app.js

Este archivo es el punto de entrada principal del servidor:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();
const conexion = require("./conexion");
const clientes = require("./clientes");

app.use(express.json());
app.use(clientes);

app.listen(3000, () => {
    console.log("Servidor escuchando en el puerto 3000");
});
```

¿Qué hace?

Crea una aplicación con Express.

Importa módulos para conectar con la base de datos (conexion.js) y rutas de clientes (clientes.js).

Usa middleware para leer JSON y luego monta las rutas.

Escucha peticiones en el puerto 3000.

conexion.js

Este archivo establece la conexión con SQLite:

```
const sqlite3 = require("sqlite3").verbose();
const db = new sqlite3.Database("clientes.sqlite");

module.exports = db;
```

¿Qué hace?

Usa el paquete sqlite3 para conectarse a la base de datos clientes.sqlite.

Exporta la conexión (db) para que otros archivos puedan usarla.

clientes.js

Este es el archivo principal de lógica de autenticación y operaciones CRUD:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const db = require("./conexion");
const ruta = express.Router();
```

Define rutas para:

POST /login

Autentica un cliente y devuelve un token JWT:

```
ruta.post("/login", (req, res) => {
    const { usuario, password } = req.body;

    db.get("SELECT * FROM clientes WHERE usuario = ? AND password = ?", [usuario, password], (err, fila) => {
        if (fila) {
            const token = jwt.sign({ usuario: usuario }, "miclave", { expiresIn: "1h" });
            res.json({ token: token });
        } else {
            res.status(401).json({ error: "Usuario o contraseña incorrecta" });
        }
    });
});
```

Middleware: verificarToken

Protege rutas privadas validando el JWT:

```
function verificarToken(req, res, next) {
    const token = req.headers["authorization"];
    if (!token) return res.status(403).json({ error: "Token requerido" });

    jwt.verify(token, "miclave", (err, decoded) => {
        if (err) return res.status(403).json({ error: "Token inválido" });
        req.usuario = decoded.usuario;
        next();
    });
}
```

GET /clientes (protegida)

Devuelve todos los clientes solo si el token es válido:

```
ruta.get("/clientes", verificarToken, (req, res) => {
    db.all("SELECT * FROM clientes", (err, filas) => {
        res.json(filas);
    });
});
```

En resumen

Componente	Función
app.js	Inicia el servidor y monta rutas
conexion.js	Conecta con base de datos SQLite
clientes.js	Maneja login, genera token, y rutas protegidas
JWT	Se usa para autenticar usuarios (sin sesiones)
Base de datos	Contiene usuarios con usuario y password