



Universidad

Universidad Autónoma de Sinaloa

Carrera

Lic. en Informática

Materia

Desarrollo web del lado del servidor

Actividad

Actividad final: CRUD completo

Grupo

2-3

Fecha

08/06/2025 Culiacán, Sinaloa

Maestro

José Manuel Cazarez Alderete

Alumna

Núñez Sarabia Jessica Anahí



ÍNDICE

03

DB.JS

05

INDEX.JS

07

LIBROS.JS

10

BIBLIOTECAAPI.JS

13

APP.JSX

DB.JS

Este archivo configura la base de datos SQLite y define la tabla libros, que almacena los registros de los préstamos en la biblioteca escolar.

Configuración de SQLite

```
const sqlite3 = require('sqlite3').verbose();  
const db = new sqlite3.Database('./biblioteca.sqlite');
```

Crea o abre la base de datos biblioteca.sqlite, permitiendo el almacenamiento persistente de datos.

Creación de la tabla libros

```
db.serialize(() => {  
  db.run(CREATE TABLE IF NOT EXISTS libros (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    alumno TEXT NOT NULL,  
    titulo TEXT NOT NULL,  
    autor TEXT NOT NULL,  
    grupo TEXT NOT NULL  
  ));  
});
```

id: Identificador único, PRIMARY KEY AUTOINCREMENT (se genera automáticamente).

alumno: Guarda el nombre del alumno que toma prestado el libro (TEXT NOT NULL asegura que no esté vacío).

titulo: Título del libro solicitado.

autor: Nombre del autor del libro.

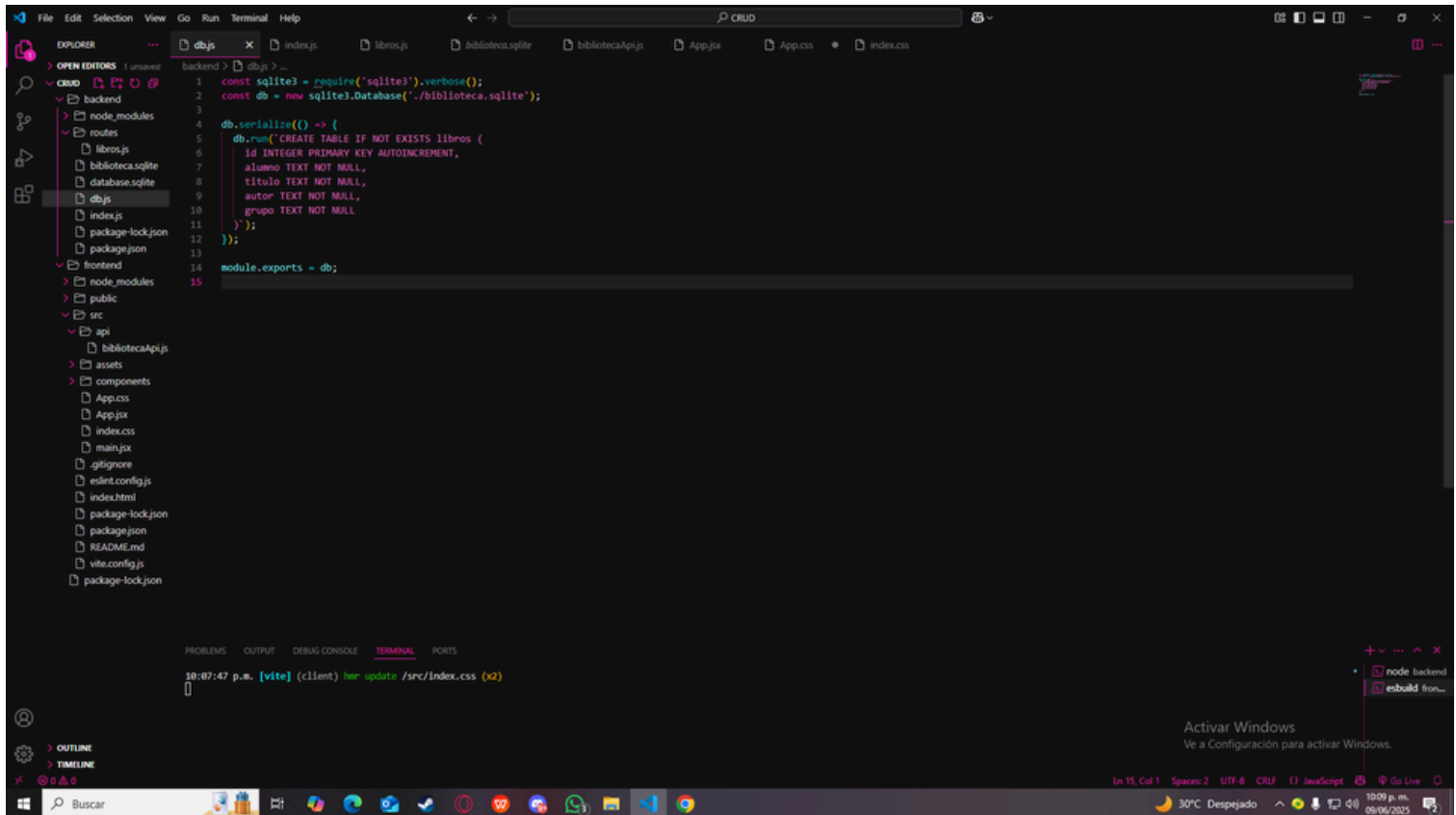
grupo: Grupo al que pertenece el alumno (útil para organización escolar).

Exportación de db

```
module.exports = db;
```

ase que db esté disponible en otros archivos (index.js, libros.js).

DB.JS



The screenshot shows a VS Code editor interface with a project structure on the left and a code editor in the center. The project structure includes a 'backend' folder with 'db.js', 'index.js', and 'libros.js', and a 'frontend' folder with 'App.js', 'App.jsx', 'index.css', 'main.jsx', and 'vite.config.js'. The 'db.js' file is open in the editor, showing the following code:

```
1 const sqlite3 = require('sqlite3').verbose();
2 const db = new sqlite3.Database('./Biblioteca.sqlite');
3
4 db.serialize(() => {
5   db.run('CREATE TABLE IF NOT EXISTS libros (
6     id INTEGER PRIMARY KEY AUTOINCREMENT,
7     alumno TEXT NOT NULL,
8     titulo TEXT NOT NULL,
9     autor TEXT NOT NULL,
10    grupo TEXT NOT NULL
11  );');
12 });
13
14 module.exports = db;
```

The terminal at the bottom shows the command 'npm update /src/index.css (x2)' being executed. The status bar at the bottom indicates the file is 'Ln 15, Col 1' and the encoding is 'UTF-8'.

INDEX.JS

Este archivo configura la base de datos SQLite y define la tabla libros, que almacena los registros de los préstamos en la biblioteca escolar.

Creación de la tabla libros

```
const librosRoutes = require('./routes/libros');
```

Carga las rutas CRUD desde libros.js

Configuración de middleware

```
app.use(cors());  
app.use(express.json());  
app.use('/api/libros', librosRoutes);
```

`cors()`: Permite que el frontend (React) haga peticiones al backend sin restricciones de origen.

`express.json()`: Habilita soporte para datos en formato JSON, asegurando que `req.body` funcione correctamente.

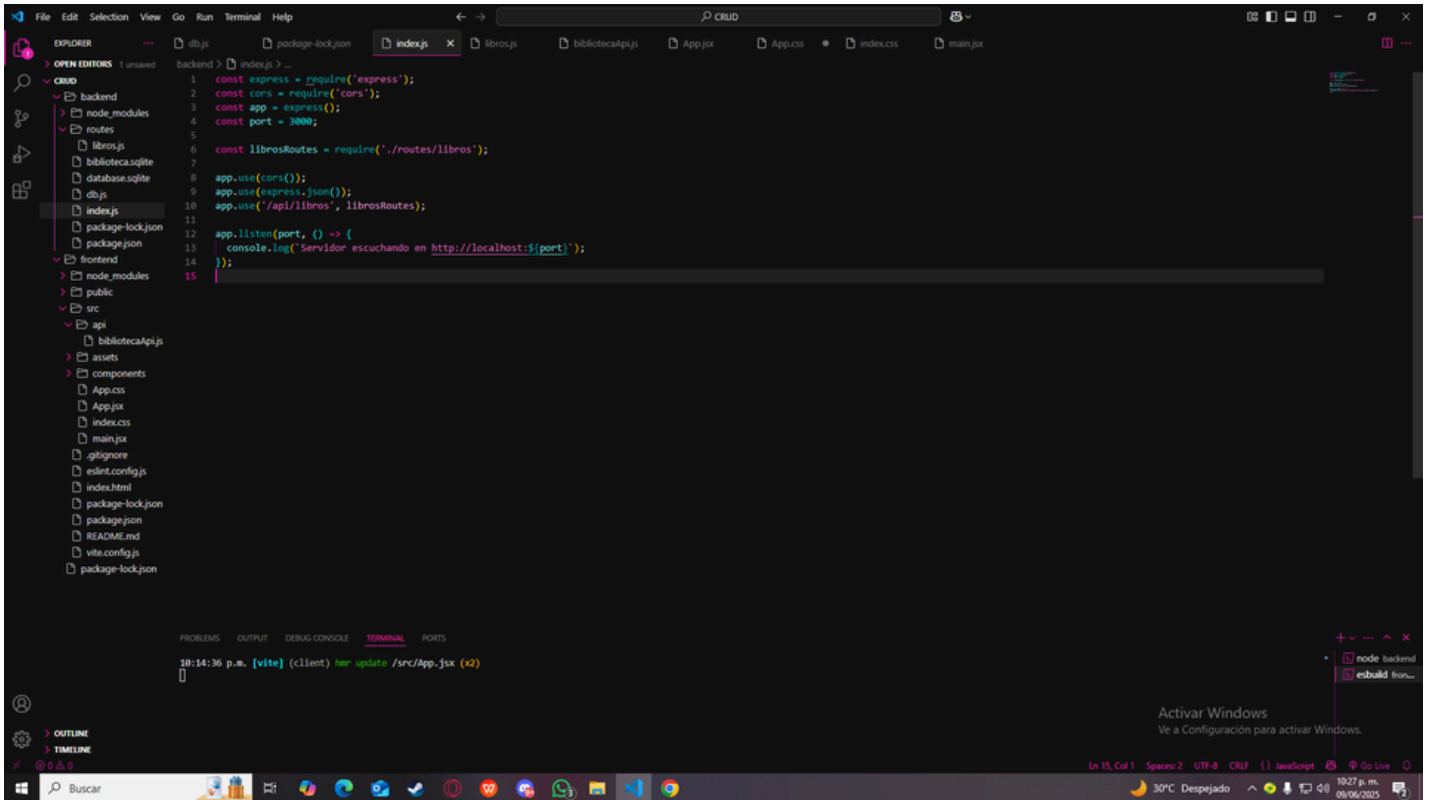
`app.use('/api/libros', librosRoutes)`: Define el prefijo `/api/libros`, evitando conflictos con otras rutas.

Inicio del servidor

```
app.listen(port, () => {  
  console.log(`Servidor escuchando en http://localhost:${port}`);  
});
```

Inicia el servidor en el puerto 3000

INDEX.JS



```
1 const express = require('express');
2 const cors = require('cors');
3 const app = express();
4 const port = 3000;
5
6 const librosRoutes = require('./routes/libros');
7
8 app.use(cors());
9 app.use(express.json());
10 app.use('/api/libros', librosRoutes);
11
12 app.listen(port, () => {
13   console.log('Servidor escuchando en http://localhost:3000');
14 });
```

10:14:36 p.m. [vite] (client) her update /src/App.jsx (x2)

LIBROS.JS

Obtener todos los libros (GET /api/libros)

```
router.get('/', (req, res) => {  
  db.all("SELECT * FROM libros", (err, rows) => {  
    if (err) return res.status(500).json({ error: err.message });  
    res.json(rows);  
  });  
});
```

Usa `db.all()` para obtener todos los registros de la tabla libros.
Manejo de errores: Devuelve un status 500 en caso de fallo, asegurando que el servidor responda correctamente.

Obtener un libro por ID (GET /api/libros/:id)

```
router.get('/:id', (req, res) => {  
  db.get("SELECT * FROM libros WHERE id = ?", [req.params.id], (err,  
row) => {  
    if (err) return res.status(500).json({ error: err.message });  
    res.json(row);  
  });  
});
```

Usa `db.get()` para recuperar un único registro por id.
Uso de `req.params.id`: Extrae el ID desde la URL, permitiendo búsquedas dinámicas.

Agregar un nuevo libro (POST /api/libros)

```
router.post('/', (req, res) => {  
  const { alumno, titulo, autor, grupo } = req.body;  
  db.run("INSERT INTO libros (alumno, titulo, autor, grupo) VALUES (?,  
?, ?, ?)",  
    [alumno, titulo, autor, grupo],  
    function (err) {  
      if (err) return res.status(500).json({ error: err.message });  
      res.json({ id: this.lastID, alumno, titulo, autor, grupo });  
    });  
});
```

LIBROS.JS

```
r});
```

db.run() con INSERT INTO para almacenar registros en la base de datos.

this.lastID: Devuelve el ID del nuevo registro, asegurando que el frontend pueda manejarlo fácilmente.

Actualizar un libro (PUT /api/libros/:id)

```
router.put('/:id', (req, res) => {  
  const { alumno, titulo, autor, grupo } = req.body;  
  db.run("UPDATE libros SET alumno = ?, titulo = ?, autor = ?, grupo = ?  
WHERE id = ?",  
    [alumno, titulo, autor, grupo, req.params.id],  
    function (err) {  
      if (err) return res.status(500).json({ error: err.message });  
      res.json({ mensaje: "Libro actualizado correctamente" });  
    });  
});
```

Usa UPDATE libros SET ... WHERE id = ? para actualizar registros existentes.

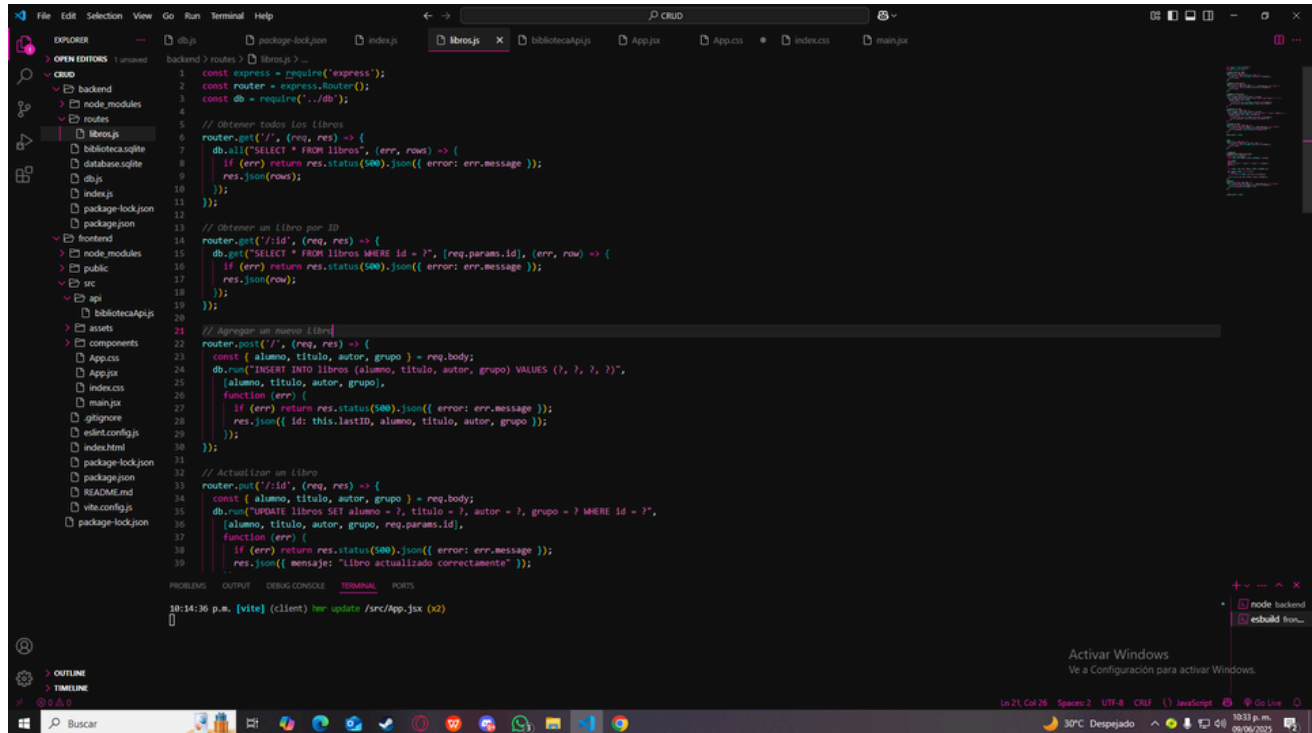
Uso de req.body: Recibe datos en JSON, asegurando que el frontend pueda enviar actualizaciones correctamente.

Eliminar un libro (DELETE /api/libros/:id)

```
router.delete('/:id', (req, res) => {  
  db.run("DELETE FROM libros WHERE id = ?", [req.params.id], function  
(err) {  
    if (err) return res.status(500).json({ error: err.message });  
    res.json({ mensaje: "Libro eliminado correctamente" });  
  });  
});
```

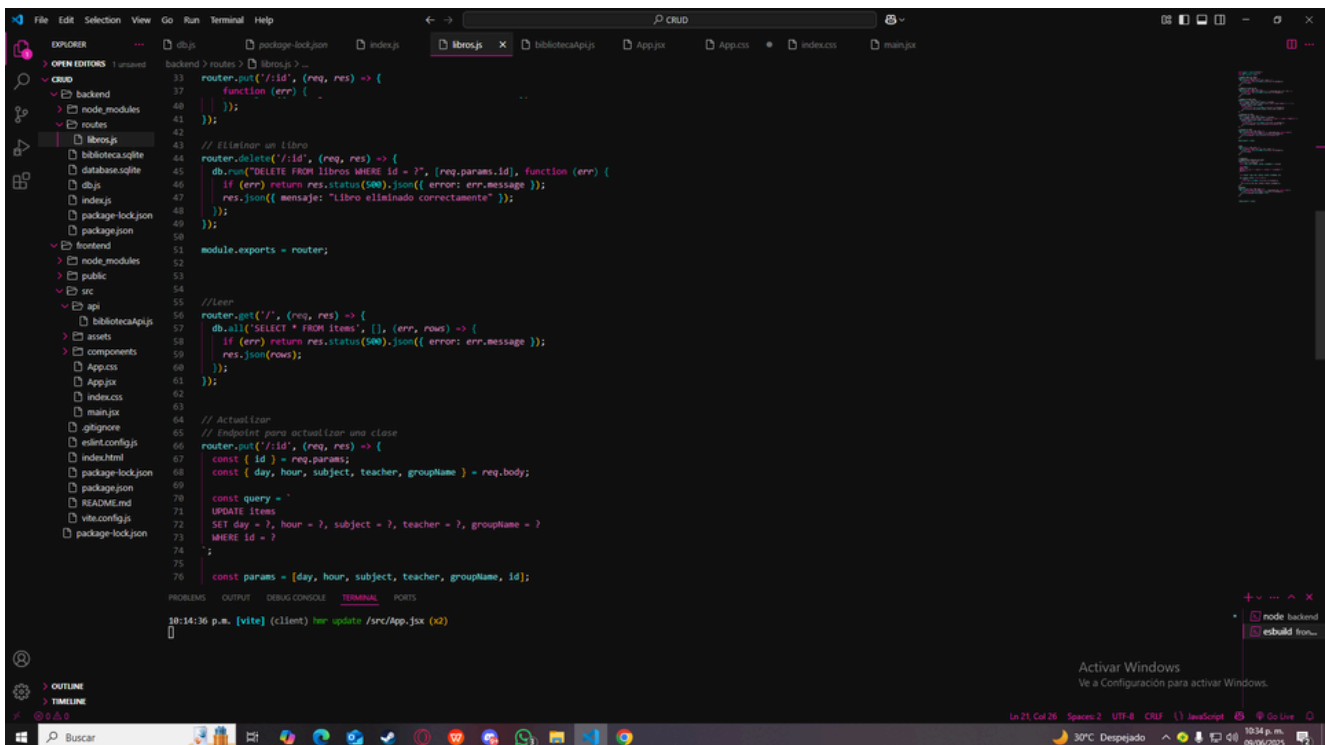
Usa DELETE FROM libros WHERE id = ? para borrar un libro.
Devuelve un mensaje "Libro eliminado correctamente" tras la acción.

LIBROS.JS



```
1 const express = require('express');
2 const router = express.Router();
3 const db = require('../db');
4
5 // Obtener todas las libros
6 router.get('/', (req, res) => {
7   db.all("SELECT * FROM libros", (err, rows) => {
8     if (err) return res.status(500).json({ error: err.message });
9     res.json(rows);
10   });
11 });
12
13 // Obtener un libro por ID
14 router.get('/:id', (req, res) => {
15   db.get("SELECT * FROM libros WHERE id = ?", [req.params.id], (err, row) => {
16     if (err) return res.status(500).json({ error: err.message });
17     res.json(row);
18   });
19 });
20
21 // Agregar un nuevo libro
22 router.post('/', (req, res) => {
23   const { alumno, titulo, autor, grupo } = req.body;
24   db.run("INSERT INTO libros (alumno, titulo, autor, grupo) VALUES (?, ?, ?, ?)",
25     [alumno, titulo, autor, grupo],
26     function (err) {
27       if (err) return res.status(500).json({ error: err.message });
28       res.json({ id: this.lastID, alumno, titulo, autor, grupo });
29     });
30 });
31
32 // Actualizar un libro
33 router.put('/:id', (req, res) => {
34   const { alumno, titulo, autor, grupo } = req.body;
35   db.run("UPDATE libros SET alumno = ?, titulo = ?, autor = ?, grupo = ? WHERE id = ?",
36     [alumno, titulo, autor, grupo, req.params.id],
37     function (err) {
38       if (err) return res.status(500).json({ error: err.message });
39       res.json({ mensaje: "Libro actualizado correctamente" });
40     });
41 });
```

18:14:36 p.m. [vite] (client) her update /src/App.jsx (x2)



```
33 router.put('/:id', (req, res) => {
34   function (err) {
35     // ...
36   }
37 });
38
39 // Eliminar un libro
40 router.delete('/:id', (req, res) => {
41   db.run("DELETE FROM libros WHERE id = ?", [req.params.id], function (err) {
42     if (err) return res.status(500).json({ error: err.message });
43     res.json({ mensaje: "Libro eliminado correctamente" });
44   });
45 });
46
47 module.exports = router;
48
49 // Leer
50 router.get('/', (req, res) => {
51   db.all("SELECT * FROM items", [], (err, rows) => {
52     if (err) return res.status(500).json({ error: err.message });
53     res.json(rows);
54   });
55 });
56
57 // Actualizar
58 // Endpoint para actualizar una clase
59 router.put('/:id', (req, res) => {
60   const { id } = req.params;
61   const { day, hour, subject, teacher, groupName } = req.body;
62
63   const query = `
64     UPDATE items
65     SET day = ?, hour = ?, subject = ?, teacher = ?, groupName = ?
66     WHERE id = ?
67   `;
68
69   const params = [day, hour, subject, teacher, groupName, id];
70
71   db.run(query, params, function (err) {
72     if (err) return res.status(500).json({ error: err.message });
73     res.json({ mensaje: "Clase actualizada correctamente" });
74   });
75 });
```

18:14:36 p.m. [vite] (client) her update /src/App.jsx (x2)

BIBLIOTECAAPI.JS

Obtener todos los libros (GET /api/libros)

```
export async function getLibros() {  
  const res = await fetch(API_URL);  
  return res.json();  
}
```

Solicita al backend la lista de libros con `fetch()`.
Convierte la respuesta JSON para usarla en el frontend (`res.json()`).
Uso de `async/await`: Asegura que la función espere la respuesta antes de continuar.

Crear un nuevo libro (POST /api/libros)

```
export async function createLibro(data) {  
  const res = await fetch(API_URL, {  
    method: 'POST',  
    headers: { 'Content-Type': 'application/json' },  
    body: JSON.stringify(data),  
  });  
  return res.json();  
}
```

`method: 'POST'`: Envía datos nuevos al servidor.
`headers: { 'Content-Type': 'application/json' }`: Define el formato para la solicitud.
`JSON.stringify(data)`: Convierte los datos del formulario en texto JSON para enviarlos.

Actualizar un libro (PUT /api/libros/:id)

```
export async function updateLibro(id, data) {  
  const res = await fetch(`${API_URL}/${id}`, {  
    method: 'PUT',  
    headers: { 'Content-Type': 'application/json' },  
    body: JSON.stringify(data),  
  });  
  return res.json();  
}
```

BIBLIOTECAAPI.JS

```
}
```

PUT: Modifica un registro existente.

id en la URL: Permite actualizar un libro específico.

body: JSON.stringify(data): Envía los datos modificados al backend.

Eliminar un libro (DELETE /api/libros/:id)

```
export async function deleteLibro(id) {  
  const res = await fetch( `${API_URL}/${id}`, {  
    method: 'DELETE',  
  });  
  return res.json();  
}
```

DELETE: Solicita la eliminación de un libro por id.

No necesita body, solo el id en la URL.

Devuelve la respuesta JSON del servidor.

BIBLIOTECA API.JS

```
File Edit Selection View Go Run Terminal Help
bibliotecaApi.js x App.jsx App.css index.css
bibliotecaApi.js x
1 const API_URL = '/api/libros';
2
3 export async function getLibros() {
4   const res = await fetch(API_URL);
5   return res.json();
6 }
7
8 export async function createLibro(data) {
9   const res = await fetch(API_URL, {
10     method: 'POST',
11     headers: { 'Content-Type': 'application/json' },
12     body: JSON.stringify(data),
13   });
14   return res.json();
15 }
16
17 export async function updateLibro(id, data) {
18   const res = await fetch(`${API_URL}/${id}`, {
19     method: 'PUT',
20     headers: { 'Content-Type': 'application/json' },
21     body: JSON.stringify(data),
22   });
23   return res.json();
24 }
25
26 export async function deleteLibro(id) {
27   const res = await fetch(`${API_URL}/${id}`, {
28     method: 'DELETE',
29   });
30   return res.json();
31 }
32
33 export async function deleteClass(id) {
34   const res = await fetch(`${API_URL}/${id}`, {
35     method: 'DELETE',
36   });
37   return res.json();
38 }
39
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
10:07:47 p.m. [vite] (client) her update /src/index.css (x2)
Activar Windows
Ve a Configuración para activar Windows.
Ln 32, Col 1 Spaces: 2 UTF-8 CR LF JavaScript Go Live
30°C Despejado 10:10 p.m. 06/06/2023
```

APP.JSX

Carga de datos desde el backend

```
useEffect(() => {  
  const loadData = async () => {  
    const data = await getLibros();  
    setLibros(data);  
  };  
  loadData();  
}, [reload]);
```

Obtiene los libros desde el backend y los muestra en la interfaz. Se ejecuta cada vez que reload cambia, asegurando datos actualizados.

Manejo del formulario (useState y handleChange)

```
const [form, setForm] = useState({ alumno: "", titulo: "", autor: "", grupo: ""  
});  
  
const handleChange = (e) => {  
  setForm({ ...form, [e.target.name]: e.target.value });  
};
```

Guarda los datos ingresados por el usuario en el formulario, permitiendo actualizarlos dinámicamente sin recargar la página.

Crear o actualizar un libro (handleSave)

```
const handleSave = async () => {  
  if (!form.alumno || !form.titulo || !form.autor || !form.grupo) {  
    alert("Todos los campos son obligatorios.");  
    return;  
  }  
  if (form.id) {  
    await updateLibro(form.id, form);  
  } else {  
    await createLibro(form);  
  }  
  setReload(!reload);  
  setForm({ alumno: "", titulo: "", autor: "", grupo: "" });  
};
```

APP.JSX

Evita que se registren libros sin datos completos.

Si el libro ya tiene un id, lo actualiza, si no, lo crea como nuevo.

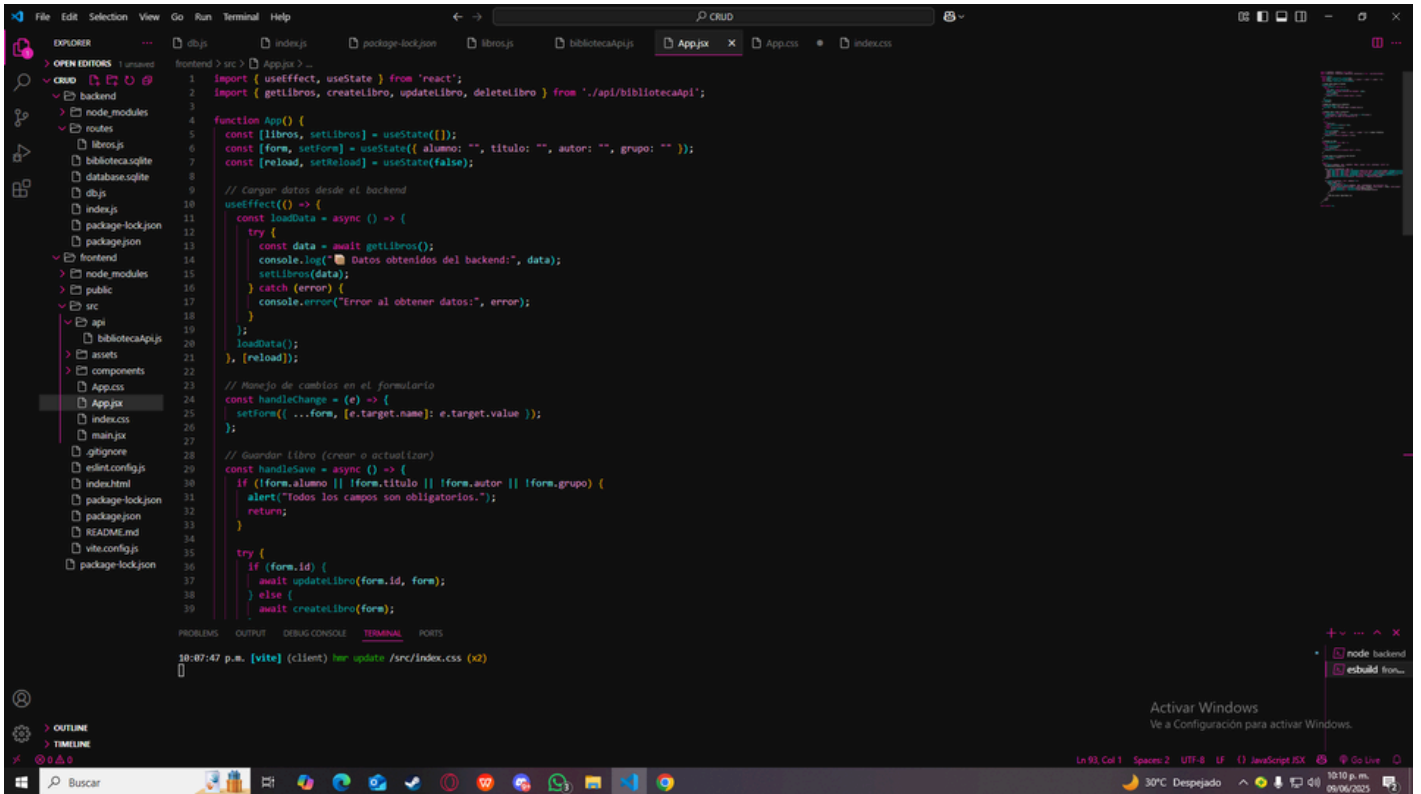
setReload(!reload): Forza la recarga para actualizar la lista en pantalla.

Eliminar un libro (handleDelete)

```
const handleDelete = async (id) => {  
  await deleteLibro(id);  
  setReload(!reload);  
};
```

Llama al backend para eliminar un registro y luego actualiza la interfaz sin necesidad de recargar manualmente.

APP.JSX



```
1 import { useEffect, useState } from 'react';
2 import { getLibros, createLibro, updateLibro, deleteLibro } from './api/bibliotecaApi';
3
4 function App() {
5   const [libros, setLibros] = useState([]);
6   const [form, setForm] = useState({ alumno: '', titulo: '', autor: '', grupo: '' });
7   const [reload, setReload] = useState(false);
8
9   // Cargar datos desde el backend
10  useEffect(() => {
11    const loadData = async () => {
12      try {
13        const data = await getLibros();
14        console.log("Datos obtenidos del backend:", data);
15        setLibros(data);
16      } catch (error) {
17        console.error("Error al obtener datos:", error);
18      }
19    };
20    loadData();
21    setReload(true);
22  }, [reload]);
23
24  // Manejo de cambios en el formulario
25  const handleChange = (e) => {
26    setForm({ ...form, [e.target.name]: e.target.value });
27  };
28
29  // Guardar libro (crear o actualizar)
30  const handleSubmit = async () => {
31    if (!form.alumno || !form.titulo || !form.autor || !form.grupo) {
32      alert("Todos los campos son obligatorios.");
33      return;
34    }
35
36    try {
37      if (form.id) {
38        await updateLibro(form.id, form);
39      } else {
40        await createLibro(form);
41      }
42    } catch (error) {
43      console.error("Error al guardar libro:", error);
44    }
45  };
46
47  return (
48    <div>
49      <h2>CRUD de Libros</h2>
50      <div>
51        <input type="text" value={form.alumno} />
52        <input type="text" value={form.titulo} />
53        <input type="text" value={form.autor} />
54        <input type="text" value={form.grupo} />
55      </div>
56      <button type="button" value="Guardar" />
57      <div>
58        {libros.map((libro) => (
59          <div>
60            <span>{libro.id}</span>
61            <span>{libro.alumno}</span>
62            <span>{libro.titulo}</span>
63            <span>{libro.autor}</span>
64            <span>{libro.grupo}</span>
65            <span>{libro.fecha}</span>
66            <span>{libro.estado}</span>
67            <span>{libro.descripcion}</span>
68          </div>
69        ))}
70      </div>
71    </div>
72  );
73}
```

18:07:47 p.m. [vite] (client) hmr update /src/index.css (x2)