Práctica: Vinculación de una Base de Datos MySQL a un Proyecto Node.js con TypeScript

Objetivo:

En esta práctica guiada, los alumnos aprenderán a conectar un proyecto en Node.js con TypeScript a una base de datos MySQL. Implementarán una conexión básica y realizarán operaciones CRUD sencillas utilizando un ORM (Sequelize).

1. Configuración inicial

Crear el proyecto Node.js con TypeScript:

- Abre una terminal y crea un nuevo proyecto:

```
mkdir practica-mysql-node
cd practica-mysql-node
npm init -y
```

- Instala las dependencias necesarias:

```
npm install typescript ts-node nodemon @types/node --save-dev
npm install express sequelize mysql2 dotenv
npm install @types/express --save-dev
```

Configurar TypeScript:

- Crea el archivo tsconfig.json:

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true
  }
}
```

- Añade scripts al archivo package.json para facilitar el desarrollo:

```
"scripts": {
  "start": "node dist/index.js",
  "dev": "nodemon src/index.ts"
}
```

- Estructura del proyecto: Crea las siguientes carpetas y archivos:

```
practica-mysql-node/
├── src/
│   ├── config/
│   │   └── database.ts
│   ├── models/
│   │   └── usuario.ts
│   ├── routes/
│   │   └── usuarioRoutes.ts
│   ├── controllers/
│   │   └── usuarioController.ts
│   └── index.ts
├── .env
├── package.json
├── tsconfig.json
```

2. Conexión a la base de datos

Crear la base de datos en MySQL:

- Accede al cliente MySQL (puede ser desde MySQL Workbench o terminal):

```
CREATE DATABASE practica_db;
```

- Dentro de la base de datos, crea una tabla usuarios:

```
CREATE TABLE usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE
);
```

- Configurar la conexión en src/config/database.ts:
- Crea un archivo .env en la raíz del proyecto con las credenciales de la base de datos:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=tu_password
DB_NAME=practica_db
```

- Implementa la conexión en src/config/database.ts:

```
import { Sequelize } from 'sequelize';

const sequelize = new Sequelize(
  process.env.DB_NAME as string,
  process.env.DB_USER as string,
  process.env.DB_PASSWORD,
  {
    host: process.env.DB_HOST,
    dialect: 'mysql',
  }
);

export default sequelize;
```

Probar la conexión:

- En src/index.ts, verifica la conexión:

```
import express from 'express';
import sequelize from './config/database';

const app = express();
const PORT = 3000;

app.use(express.json());

app.get('/', (req, res) => {
```

```
  res.send('Servidor funcionando');
});

// Probar conexión a la base de datos
sequelize
  .authenticate()
  .then(() => console.log('Conexión a la base de datos exitosa'))
  .catch((error) => console.error('Error al conectar la base de datos:', error));

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

3. Definición del modelo

Crear el modelo en src/models/usuario.ts:

```
import { DataTypes, Model } from 'sequelize';
import sequelize from '../config/database';

class Usuario extends Model {
  public id!: number;
  public nombre!: string;
  public email!: string;
}

Usuario.init(
  {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
    },
    nombre: {
      type: DataTypes.STRING,
      allowNull: false,
    },
```

```
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
    },
  },
  {
    sequelize,
    tableName: 'usuarios',
  }
);

export default Usuario;
```

Sincronizar el modelo:

- En src/index.ts, sincroniza el modelo con la base de datos:

```
import Usuario from './models/usuario';

sequelize.sync({ force: true }).then(() => {
  console.log('Base de datos sincronizada');
});
```

4. CRUD básico

Crear el controlador en src/controllers/usuarioController.ts:

```
import { Request, Response } from 'express';
import Usuario from '../models/usuario';

export const obtenerUsuarios = async (req: Request, res: Response) => {
  const usuarios = await Usuario.findAll();
  res.json(usuarios);
};

export const crearUsuario = async (req: Request, res: Response) => {
```

```
  const { nombre, email } = req.body;
  const nuevoUsuario = await Usuario.create({ nombre, email });
  res.status(201).json(nuevoUsuario);
};
```

Crear las rutas en src/routes/usuarioRoutes.ts:

```
import { Router } from 'express';
import { obtenerUsuarios, crearUsuario } from '../controllers/usuarioController';

const router = Router();

router.get('/usuarios', obtenerUsuarios);
router.post('/usuarios', crearUsuario);

export default router;
```

Integrar las rutas en src/index.ts:

```
import usuarioRoutes from './routes/usuarioRoutes';

app.use('/api', usuarioRoutes);
```

5. Probar la aplicación

Ejecuta el servidor:

```
npm run dev
```

Usa Postman para realizar las siguientes pruebas:

- GET http://localhost:3000/api/usuarios: Lista todos los usuarios.
- POST http://localhost:3000/api/usuarios: Crea un usuario con el cuerpo:

```
{
  "nombre": "Carlos",
  "email": "carlos@example.com"
}
```