Implementacion del CRUD de Proyectos en Node.js  
  
**Configuración e implementación del CRUD para proyectos en Node.js, probando todas las funcionalidades mediante Postman, incluyendo la creación, actualización, eliminación y obtención de proyectos desde la base de datos SQL Server.**

**1. Implementación del CRUD de Proyectos en Node.js**

* **GET**: Recuperamos todos los proyectos almacenados en la base de datos con éxito.
  + Ruta: /api/proyectos (GET)
  + Utilizamos Postman para verificar que la consulta trae correctamente todos los registros.
* **POST**: Implementamos la funcionalidad para crear nuevos proyectos.
  + Ruta: /api/proyectos (POST)
  + Probamos en Postman con un cuerpo JSON para insertar un nuevo proyecto y verificamos que se inserta correctamente en la base de datos.
* **PUT**: Actualizamos un proyecto existente.
  + Ruta: /api/proyectos/:id (PUT)
  + Realizamos una prueba para modificar un proyecto existente (en este caso, cambiamos el proyecto “Conversor de Monedas” a uno más sencillo) y verificamos que los datos se actualizan correctamente.
* **DELETE**: Eliminamos un proyecto de la base de datos.
  + Ruta: /api/proyectos/:id (DELETE)
  + Al principio, tuvimos un error de referencia debido a pruebas relacionadas, pero agregamos una funcionalidad de eliminación en cascada (ON DELETE CASCADE), lo que nos permitió eliminar el proyecto sin problemas.

**2. Pruebas con Postman:**

* Verificamos que cada uno de los métodos (GET, POST, PUT, DELETE) funciona correctamente utilizando Postman.
* Corregimos errores y ajustamos la funcionalidad para obtener el comportamiento deseado, especialmente en los casos de eliminación y actualización.

**Reminder de errores iniciales**

1. **Error con la conexión entre frontend (React) y backend (Node.js)**:
   * **Problema**: Al intentar conectar el frontend con el backend, se producían errores de proxy en la configuración, mostrando mensajes como *"Proxy error: Could not proxy request /api/proyectos from localhost:3000 to localhost:3001"*. Esto ocurría porque las rutas de API estaban mal configuradas en React.
   * **Solución**: Ajustamos la configuración del package.json para que el proxy apunte correctamente a localhost:3001 y modificamos el código de React para que las solicitudes usen solo /api/proyectos en lugar de http://localhost:3000/api/proyectos.
2. **Vulnerabilidades de seguridad en los paquetes de npm**:
   * **Problema**: Al actualizar las dependencias del proyecto con npm audit fix, surgieron advertencias de vulnerabilidades y posibles cambios que afectaban la estabilidad del proyecto.
   * **Solución**: Decidimos optar por **la opción 1**: continuar el desarrollo sin resolver todas las vulnerabilidades menores o moderadas, y revisar al final del proyecto si es necesario.
3. **Error en Node.js al levantar el frontend**:
   * **Problema**: Aparecía el error *"Error: error:0308010C*

*envelope routines::unsupported"* al iniciar la aplicación React.

* + **Solución**: Ejecutamos el comando set NODE\_OPTIONS=--openssl-legacy-provider para solucionar el problema relacionado con versiones más recientes de Node.js que no son compatibles con algunas bibliotecas antiguas.

**Pasos a seguir para replicar el proceso de hoy:**

1. **Configuración de rutas en el servidor**:

Implementamos rutas **GET**, **POST**, **PUT** y **DELETE** en el archivo server.js para interactuar con la base de datos.  
  
const express = require('express'); // Importa Express

const { poolPromise, sql } = require('./db'); // Importa db.js desde el archivo que creaste

const app = express();

const port = process.env.PORT || 3001;

// Configuración de middlewares

app.use(express.json());

// Ruta GET para obtener todos los proyectos

app.get('/api/proyectos', async (req, res) => {

    try {

        const pool = await poolPromise; // Obtiene la conexión desde poolPromise

        const result = await pool.request().query('SELECT \* FROM Proyectos'); // Consulta a la tabla Proyectos

        res.json(result.recordset); // Devuelve los datos en formato JSON

    } catch (err) {

        res.status(500).send({ message: err.message }); // Manejo de errores

    }

});

// Ruta POST para crear un nuevo proyecto

app.post('/api/proyectos', async (req, res) => {

    try {

        const { idUsuario, nombreProyecto, descripcion, fechaInicio, fechaFin, estado } = req.body;

        // Verifica si se han proporcionado todos los campos requeridos

        if (!idUsuario || !nombreProyecto || !descripcion || !fechaInicio || !fechaFin || !estado) {

            return res.status(400).send({ message: 'Por favor, llena todos los campos requeridos.' });

        }

        const pool = await poolPromise;

        await pool.request()

            .input('idUsuario', sql.NVarChar, idUsuario)

            .input('nombreProyecto', sql.NVarChar, nombreProyecto)

            .input('descripcion', sql.NVarChar, descripcion)

            .input('fechaInicio', sql.DateTime, fechaInicio)

            .input('fechaFin', sql.DateTime, fechaFin)

            .input('estado', sql.NVarChar, estado)

            .query('INSERT INTO Proyectos (idUsuario, nombreProyecto, descripcion, fechaInicio, fechaFin, estado) VALUES (@idUsuario, @nombreProyecto, @descripcion, @fechaInicio, @fechaFin, @estado)');

        res.status(201).send({ message: 'Proyecto creado exitosamente.' });

    } catch (err) {

        res.status(500).send({ message: err.message });

    }

});

/\* Ruta POST para crear un nuevo proyecto (prueba básica)

app.post('/api/proyectos', (req, res) => {

    // Respuesta básica para probar la ruta POST

    res.status(200).send('POST request to /api/proyectos received!');

});\*/

// Ruta PUT para actualizar un proyecto existente

app.put('/api/proyectos/:id', async (req, res) => {

    try {

        const { id } = req.params;

        const { nombreProyecto, descripcion, fechaInicio, fechaFin, estado } = req.body;

        const pool = await poolPromise;

        const result = await pool.request()

            .input('nombreProyecto', sql.NVarChar, nombreProyecto)

            .input('descripcion', sql.NVarChar, descripcion)

            .input('fechaInicio', sql.DateTime, fechaInicio)

            .input('fechaFin', sql.DateTime, fechaFin)

            .input('estado', sql.NVarChar, estado)

            .input('id', sql.Int, id)

            .query('UPDATE Proyectos SET nombreProyecto = @nombreProyecto, descripcion = @descripcion, fechaInicio = @fechaInicio, fechaFin = @fechaFin, estado = @estado WHERE idProyecto = @id');

        if (result.rowsAffected[0] === 0) {

            return res.status(404).send({ message: 'Proyecto no encontrado.' });

        }

        res.send({ message: 'Proyecto actualizado exitosamente.' });

    } catch (err) {

        res.status(500).send({ message: err.message });

    }

});

// Ruta DELETE para eliminar un proyecto y sus pruebas/defectos relacionados

app.delete('/api/proyectos/:id', async (req, res) => {

    try {

        const { id } = req.params;

        const pool = await poolPromise;

        // Primero, elimina los defectos relacionados con las pruebas del proyecto

        await pool.request()

            .input('id', sql.Int, id)

            .query('DELETE FROM Defectos WHERE idPrueba IN (SELECT idPrueba FROM Pruebas WHERE idProyecto = @id)');

        // Luego, elimina las pruebas relacionadas con el proyecto

        await pool.request()

            .input('id', sql.Int, id)

            .query('DELETE FROM Pruebas WHERE idProyecto = @id');

        // Finalmente, elimina el proyecto

        const result = await pool.request()

            .input('id', sql.Int, id)

            .query('DELETE FROM Proyectos WHERE idProyecto = @id');

        if (result.rowsAffected[0] === 0) {

            return res.status(404).send({ message: 'Proyecto no encontrado.' });

        }

        res.send({ message: 'Proyecto y sus datos relacionados eliminados exitosamente.' });

    } catch (err) {

        res.status(500).send({ message: err.message });

    }

});

// Inicia el servidor

app.listen(port, () => {

    console.log(`Servidor ejecutándose en http://localhost:${port}`);

});

1. **Pruebas de API con Postman**:
   * Asegúrate de que las rutas estén configuradas correctamente en Postman.

Usa cuerpos JSON válidos para las solicitudes **POST** y **PUT**.  
**GET:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**POST:**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

"Proyecto creado exitosamente."

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**PUT:**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Imagen que contiene Calendario

Descripción generada automáticamente

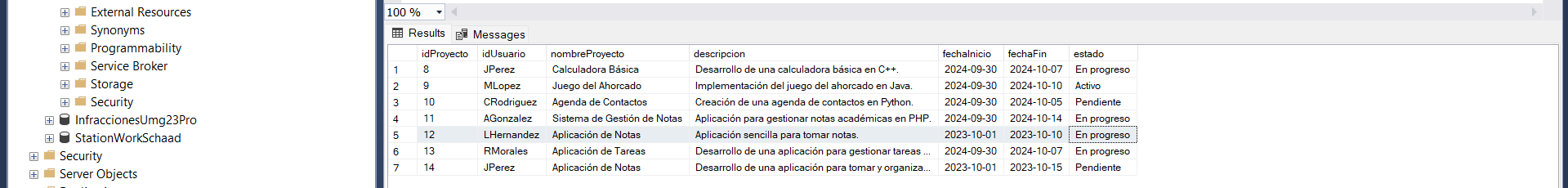
"Proyecto actualizado exitosamente."

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



**DELETE:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

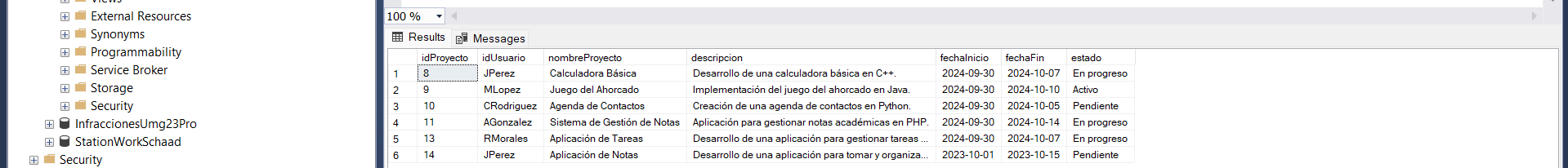
Descripción generada automáticamente

"Proyecto y sus datos relacionados eliminados exitosamente."  
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



1. **Manejo de errores y referencias cruzadas**:
   * Al intentar eliminar proyectos que tienen pruebas o defectos asociados, primero debes asegurar la integridad referencial o activar la opción de eliminación en cascada (**ON DELETE CASCADE**).

**Conclusión:**

Hoy completamos con éxito la implementación del CRUD de proyectos, resolviendo problemas de conexión y referencias en la base de datos, y aseguramos que las funciones están listas para ser utilizadas en el sistema de gestión de pruebas.