Seguridad basada en tokens JWT

Este proyecto implementa un sistema de autenticación de usuarios utilizando tokens JWT (JSON Web Tokens) en un servidor construido con Node.js y Express, y conectado a una base de datos SQLite. A través de un login seguro, los usuarios obtienen un token que les permite acceder a rutas protegidas.

conexion (1).js

Este archivo se encarga de establecer la conexión con la base de datos SQLite.

javascript

CopiarEditar

```
const sqlite3 = require("sqlite3").verbose();

const db = new sqlite3.Database("./clientes.sqlite", (err) => {
    if (err) return console.error(err.message);
    console.log("Conectado a la base de datos SQLite.");
});


module.exports = db;
```

- require("sqlite3").verbose():
  Importa la biblioteca sqlite3 en modo detallado, lo que permite mensajes de error más explícitos.
- new sqlite3.Database(...):
  Abre o crea el archivo clientes.sqlite. Si hay algún error al conectarse, lo muestra por consola.
- console.log(...):
  Muestra un mensaje de confirmación si la conexión se establece correctamente.
- module.exports = db:
  Exporta la conexión a la base de datos para que otros archivos puedan usarla.

clientes.js

Este archivo define funciones para interactuar con la tabla de clientes en la base de datos. Se encarga de insertar nuevos usuarios y recuperar usuarios por su correo.

javascript

CopiarEditar

```
const db = require("./conexion (1)");


function insertarCliente(cliente, callback) {
    const { nombre, correo, contrasena } = cliente;
    const query = `INSERT INTO clientes (nombre, correo, contrasena) VALUES (?, ?, ?)`;
    db.run(query, [nombre, correo, contrasena], function (err) {
        if (err) return callback(err);
        callback(null, this.lastID);
    });
}


function obtenerClientePorCorreo(correo, callback) {
    const query = `SELECT * FROM clientes WHERE correo = ?`;
    db.get(query, [correo], (err, row) => {
        if (err) return callback(err);
        callback(null, row);
    });
}


module.exports = { insertarCliente, obtenerClientePorCorreo };
```

- const db = require(...):
  Importa la conexión a la base de datos.
- insertarCliente(...):
  - Inserta un nuevo cliente en la base de datos.
  - Usa una consulta SQL parametrizada para evitar inyecciones de SQL.
  - callback(null, this.lastID) retorna el ID del nuevo registro insertado.
- obtenerClientePorCorreo(...):
  - Realiza una consulta para buscar un cliente por su correo electrónico.
  - Devuelve el resultado usando el callback.
- module.exports = { ... }:
  Exporta ambas funciones para usarlas en el servidor principal.

app (3).js

Este es el archivo principal del servidor Express. Aquí se manejan las rutas públicas y protegidas, se genera el token JWT, y se valida antes de acceder a rutas privadas.

javascript

CopiarEditar

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

const jwt = require("jsonwebtoken");

const bodyParser = require("body-parser");

const clientes = require("./clientes");

const app = express();
```

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

const claveSecreta = "mi_clave_secreta";

// Ruta para autenticación
app.post("/login", (req, res) => {
    const { correo, contrasena } = req.body;
    clientes.obtenerClientePorCorreo(correo, (err, cliente) => {
        if (err) return res.status(500).send("Error en el servidor.");
        if (!cliente || cliente.contrasena !== contrasena) {
            return res.status(401).send("Credenciales incorrectas.");
        }

        const token = jwt.sign({ id: cliente.id, correo: cliente.correo }, claveSecreta, {
            expiresIn: "1h",
        });

        res.json({ token });
    });
});

// Middleware para verificar token
function autenticarToken(req, res, next) {
    const authHeader = req.headers["authorization"];
    const token = authHeader && authHeader.split(" ")[1];

    if (!token) return res.sendStatus(401);
```

```
    jwt.verify(token, claveSecreta, (err, usuario) => {
        if (err) return res.sendStatus(403);
        req.usuario = usuario;
        next();
    });
}

// Ruta protegida
app.get("/protegido", autenticarToken, (req, res) => {
    res.send("Accediste a una ruta protegida.");
});

app.listen(3000, () => {
    console.log("Servidor iniciado en http://localhost:3000");
});
```

Autenticación con JWT:

- POST /login: Ruta para autenticarse.
    - Se extraen correo y contrasena del cuerpo de la petición.
    - Se busca al usuario en la base de datos.
    - Si las credenciales son correctas, se genera un token JWT con jwt.sign() que contiene:
        - id y correo como payload.
        - claveSecreta como firma.
        - Una expiración de 1 hora.
    - Se devuelve el token al cliente.

Middleware autenticarToken:

- Extrae el token del encabezado Authorization.
- Verifica si es válido con jwt.verify().
- Si es correcto, permite continuar a la ruta siguiente.

Ruta protegida:

- GET /protegido solo es accesible si se incluye un token válido.
- Responde con un mensaje de confirmación.

Servidor:

- app.listen(3000) inicia el servidor en el puerto 3000.
- Se usa bodyParser.json() para leer datos en formato JSON desde el cuerpo de las peticiones.

Funcionamiento General del Proyecto

1. Un usuario hace una solicitud POST a /login con su correo y contraseña.
2. Si son válidos, el servidor genera un token JWT y lo devuelve.
3. El usuario utiliza ese token para acceder a la ruta /protegido, incluyendo el token en el encabezado Authorization.
4. El middleware valida el token. Si es válido, se permite el acceso.