Autenticación Segura con JWT en Node.js

Este proyecto demuestra cómo implementar un sistema de autenticación de usuarios utilizando JSON Web Tokens (JWT) en una aplicación desarrollada con Node.js y Express, conectada a una base de datos SQLite. A través de un sistema de inicio de sesión, los usuarios obtienen un token que les permite acceder a rutas restringidas.

Archivo: conexion.js

Este script establece la conexión con la base de datos SQLite utilizada en la aplicación.

javascript

CopiarEditar

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


const db = new sqlite3.Database("./clientes.sqlite", (err) => {
    if (err) {
        console.error("Error al conectar a la base de datos:", err.message);
    } else {
        console.log("Conexión exitosa a la base de datos SQLite.");
    }
});


module.exports = db;
```

Descripción del código:

- require("sqlite3").verbose(): Importa SQLite en modo detallado, permitiendo ver más información sobre errores.
- new sqlite3.Database(...): Crea una nueva conexión a la base de datos clientes.sqlite.
- module.exports = db: Permite exportar la instancia de la base para usarla en otros archivos.

Archivo: clientes.js

Define funciones que interactúan con la base de datos, como agregar un nuevo usuario o buscar uno existente por correo.

javascript

CopiarEditar

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


function crearCliente(cliente, callback) {
    const { nombre, correo, contrasena } = cliente;
    const sql = "INSERT INTO clientes (nombre, correo, contrasena) VALUES (?, ?, ?)";


    db.run(sql, [nombre, correo, contrasena], function (err) {
        if (err) return callback(err);
        callback(null, this.lastID);
    });
}


function buscarClientePorCorreo(correo, callback) {
    const sql = "SELECT * FROM clientes WHERE correo = ?";


    db.get(sql, [correo], (err, fila) => {
        if (err) return callback(err);
        callback(null, fila);
    });
}


module.exports = { crearCliente, buscarClientePorCorreo };
```

Descripción:

- crearCliente: Inserta un nuevo registro en la tabla clientes.
- buscarClientePorCorreo: Consulta la base para obtener un cliente por su correo.
- Ambas funciones usan callbacks para manejar resultados o errores.
- Se usan sentencias SQL parametrizadas para evitar inyecciones.

Archivo: app.js

Es el punto central del servidor Express. Se encargará de:

- Definir rutas públicas y privadas.
- Generar y validar los tokens JWT.
- Verificar si el usuario está autenticado antes de permitir el acceso.

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 CLAVE_SECRETA = "mi_clave_secreta";

// Ruta para iniciar sesión
app.post("/login", (req, res) => {
    const { correo, contrasena } = req.body;

    clientes.buscarClientePorCorreo(correo, (err, usuario) => {
```

```
        if (err) return res.status(500).send("Error interno.");
        if (!usuario || usuario.contrasena !== contrasena) {
            return res.status(401).send("Credenciales inválidas.");
        }

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

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

// Middleware para validar JWT
function verificarToken(req, res, next) {
    const authHeader = req.headers["authorization"];
    const token = authHeader?.split(" ")[1];

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

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

```
// Ruta privada
app.get("/privado", verificarToken, (req, res) => {
   res.send("Has accedido a una ruta segura.");
});

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

Autenticación con JWT

- POST /login: Ruta donde los usuarios inician sesión.
    - Si el correo y la contraseña son válidos, se genera un token firmado con una clave secreta.
    - El token incluye información del usuario (id, correo) y una expiración (1 hora).
    - Se responde al cliente con el token JWT.

Middleware de validación

- Extrae el token del encabezado Authorization.
- Usa jwt.verify para comprobar su validez.
- Si el token es correcto, se permite el acceso a la siguiente ruta.

Ruta protegida

- GET /privado: Solo se puede acceder si se incluye un token válido.
- Si el middleware valida el token, la respuesta será un mensaje de confirmación.

Flujo general del sistema

1. El usuario envía un POST a /login con su correo y contraseña.

2. El servidor valida las credenciales y genera un JWT.
3. El cliente usa ese token para hacer peticiones a rutas privadas.
4. El middleware intercepta esas peticiones, valida el token, y permite o deniega el acceso.