Universidad Tecnológica Metropolitana

Estructuras de Datos Aplicadas

ISC. Ruth Betsaida Martínez Domínguez, MGTI

Práctica 2. Árboles binarios

- Soberanis Acosta Jimena Monserrat

Desarrollo de Software Multiplataforma

Cuarto Cuatrimestre

4°B

Parcial III

Miércoles, 22 de noviembre de 2023

Para esta práctica comencé creando mi carpeta, descargué sqlite como archivo y usando las dependencias de sqlite y express con los siguientes comandos:

- npm install sqlite3
- npm install express sqlite3

Proseguí creando el archivo app-server.js para la base de datos con sqlite3, en donde también creo la tabla llamada Camion que se creará solo sino existe, además le creé el primer registro de la tabla.

```
const db = new sqlite3.Database('refaccionaria.db');

// Crear la tabla "Camion" si no existe
db.run(`
    CREATE TABLE IF NOT EXISTS Camion (
        Idcamion INTEGER PRIMARY KEY,
        Nombre TEXT,
        Totalmacenaje REAL,
        Placas TEXT,
        Marca TEXT
    )
`, (err) => {
    if (err) {
        console.error(err.message);
        throw err;
    }

    console.log('Tabla "Camion" creada exitosamente.');

    // Insertar un ejemplo de dato
    db.run(`
        INSERT INTO Camion (Nombre, Totalmacenaje, Placas, Marca) VALUES ('Camion1', 10000, 'ABC123', 'Marca1')
    `);
});
```

Esa parte del código lo ejecuté una sola vez, por lo que lo comenté y seguí creando la clase CamionService que interactuará con la base de datos creada. Está primeramente el constructor que inicializa la conexión a la base de datos SQLite.

```
class CamionService {
    constructor() {
        this.db = new sqlite3.Database('refaccionaria.db', (err) => {
            if (err) {
                console.error(err.message);
                throw new Error('Error al abrir la base de datos');
            }
            console.log('Base de datos abierta correctamente.');
        });
    }
```

Existe el método getAllCamiones que sirve para obtener todos los registros de la tabla Camion, como bien se muestra con el select * from Camion

```
    getAllCamiones(callback) {
        this.db.all('SELECT * FROM Camion', (err, rows) => {
            if (err) {
                console.error(err.message);
                callback(err, null);
                return;
            }
            callback(null, rows);
        });
    }
```

Sigue el método de buscarPorPlaca, que como su nombre dice hará búsqueda entre la base de datos a través del número de placa, es con el select * from Camion where Placas = ? que realiza este select.

```
    buscarPorPlaca(placa, callback) {
        this.db.all('SELECT * FROM Camion WHERE Placas = ?', [placa], (err, rows) => {
            if (err) {
                console.error(err.message);
                callback(err, null);
                return;
            }
            callback(null, rows);
        });
    }
```

Por último, entre los métodos está closeDatabase que solo cierra la conexión que hay con la misma.

```
closeDatabase() {
    this.db.close((err) => {
        if (err) {
            console.error(err.message);
        } else {
            console.log('Conexión a la base de datos cerrada correctamente.');
        }
    });
}
```

Proseguí con la configuración con Express, primero llamé a los archivos de la carpeta ‘public’, y comencé con la creación de los Endpoints para mi proyecto:

Endpoint GET /camiones: se obtiene cada uno de los registros de la tabla camiones.

```
app.use(express.static('public'));

app.get('/camiones', (req, res) => {
    const camionService = new CamionService();

    camionService.getAllCamiones((err, rows) => {
        if (err) {
            res.status(500).json({ error: 'Error interno del servidor' });
            return;
        }

        res.json(rows);
        camionService.closeDatabase();
    });
});
```

Endpoint GET /camiones/placa/:placa: busca camiones a través del número de placa.

```
app.get('/camiones/placa/:placa', (req, res) => {
    const placa = req.params.placa;

    const camionService = new CamionService();

    camionService.buscarPorPlaca(placa, (err, rows) => {
        if (err) {
            res.status(500).json({ error: 'Error interno del servidor' });
            return;
        }

        res.json(rows);
        camionService.closeDatabase();
    });
});
```

Por último, en esta sección se encuentra la llamada al servidor en el puerto especificado antes.

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

Pero para que se pueda visualizar la base de datos con sus respectivos métodos con una interfaz adecuada, tuve que crear la carpeta 'public' que se llamó anteriormente, en esta carpeta se encuentran 3 archivos js, html y css.

Index.html

Tiene la estructura simple: un botón para la carga de los registros en la base de datos, un div con un input y un botón para la búsqueda de camiones por la placa, y por último, la tabla donde se visualiza cada botón.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Resultados de la Consulta</title>
</head>
<body>
    <h1>Resultados de la Consulta</h1>

    <button id="botonCargar">Cargar Datos</button>

    <div>
        <label for="inputPlaca">Buscar por Placa:</label>
        <input type="text" id="inputPlaca" placeholder="Ingrese la placa">
        <button id="botonBuscarPlaca">Buscar</button>
    </div>

    <table id="tablaCamiones">
        <tr>
            <th>Idcamion</th>
            <th>Nombre</th>
            <th>Totalmacenaje</th>
            <th>Placas</th>
            <th>Marca</th>
        </tr>
    </table>
    <script src="app.js"></script>
</body>
</html>
```

El archivo styles.css se encuentra en el repositorio de mi github

Por otro lado, se encuentra el archivo llamado app.js que se encarga de la interacción con el documento:

app.js

```javascript
document.addEventListener('DOMContentLoaded', function () {
    const tablaCamiones = document.getElementById('tablaCamiones');
    const botonCargar = document.getElementById('botonCargar');
    const inputPlaca = document.getElementById('inputPlaca');
    const botonBuscarPlaca = document.getElementById('botonBuscarPlaca');

```

Está la función cargarDatos, que comienza creando y llenando la tabla con cada registro de la base de datos (título y registros de la tabla).

```
async function cargarDatos() {
    try {
        const response = await fetch('/camiones');
        const data = await response.json();

        tablaCamiones.innerHTML = '';

        // Agregar fila de títulos
        const titulos = document.createElement('tr');
        titulos.innerHTML = `
            <th>Idcamion</th>
            <th>Nombre</th>
            <th>Totalmacenaje</th>
            <th>Placas</th>
            <th>Marca</th>
        `;
        tablaCamiones.appendChild(titulos);

        // Agregar filas de datos
        data.forEach(camion => {
            const fila = document.createElement('tr');
            fila.innerHTML = `
                <td>${camion.Idcamion}</td>
                <td>${camion.Nombre}</td>
                <td>${camion.Totalmacenaje}</td>
                <td>${camion.Placas}</td>
                <td>${camion.Marca}</td>
            `;
            tablaCamiones.appendChild(fila);
        });

        console.log('Datos cargados correctamente.');
    } catch (error) {
        console.error('Error al cargar los datos:', error);
    }
}
```

Para esta función se encuentra su respectivo evento de su botón que llama a esa función mencionada

```
// Evento clic en el botón Cargar
botonCargar.addEventListener('click', function () {
    cargarDatos();
});
```

De igual manera hice lo mismo con la función de buscarPorPlaca:

Se crea los títulos y los registros de la tabla con la base de datos.

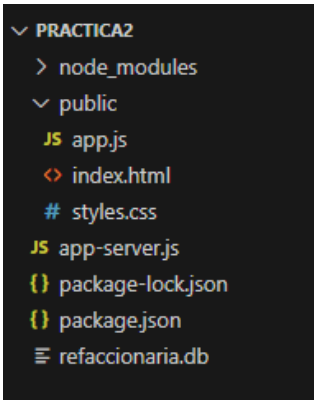
```
async function buscarPorPlaca(placa) {
    try {
        const response = await fetch(`/camiones/placa/${placa}`);
        const data = await response.json();

        tablaCamiones.innerHTML = '';

        // Agregar fila de títulos
        const titulos = document.createElement('tr');
        titulos.innerHTML = `
            <th>Idcamion</th>
            <th>Nombre</th>
            <th>Totalmacenaje</th>
            <th>Placas</th>
            <th>Marca</th>
        `;
        tablaCamiones.appendChild(titulos);
        // Agregar filas de datos
        data.forEach(camion => {
            const fila = document.createElement('tr');
            fila.innerHTML = `
                <td>${camion.Idcamion}</td>
                <td>${camion.Nombre}</td>
                <td>${camion.Totalmacenaje}</td>
                <td>${camion.Placas}</td>
                <td>${camion.Marca}</td>
            `;
            tablaCamiones.appendChild(fila);
        });

        console.log('Búsqueda por placa completada.');
    } catch (error) {
        console.error('Error al buscar por placa:', error);
    }
}
});
```

La organización de carpetas quedó de esta manera (node_modules son muchos archivos):



Una vez que terminé esto, pude lanzar la aplicación con node app-server.js y se ve así:

Al hacer click en el botón Cargar Datos:

Al buscar por placa: