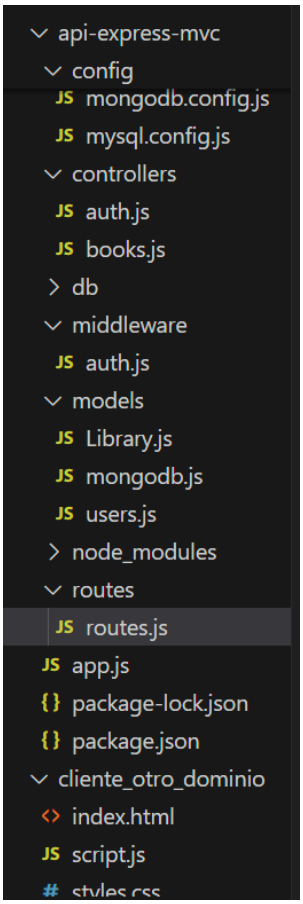
1. Parts de l’arquitectura de l’aplicació

L'aplicació es divideix en tres parts principals:

Backend:
Servidor Express amb rutes, controladors i models.

```
const express = require("express");
const app = express();
app.use(express.json());
app.use("/api", require("./routes/routes"));
app.listen(5000, () => {
    console.log("Servidor corriendo en http://localhost:5000");
});
```

- **Controladors i Models:**
  Gestió de la lògica d'autenticació i operacions CRUD (p.ex. auth.js, books.js, mongodb.js).
- **Frontend:**
  Pàgina HTML i script per interactuar amb l'API i gestionar la interfície (index.html i script.js).



---

2. Adaptació del model de la llibreria de MySQL a Mongo

En comptes d'utilitzar MySQL, s'ha creat un mòdul per connectar amb MongoDB. Això implica canviar la configuració i les operacions de consulta.

```
const { MongoClient } = require('mongodb');
const uri =
"mongodb+srv://user:1234@cluster0.jyfuu.mongodb.net/book?retryWrites
=true&w=majority&tls=true";
const client = new MongoClient(uri, { useNewUrlParser: true,
useUnifiedTopology: true, tls: true });

let db;

async function connectToMongoDB() {
    if (!db) {
        await client.connect();
        db = client.db('book');
        console.log('Conexión exitosa a MongoDB Atlas');
    }
    return db;
}

module.exports = { connectToMongoDB };
```

Aquest mòdul substitueix les consultes SQL per operacions amb el client de MongoDB.

---

3. Funcionalitat completa usant Mongo (CRUD de llibres)

Obtener Libros

```
async function getBooks(req, res) {
    const books = await listAll();
    res.status(200).json(books);
}
```

Crear

```
async function addBook(req, res) {
```

```
    const { title, author, year } = req.body;
    await create({ title, author, year });
    res.status(201).json({ message: "Libro creado exitosamente" });
}
```

Modificar

```
async function updateBook(req, res) {
    const { id } = req.params;
    const { title, author, year } = req.body;
    const success = await update(id, { title, author, year });
    res.status(success ? 200 : 404).json({ message: success ? "Libro
actualizado" : "Libro no encontrado" });
}
```

Eliminar

```
async function deleteBookById(req, res) {
    const { id } = req.params;
    const success = await deleteBook(id);
    res.status(success ? 200 : 404).json({ message: success ? "Libro
eliminado" : "Libro no encontrado" });
}
```

Aquests fragments mostren com s'implementa el CRUD amb MongoDB al backend.

---

4. Canvis per implementar l'autenticació JWT (Backend i Frontend)

Backend:
S'ha creat un mòdul per generar i verificar el token JWT i un middleware per protegir rutes.

```
const jwt = require("jsonwebtoken");
const SECRET_KEY = "supersecreto";

function generateToken(user) {
    return jwt.sign(user, SECRET_KEY, { expiresIn: "1h" });
}
```

Middleware

```
function authenticateToken(req, res, next) {
    const authHeader = req.headers["authorization"];
    const token = authHeader && authHeader.split(" ")[1];
    if (!token) return res.status(401).json({ error: "Token no
proporcionado" });
    jwt.verify(token, SECRET_KEY, (err, user) => {
        if (err) return res.status(403).json({ error: "Token
inválido" });
        req.user = user;
        next();
    });
}

module.exports = { generateToken, authenticateToken };
```

- Les rutes protegides (afegir, modificar o eliminar llibres) utilitzen aquest middleware.

Frontend:
S'ha afegit gestió del token en el navegador, emmagatzemat en el localStorage, i s'envia en les capçaleres de les peticions a l'API.

```
function saveToken(token) {
    localStorage.setItem("jwtToken", token);
}

function getToken() {
    return localStorage.getItem("jwtToken");
}

fetch(`${API_URL}/books`, {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${getToken()}`
    },
    body: JSON.stringify({ title, author, year }),
});
```

- D'aquesta manera, el frontend gestiona l'autenticació i envia el token en les sol·licituds protegides.