README.md

Proyecto ExpertSoft: Sistema de Gestión Financiera para el Sector Eléctrico

README.md

Descripción del Sistema

Este proyecto es una solución integral para la gestión de información financiera proveniente de plataformas Fintech (Nequi y Daviplata) para el sector eléctrico en Colombia. La solución consiste en un backend (API REST) desarrollado con Node.js y Express.js, una base de datos relacional en PostgreSQL, y un frontend minimalista para la administración de clientes.

El sistema permite la normalización y estructuración de datos desorganizados, la carga masiva de transacciones, un CRUD completo para la gestión de clientes y la ejecución de consultas avanzadas para obtener insights financieros.

Tecnologías Utilizadas

Backend: Node.js, Express.js

Base de datos: PostgreSQL

Librerías: pg, csv-parser

Frontend: HTML, JavaScript, Bootstrap 5

Herramientas: Postman, Git

Explicación de la Normalización

El proceso de normalización se aplicó para transformar la información desorganizada del archivo Excel en un modelo de base de datos relacional robusto.

1FN (Primera Forma Normal): Se aseguró que cada columna contuviera un solo valor atómico, eliminando listas o grupos repetitivos.

2FN (Segunda Forma Normal): Se eliminaron las dependencias funcionales parciales. Se crearon tablas separadas para clients y platforms para que sus atributos dependieran únicamente de sus respectivas claves primarias.

3FN (Tercera Forma Normal): Se eliminaron las dependencias transitivas. Se crearon tablas para bills y transactions para evitar que atributos no clave dependieran de otros atributos no clave.

El resultado es un modelo relacional con las siguientes entidades: clients, platforms, bills y transactions.

Modelo Relacional

A continuación se muestra el modelo relacional diseñado.

Instrucciones para Ejecutar el Proyecto

Clonar el repositorio: git clone [URL\_DEL\_REPOSITORIO]

Configurar la base de datos:

Crear una base de datos PostgreSQL con el nombre pd\_tu\_nombre\_tu\_apellido\_clan.

Ejecutar el script SQL proporcionado (db\_script.sql) para crear la estructura de las tablas.

Instalar dependencias del backend:

Navegar a la carpeta del backend.

npm install

Configurar la conexión a la base de datos:

Editar el archivo server.js con las credenciales de tu base de datos (usuario, contraseña).

Ejecutar el backend:

node server.js

El servidor se ejecutará en http://localhost:3000.

Abrir el frontend:

Abrir el archivo index.html en tu navegador web.

Instrucciones para la Carga Masiva desde CSV

Asegurarse de que el archivo transactions.csv esté en la raíz del proyecto backend.

El script de carga masiva se ejecuta a través del endpoint POST /api/load-data.

Puedes usar Postman o cualquier cliente HTTP para realizar una petición a este endpoint.

Una vez ejecutado, el script insertará los datos del CSV en las tablas clients, platforms, bills y transactions, manejando automáticamente la creación de registros únicos.

Explicación de las Consultas Avanzadas

Total pagado por cada cliente: Endpoint GET /api/reports/total-paid-by-client. Esta consulta utiliza JOIN y GROUP BY para sumar el transaction\_amount de todas las transacciones, agrupando los resultados por el nombre del cliente. Es ideal para un control de ingresos global.

Facturas pendientes: Endpoint GET /api/reports/pending-bills. Esta consulta identifica facturas donde la suma de las transacciones asociadas es menor al bill\_amount. Utiliza LEFT JOIN para incluir facturas sin transacciones y HAVING para filtrar las que están parcialmente o no pagadas.

Transacciones por plataforma: Endpoint GET /api/reports/transactions-by-platform/:platformName. Esta consulta filtra las transacciones por una plataforma específica (ej. "Nequi"). Relaciona las tablas transactions, clients, platforms y bills para proporcionar una visión completa de quién pagó, a través de qué plataforma y a qué factura corresponde.

Datos del Desarrollador

Nombre: [Tu Nombre Completo]

Clan: [Tu Clan]

Correo: [Tu Correo Electrónico]

SQL

-- DDL para la base de datos y tablas del proyecto

-- Nombre de la base de datos: pd\_tu\_nombre\_tu\_apellido\_clan

-- Creamos la tabla 'platforms' para las plataformas Fintech

CREATE TABLE platforms (

platform\_id SERIAL PRIMARY KEY,

platform\_name VARCHAR(50) UNIQUE NOT NULL

);

-- Creamos la tabla 'clients' para los clientes

CREATE TABLE clients (

client\_id SERIAL PRIMARY KEY,

client\_name VARCHAR(100) NOT NULL,

client\_email VARCHAR(100) UNIQUE NOT NULL,

client\_phone VARCHAR(20) UNIQUE NOT NULL

);

-- Creamos la tabla 'bills' para las facturas

CREATE TABLE bills (

bill\_id SERIAL PRIMARY KEY,

client\_id INT NOT NULL,

bill\_amount DECIMAL(10, 2) NOT NULL,

bill\_date DATE NOT NULL,

is\_paid BOOLEAN DEFAULT FALSE,

FOREIGN KEY (client\_id) REFERENCES clients(client\_id)

);

-- Creamos la tabla 'transactions' para las transacciones

CREATE TABLE transactions (

transaction\_id SERIAL PRIMARY KEY,

bill\_id INT,

client\_id INT NOT NULL,

platform\_id INT NOT NULL,

transaction\_amount DECIMAL(10, 2) NOT NULL,

transaction\_date DATE NOT NULL,

FOREIGN KEY (bill\_id) REFERENCES bills(bill\_id),

FOREIGN KEY (client\_id) REFERENCES clients(client\_id),

FOREIGN KEY (platform\_id) REFERENCES platforms(platform\_id)

);

EJEMPLO ARCHIVOS.CSV

client\_name,client\_email,client\_phone,bill\_amount,bill\_date,platform\_name,transaction\_amount,transaction\_date

Juan Perez,juan.perez@email.com,3001234567,150000.00,2024-05-10,Nequi,50000.00,2024-05-10

Juan Perez,juan.perez@email.com,3001234567,150000.00,2024-05-10,Daviplata,100000.00,2024-05-11

Maria Lopez,maria.lopez@email.com,3017654321,250000.00,2024-05-12,Nequi,250000.00,2024-05-12

Pedro Gomez,pedro.gomez@email.com,3059876543,50000.00,2024-05-13,Daviplata,25000.00,2024-05-13

Pedro Gomez,pedro.gomez@email.com,3059876543,50000.00,2024-05-13,Daviplata,25000.00,2024-05-14

JAVASCRIPT

// CÓDIGO DEL SERVIDOR (se asume que las dependencias están instaladas)

const express = require('express');

const { Pool } = require('pg');

const fs = require('fs');

const path = require('path');

const csv = require('csv-parser');

const app = express();

const port = 3000;

app.use(express.json());

// Configuración de la base de datos

const pool = new Pool({

user: 'tu\_usuario', // <-- ¡ACTUALIZAR!

host: 'localhost',

database: 'pd\_tu\_nombre\_tu\_apellido\_clan', // <-- ¡ACTUALIZAR!

password: 'tu\_password', // <-- ¡ACTUALIZAR!

port: 5432,

});

// Endpoint para cargar datos masivamente desde CSV

app.post('/api/load-data', async (req, res) => {

try {

const filePath = path.join(\_\_dirname, 'transactions.csv');

const data = [];

fs.createReadStream(filePath)

.pipe(csv())

.on('data', (row) => data.push(row))

.on('end', async () => {

const client = await pool.connect();

try {

await client.query('BEGIN');

for (const row of data) {

let clientResult = await client.query('SELECT client\_id FROM clients WHERE client\_email = $1', [row.client\_email]);

if (clientResult.rows.length === 0) {

clientResult = await client.query('INSERT INTO clients (client\_name, client\_email, client\_phone) VALUES ($1, $2, $3) ON CONFLICT (client\_email) DO NOTHING RETURNING client\_id', [row.client\_name, row.client\_email, row.client\_phone]);

if (clientResult.rows.length === 0) {

clientResult = await client.query('SELECT client\_id FROM clients WHERE client\_email = $1', [row.client\_email]);

}

}

const clientId = clientResult.rows[0].client\_id;

let platformResult = await client.query('SELECT platform\_id FROM platforms WHERE platform\_name = $1', [row.platform\_name]);

if (platformResult.rows.length === 0) {

platformResult = await client.query('INSERT INTO platforms (platform\_name) VALUES ($1) ON CONFLICT (platform\_name) DO NOTHING RETURNING platform\_id', [row.platform\_name]);

if (platformResult.rows.length === 0) {

platformResult = await client.query('SELECT platform\_id FROM platforms WHERE platform\_name = $1', [row.platform\_name]);

}

}

const platformId = platformResult.rows[0].platform\_id;

let billResult = await client.query('SELECT bill\_id FROM bills WHERE client\_id = $1 AND bill\_amount = $2 AND bill\_date = $3', [clientId, row.bill\_amount, row.bill\_date]);

if (billResult.rows.length === 0) {

billResult = await client.query('INSERT INTO bills (client\_id, bill\_amount, bill\_date) VALUES ($1, $2, $3) RETURNING bill\_id', [clientId, row.bill\_amount, row.bill\_date]);

}

const billId = billResult.rows[0].bill\_id;

await client.query('INSERT INTO transactions (bill\_id, client\_id, platform\_id, transaction\_amount, transaction\_date) VALUES ($1, $2, $3, $4, $5)', [billId, clientId, platformId, row.transaction\_amount, row.transaction\_date]);

}

await client.query('COMMIT');

res.status(200).send('Carga masiva completada con éxito.');

} catch (error) {

await client.query('ROLLBACK');

throw error;

} finally {

client.release();

}

});

} catch (error) {

console.error('Error al cargar datos:', error);

res.status(500).send('Error al cargar datos.');

}

});

// --- CRUD para la entidad 'clients' ---

// ... (El código completo del CRUD y las consultas avanzadas está en la respuesta anterior) ...

HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dashboard de Clientes</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h2>Gestión de Clientes</h2>

<div class="card mb-4">

<div class="card-header">

Formulario de Cliente

</div>

<div class="card-body">

<form id="client-form">

<input type="hidden" id="client-id">

<div class="mb-3">

<label for="client-name" class="form-label">Nombre</label>

<input type="text" class="form-control" id="client-name" required>

</div>

<div class="mb-3">

<label for="client-email" class="form-label">Email</label>

<input type="email" class="form-control" id="client-email" required>

</div>

<div class="mb-3">

<label for="client-phone" class="form-label">Teléfono</label>

<input type="tel" class="form-control" id="client-phone" required>

</div>

<button type="submit" class="btn btn-primary" id="submit-btn">Crear Cliente</button>

<button type="button" class="btn btn-secondary" id="clear-btn">Limpiar</button>

</form>

</div>

</div>

<h3>Listado de Clientes</h3>

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Nombre</th>

<th>Email</th>

<th>Teléfono</th>

<th>Acciones</th>

</tr>

</thead>

<tbody id="clients-table-body">

</tbody>

</table>

</div>

<script>

// CÓDIGO JAVASCRIPT DEL FRONTEND (se incluye en la respuesta anterior)

const form = document.getElementById('client-form');

const clientIdInput = document.getElementById('client-id');

const clientNameInput = document.getElementById('client-name');

const clientEmailInput = document.getElementById('client-email');

const clientPhoneInput = document.getElementById('client-phone');

const submitBtn = document.getElementById('submit-btn');

const clearBtn = document.getElementById('clear-btn');

const clientsTableBody = document.getElementById('clients-table-body');

async function fetchClients() {

const response = await fetch('/api/clients');

const clients = await response.json();

clientsTableBody.innerHTML = '';

clients.forEach(client => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${client.client\_id}</td>

<td>${client.client\_name}</td>

<td>${client.client\_email}</td>

<td>${client.client\_phone}</td>

<td>

<button class="btn btn-sm btn-warning edit-btn" data-id="${client.client\_id}">Editar</button>

<button class="btn btn-sm btn-danger delete-btn" data-id="${client.client\_id}">Eliminar</button>

</td>

`;

clientsTableBody.appendChild(row);

});

}

form.addEventListener('submit', async (e) => {

e.preventDefault();

const id = clientIdInput.value;

const name = clientNameInput.value;

const email = clientEmailInput.value;

const phone = clientPhoneInput.value;

const method = id ? 'PUT' : 'POST';

const url = id ? `/api/clients/${id}` : '/api/clients';

await fetch(url, {

method: method,

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ name, email, phone }),

});

form.reset();

clientIdInput.value = '';

submitBtn.textContent = 'Crear Cliente';

fetchClients();

});

clientsTableBody.addEventListener('click', async (e) => {

if (e.target.classList.contains('edit-btn')) {

const id = e.target.dataset.id;

const response = await fetch(`/api/clients/${id}`);

const client = (await response.json())[0];

clientIdInput.value = client.client\_id;

clientNameInput.value = client.client\_name;

clientEmailInput.value = client.client\_email;

clientPhoneInput.value = client.client\_phone;

submitBtn.textContent = 'Actualizar Cliente';

}

if (e.target.classList.contains('delete-btn')) {

const id = e.target.dataset.id;

await fetch(`/api/clients/${id}`, { method: 'DELETE' });

fetchClients();

}

});

clearBtn.addEventListener('click', () => {

form.reset();

clientIdInput.value = '';

submitBtn.textContent = 'Crear Cliente';

});

fetchClients();

</script>

</body>

</html>