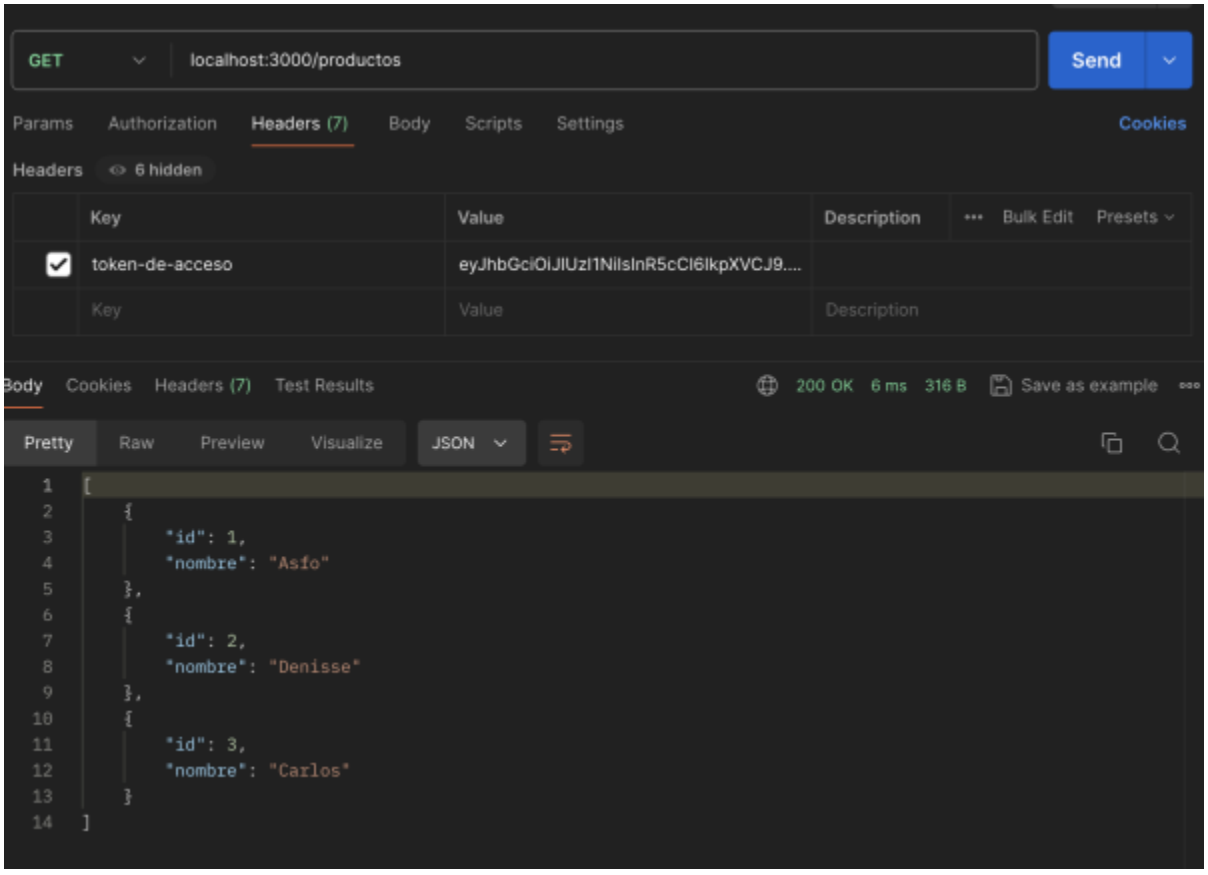
```
app.get('/datos', rutasProtegidas, (req, res) => {
 const datos = [
  { id: 1, nombre: "Asfo" },
  { id: 2, nombre: "Denisse" },
  { id: 3, nombre: "Carlos" }
 ];

 res.json(datos);
});
```

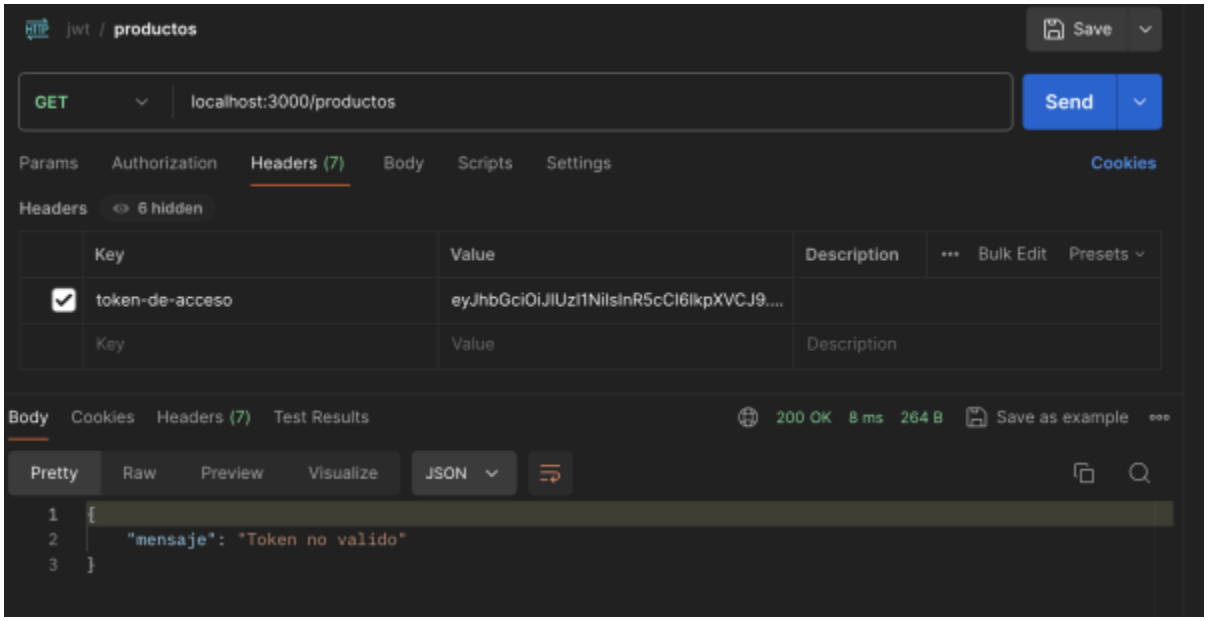
Reiniciamos el servidor y hacemos una petición con POSTMAN sin enviar el token



Enviamos el token a través del parámetro de cabecera `access-token`:



Probando con un token no válido:



URL del repositorio:

https://github.com/F1852D160/04-jwt.git