Escuela:

Facultad de Informática

Carrera:

Licenciatura en informática

Hecho por:

Gastelum Landeros Antonio

Grupo: 2-1

Código Explicado:

Primera parte app.js:

Importación de módulos

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

const jwt = require("jsonwebtoken");

const { sequelize } = require("./database");

const clientesRoutes = require("./routes/clientes");
```

- express: Framework web para construir la API.
- jsonwebtoken: Biblioteca para generar y verificar tokens JWT.
- sequelize: ORM que se conecta a una base de datos SQLite y gestiona modelos.
- clientesRoutes: Importa las rutas CRUD de clientes (protegidas con JWT).

Inicialización de la aplicación

```
const app = express();

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

- app: crea la instancia principal de la aplicación Express.
- express.json(): middleware que permite interpretar cuerpos de solicitud en formato JSON (requerido para POST, PUT, etc.).

Ruta de login simulada

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

 const { usuario, password } = req.body;

 if (usuario === "admin" && password === "1234") {

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

  res.json({ token });

 } else {
```

```
    res.status(401).json({ mensaje: "Credenciales inválidas" });

  }

});
```

¿Qué hace?

- Recibe un usuario y password por POST.
- Verifica que sean "admin" y "1234" (autenticación simulada).
- Si son válidos, crea un **token JWT** con duración de 1 hora.
- Retorna ese token en formato JSON.
- Si son incorrectos, responde con error 401 Unauthorized.

JWT:

jwt.sign(payload, claveSecreta, opciones);

- payload: { usuario }
- clave secreta: "clave_secreta"
- expiresIn: "1h" → el token expira en 1 hora

CRUD de clientes (protegido con JWT)

app.use("/clientes", clientesRoutes);

- Aquí se enlazan todas las rutas CRUD que están en el archivo routes/clientes.js.
- Estas rutas están protegidas por un middleware de autenticación (verificarToken), por lo que solo se puede acceder con un token JWT válido.

Inicio del servidor y sincronización con la base de datos

```
sequelize.sync().then(() => {

  app.listen(3000, () => {

    console.log("Servidor corriendo en http://localhost:3000");

  });

});
```

¿Qué hace?

- sequelize.sync() crea las tablas necesarias si no existen (en este caso, la tabla Clientes).
- app.listen(3000) arranca el servidor web en el puerto 3000.

Segunda parte cliente.js:

Importaciones

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

const router = express.Router();

const { Cliente } = require("../database");

const verificarToken = require("../middleware/auth");
```

- express.Router() permite modularizar rutas por secciones.
- Cliente es el modelo Sequelize que representa la tabla Clientes.
- verificarToken es un middleware que valida el token JWT en cada petición.

Crear un cliente

```
router.post("/", verificarToken, async (req, res) => {

  const nuevo = await Cliente.create(req.body);

  res.json(nuevo);

});
```

- Ruta: POST /clientes
- Crea un nuevo cliente con los datos enviados en req.body.
- Requiere token JWT para poder acceder.

Obtener todos los clientes

```
router.get("/", verificarToken, async (req, res) => {

  const lista = await Cliente.findAll();

  res.json(lista);

});
```

- Ruta: GET /clientes
- Devuelve todos los clientes registrados en la base de datos.

Obtener un cliente por ID

```
router.get("/:id", verificarToken, async (req, res) => {

  const cliente = await Cliente.findByPk(req.params.id);

  if (!cliente) return res.status(404).json({ mensaje: "No encontrado" });

  res.json(cliente);

});
```

- Ruta: GET /clientes/:id
- Busca un cliente por su ID.
- Si no lo encuentra, responde con error 404.

Actualizar un cliente

```
router.put("/:id", verificarToken, async (req, res) => {

  const cliente = await Cliente.findByPk(req.params.id);

  if (!cliente) return res.status(404).json({ mensaje: "No encontrado" });

  await cliente.update(req.body);

  res.json(cliente);

});
```

- Ruta: PUT /clientes/:id
- Busca al cliente por ID.
- Si existe, actualiza sus datos con lo recibido en req.body.

Eliminar un cliente

```
router.delete("/:id", verificarToken, async (req, res) => {

  const cliente = await Cliente.findByPk(req.params.id);

  if (!cliente) return res.status(404).json({ mensaje: "No encontrado" });
```

```
  await cliente.destroy();

  res.json({ mensaje: "Cliente eliminado" });

});
```

- Ruta: DELETE /clientes/:id
- Busca al cliente, y si existe, lo elimina de la base de datos.