PRÁCTICA 22: APLICACIÓN DE GESTIÓN DE USUARIOS CON NODE.JS Y POSTGRESQL

DESCRIPCIÓN DEL PROYECTO

Esta práctica consiste en una aplicación web sencilla para la gestión de usuarios, desarrollada con Node.js y PostgreSQL. La aplicación permite:

1. Visualizar una lista de usuarios almacenados en una base de datos PostgreSQL
2. Añadir nuevos usuarios a través de un formulario web
3. Almacenar la información en una base de datos relacional

El proyecto utiliza una arquitectura cliente-servidor, donde el backend está desarrollado con Express.js y el frontend es una página HTML con JavaScript para la interacción con el usuario.

ESTRUCTURA DEL PROYECTO

```
c:\Users\Jaf\Desktop\WEB\Render\
|   .env.example      # Plantilla para variables de entorno
|   .gitignore        # Archivos ignorados por Git
|   db.js             # Configuración de conexión a la base de datos
|   init-db.js        # Script para inicializar la base de datos
|   package.json      # Dependencias y configuración del proyecto
|   server.js         # Servidor Express con endpoints API
|
└───public/
        |   index.html    # Interfaz de usuario
```

EXPLICACIÓN DE LOS ARCHIVOS

1. SERVER.JS

Este es el archivo principal que inicia el servidor web y define los endpoints de la API:

```
require("dotenv").config();
const express = require("express");
const db = require("./db");
const app = express();

// Middleware para procesar JSON
app.use(express.json());

// Servir archivos estáticos desde la carpeta 'public'
app.use(express.static("public"));

app.get("/usuarios", async (req, res) => {
  const result = await db.query("SELECT * FROM usuarios");
  res.json(result.rows);
});
```

```
// Nuevo endpoint para añadir usuarios
app.post("/usuarios", async (req, res) => {
 try {
  const { nombre, email, edad } = req.body;

  // Validación básica
  if (!nombre || !email) {
   return res.status(400).json({ error: "Nombre y email son obligatorios" });
  }

  const result = await db.query(
   "INSERT INTO usuarios (nombre, email, edad) VALUES ($1, $2, $3) RETURNING *",
   [nombre, email, edad]
  );

  res.status(201).json(result.rows[0]);
 } catch (error) {
  console.error("Error al insertar usuario:", error);
  res.status(500).json({ error: "Error al insertar el usuario" });
 }
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Servidor en puerto ${PORT}`));
```

Funcionalidad:

- Configura un servidor Express
- Define dos endpoints:
  - GET /usuarios : Devuelve todos los usuarios de la base de datos
  - POST /usuarios : Añade un nuevo usuario a la base de datos
- Sirve archivos estáticos desde la carpeta 'public'

2. DB.JS

Este archivo configura la conexión a la base de datos PostgreSQL:

```
const { Pool } = require("pg"); // Import the pg library que sirve para conectar a la base de datos

const pool = new Pool({
 connectionString: process.env.DATABASE_URL,
 ssl: { rejectUnauthorized: false },
});

module.exports = pool;
```

Funcionalidad:

- Crea un pool de conexiones a PostgreSQL usando la URL de conexión definida en las variables de entorno
- Configura SSL para permitir conexiones seguras (necesario para Render)

3. INIT-DB.JS

Este archivo contiene la lógica para inicializar la base de datos:

```
require("dotenv").config(); // Carga las variables de entorno desde .env
const db = require("./db"); // Importa la conexión a la base de datos

async function initDatabase() {
 try {
  // Crear tabla usuarios si no existe
  await db.query(`
   CREATE TABLE IF NOT EXISTS usuarios (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    edad INTEGER
   )
  `);

  console.log("Tabla usuarios creada o ya existente");
 } catch (error) {
  console.error("Error al inicializar la base de datos:", error);
 }
}

module.exports = initDatabase;
```

Funcionalidad:

- Crea la tabla usuarios si no existe
- Define la estructura de la tabla con campos para id, nombre, email y edad

4. PUBLIC/INDEX.HTML

Este archivo contiene la interfaz de usuario:

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gestión de Usuarios</title>
  <style>
    /* ... existing code ... */
```

```
  </style>
</head>
<body>
  <h1>Gestión de Usuarios</h1>

  <form id="userForm">
    <div class="form-group">
      <label for="nombre">Nombre:</label>
      <input type="text" id="nombre" required>
    </div>
    <div class="form-group">
      <label for="email">Email:</label>
      <input type="email" id="email" required>
    </div>
    <div class="form-group">
      <label for="edad">Edad:</label>
      <input type="number" id="edad">
    </div>
    <button type="submit">Añadir Usuario</button>
  </form>

  <div id="userList">
    <h2>Lista de Usuarios</h2>
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Nombre</th>
          <th>Email</th>
          <th>Edad</th>
        </tr>
      </thead>
      <tbody id="userTableBody">
        <!-- Los usuarios se cargarán aquí -->
      </tbody>
    </table>
  </div>

  <script>
    document.addEventListener('DOMContentLoaded', function() {
      // Cargar usuarios al iniciar
      loadUsers();

      // Manejar envío del formulario
      document.getElementById('userForm').addEventListener('submit', function(e) {
        e.preventDefault();

        const userData = {
          nombre: document.getElementById('nombre').value,
          email: document.getElementById('email').value,
```

```
            edad: document.getElementById('edad').value
        };

        fetch('/usuarios', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(userData)
        })
        .then(response => response.json())
        .then(data => {
            console.log('Usuario añadido:', data);
            // Limpiar formulario
            document.getElementById('userForm').reset();
            // Recargar lista de usuarios
            loadUsers();
        })
        .catch(error => {
            console.error('Error al añadir usuario:', error);
            alert('Error al añadir usuario');
        });
    });
});

function loadUsers() {
    fetch('/usuarios')
        .then(response => response.json())
        .then(users => {
            const tableBody = document.getElementById('userTableBody');
            tableBody.innerHTML = '';

            users.forEach(user => {
                const row = document.createElement('tr');
                row.innerHTML = `
                    <td>${user.id}</td>
                    <td>${user.nombre}</td>
                    <td>${user.email}</td>
                    <td>${user.edad || ''}</td>
                `;
                tableBody.appendChild(row);
            });
        })
        .catch(error => {
            console.error('Error al cargar usuarios:', error);
        });
}
</script>
</body>
</html>
```

Funcionalidad:

- Proporciona un formulario para añadir nuevos usuarios
- Muestra una tabla con la lista de usuarios existentes
- Utiliza JavaScript para interactuar con la API del servidor

5. PACKAGE.JSON

Este archivo define las dependencias y scripts del proyecto:

```
{
 "name": "node-render",
 "version": "1.0.0",
 "main": "server.js",
 "scripts": {
  "start": "node server.js"
 },
 "dependencies": {
  "express": "^4.18.2",
  "pg": "^8.9.0",
  "dotenv": "^16.0.3"
 }
}
```

Funcionalidad:

- Define las dependencias necesarias: Express, PostgreSQL y dotenv
- Configura el script de inicio de la aplicación

6. .ENV.EXAMPLE

Este archivo sirve como plantilla para las variables de entorno:

```
DATABASE_URL=postgres://usuario:clave@host:puerto/basedatos
```

Funcionalidad:

- Proporciona un ejemplo de cómo configurar la conexión a la base de datos

PASOS PARA DESPLEGAR EN RENDER

1. PREPARACIÓN DEL PROYECTO

1. Crear un repositorio en GitHub :

```
git init
git add .
git commit -m "Commit inicial"
git remote add origin https://github.com/TU_USUARIO/NOMBRE_DEL_REPOSITORIO.git
git push -u origin main
```

2. Asegúrate de tener un archivo .gitignore :

```
# .gitignore
```

```
.env
*.log
```

3. Modifica el archivo server.js para inicializar la base de datos:

```
require("dotenv").config();
const express = require("express");
const db = require("./db");
const initDatabase = require("./init-db");
const app = express();

// Inicializar la base de datos
initDatabase();

// ... existing code ...
```

2. CREAR UNA CUENTA EN RENDER

1. Regístrate en Render si aún no tienes una cuenta

2. Inicia sesión en tu cuenta

3. CREAR UNA BASE DE DATOS POSTGRESQL

1. En el dashboard de Render, haz clic en "New" y selecciona "PostgreSQL"

2. Configura tu base de datos:

    - Name: Nombre para tu base de datos (ej. "usuarios-db")

    - Database: Nombre de la base de datos (ej. "usuarios")

    - User: Se generará automáticamente

    - Region: Selecciona la región más cercana a ti

    - Plan: Free (para esta práctica)

3. Haz clic en "Create Database"

4. Guarda la información de conexión que se te proporcionará:

    - Internal Database URL: URL para conectarse desde servicios dentro de Render

    - External Database URL: URL para conectarse desde fuera de Render

4. DESPLEGAR LA APLICACIÓN WEB

1. En el dashboard de Render, haz clic en "New" y selecciona "Web Service"

2. Conecta tu repositorio de GitHub

3. Configura el servicio web:

- Name: Nombre para tu servicio (ej. "gestion-usuarios")

- Environment: Node

- Build Command: npm install

- Start Command: node server.js

- Plan: Free (para esta práctica)

4. En la sección "Environment Variables", añade:

- DATABASE_URL: Pega la "Internal Database URL" de tu base de datos PostgreSQL

5. Haz clic en "Create Web Service"

5. VERIFICAR EL DESPLIEGUE

1. Espera a que el despliegue se complete (puede tardar unos minutos)

2. Una vez desplegado, haz clic en la URL proporcionada para acceder a tu aplicación

3. Deberías ver la interfaz de gestión de usuarios

4. Prueba a añadir algunos usuarios y verificar que se muestran en la tabla

EJERCICIOS PRÁCTICOS

1. Modificación Básica: Añade un campo "teléfono" a la tabla de usuarios y actualiza la interfaz para permitir su ingreso.

2. Funcionalidad Avanzada: Implementa la funcionalidad de eliminar usuarios.

3. Mejora de Seguridad: Implementa validación de datos más robusta en el servidor.

4. Mejora de Interfaz: Añade estilos CSS para mejorar la apariencia de la aplicación.

5. Funcionalidad Extra: Implementa la funcionalidad de editar usuarios existentes.

CONCLUSIÓN

Esta práctica te ha permitido crear y desplegar una aplicación web completa con frontend y backend, utilizando tecnologías modernas como Node.js, Express, PostgreSQL y servicios en la nube como Render. Has aprendido a:

1. Configurar un servidor web con Express

2. Conectar a una base de datos PostgreSQL

3. Implementar operaciones CRUD básicas

4. Crear una interfaz de usuario interactiva

5. Desplegar una aplicación en la nube

Estos conocimientos son fundamentales para el desarrollo web moderno y te servirán como base para proyectos más complejos en el futuro.