Atividade: Desenvolvimento Full-Stack com CRUD em Node.js e SQLite3

Objetivo: Integrar o formulário frontend com um backend Node.js e banco de dados SQLite3, implementando operações CRUD completas (Create, Read, Update, Delete).

Passo a Passo Detalhado:

1. Configuração Inicial do Projeto

- **Crie um diretório para o projeto** e dentro dele:

```
pm init -y  # Inicialize um projeto Node.js
npm install express sqlite3 cors  # Instale as dependências
```

- Crie os arquivos:
  - `server.js` (servidor Node.js)
  - `database.js` (configuração do SQLite3)
  - Mantenha os arquivos frontend fornecidos (`index.html`, `styles.css`, `script.js`).

2. Configurar o Servidor Node.js (server.js)

```
const express = require('express');
const cors = require('cors');
const app = express();
app.use(express.json());
app.use(cors()); // Permite comunicação com o frontend
app.use(express.static('public')); // Serve arquivos estáticos (HTML, CSS, JS)

// Rotas serão definidas aqui

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});
```

3. Configurar o Banco de Dados SQLite3 (database.js)

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

// Cria a tabela 'usuarios' se não existir
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS usuarios (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
      nome TEXT NOT NULL,
      email TEXT NOT NULL UNIQUE,
      telefone TEXT NOT NULL
    )
  `);
});

module.exports = db;
```

4. Implementar Rotas CRUD no Backend (server.js)

Adicione as rotas após a configuração do servidor:

```
const db = require('./database.js');

// Cadastrar usuário (CREATE)
app.post('/cadastrar', (req, res) => {
  const { nome, email, telefone } = req.body;
  db.run(
    'INSERT INTO usuarios (nome, email, telefone) VALUES (?, ?, ?)',
    [nome, email, telefone],
    (err) => {
      if (err) return res.status(400).json({ error: err.message });
      res.json({ message: 'Usuário cadastrado!' });
    }
  );
});

// Atualizar usuário (UPDATE)
app.post('/atualizar', (req, res) => {
  const { nome, email, telefone } = req.body;
  db.run(
    'UPDATE usuarios SET email = ?, telefone = ? WHERE nome = ?',
    [email, telefone, nome],
    (err) => {
      if (err) return res.status(400).json({ error: err.message });
      res.json({ message: 'Usuário atualizado!' });
    }
  );
});

// Excluir usuário (DELETE)
app.post('/excluir', (req, res) => {
  const { nome } = req.body;
  db.run(
    'DELETE FROM usuarios WHERE nome = ?',
    [nome],
    (err) => {
      if (err) return res.status(400).json({ error: err.message });
```

```
      res.json({ message: 'Usuário excluído!' });
    }
  );
});
```

5. Corrigir o Frontend (script.js)

- **Remova a função** `cadastrar()` **duplicada** e garanta que a versão que faz a requisição `fetch` permaneça.
- **Modifique as funções** para limpar o formulário após ações:

```
// Exemplo na função cadastrar():
.then(data => {
  alert('Usuário cadastrado com sucesso!');
  document.getElementById('formulario').reset(); // Limpa os campos
})
```

6. Testar a Aplicação

- Inicie o servidor:

```
node server.js
```

- Acesse `http://localhost:3000` e teste:
  - Cadastre um usuário.
  - Atualize o telefone de um usuário existente.
  - Exclua um usuário pelo nome.

7. Desafio

- **Adicione uma funcionalidade de listagem de usuários:**
  - Crie uma rota `GET /usuarios` no backend.
  - Adicione um botão "Listar" no frontend que exiba os usuários em uma tabela.
- **Melhore a validação:**
  - Verifique no backend se o e-mail já está cadastrado antes de inserir.
  - Adicione validação de telefone (formato numérico).