Estructura del Proyecto

```
nodejs-sequelize-crud-api/
├── config/
│   └── config.json
├── controllers/
│   └── productController.js
├── models/
│   ├── index.js
│   └── product.js
├── routes/
│   └── productRoutes.js
├── .env
├── app.js
├── package.json
└── package-lock.json
```

Ahora instala las dependencias necesarias:

Bash

```
npm install express sequelize pg dotenv
npm install --save-dev nodemon
```

- `express` : Framework web para Node.js.
- `sequelize` : ORM (Object-Relational Mapping) para Node.js.
- `pg` : Driver de PostgreSQL.
- `dotenv` : Para cargar variables de entorno desde un archivo `.env` .
- `nodemon` : Para reiniciar el servidor automáticamente con cada cambio.

npm install express sequelize pg dotenv

npm install --save-dev nodemon

2. Configurar Sequelize

Sequelize necesita saber cómo conectarse a tu base de datos PostgreSQL existente. Vamos a usar un archivo de configuración y variables de entorno por seguridad.

Crea un archivo `.env` en la raíz de tu proyecto:

`.env` :

```
DB_USERNAME=your_db_username
DB_PASSWORD=your_db_password
DB_DATABASE=your_existing_database_name
DB_HOST=localhost
DB_PORT=5432 # Puerto por defecto de PostgreSQL
PORT=3000
```

DB_USERNAME=your_db_username DB_PASSWORD=your_db_password DB_DATABASE=your_existing_database_name DB_HOST=localhost DB_PORT=5432 # Puerto por defecto de PostgreSQL PORT=3000

Asegúrate de reemplazar `your_db_username`, `your_db_password`, y `your_existing_database_name` con tus credenciales y el nombre real de tu base de datos PostgreSQL existente.

Crea la carpeta `config` y dentro, el archivo `config.json`:

`config/config.json`:

```
{
 "development": {
  "username": process.env.DB_USERNAME,
  "password": process.env.DB_PASSWORD,
  "database": process.env.DB_DATABASE,
  "host": process.env.DB_HOST,
  "port": process.env.DB_PORT,
  "dialect": "postgres",
  "logging": false,
  "dialectOptions": {
    "ssl": {
      "require": false,
      "rejectUnauthorized": false
    }
  }
 },
 "test": {
  "username": process.env.DB_USERNAME,
  "password": process.env.DB_PASSWORD,
  "database": process.env.DB_DATABASE_TEST,
  "host": process.env.DB_HOST,
  "port": process.env.DB_PORT,
  "dialect": "postgres"
 },
 "production": {
  "username": process.env.DB_USERNAME,
  "password": process.env.DB_PASSWORD,
  "database": process.env.DB_DATABASE_PROD,
  "host": process.env.DB_HOST,
  "port": process.env.DB_PORT,
  "dialect": "postgres"
 }
}
```

<<<<<<<CREAR MODELOS<>>>>>>

Crea el archivo `index.js` dentro de la carpeta `models`. Este archivo es crucial para cargar todos tus modelos y establecer la conexión a la base de datos.

`models/index.js`:

```
'use strict';

const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const process = require('process');
const basename = path.basename(__filename);
const env = process.env.NODE_ENV || 'development';
const config = require(__dirname + '/../config/config.json')[env];
const db = {};

let sequelize;
if (config.use_env_variable) {
  sequelize = new Sequelize(process.env[config.use_env_variable], config);
} else {
  sequelize = new Sequelize(config.database, config.username, config.password, config);
}

fs
  .readdirSync(__dirname)
  .filter(file => {
    return (
      file.indexOf('.') !== 0 &&
      file !== basename &&
      file.slice(-3) === '.js' &&
      file.indexOf('.test.js') === -1
    );
  })
  .forEach(file => {
    const model = require(path.join(__dirname, file))(sequelize, Sequelize.DataTypes);
    db[model.name] = model;
  });

Object.keys(db).forEach(modelName => {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;
```

4. Crear el Controlador (ProductController)

Este archivo contendrá la lógica para manejar las solicitudes CRUD.

Crea la carpeta `controllers` y dentro, el archivo `productController.js`:

`controllers/productController.js`:

```
const db = require('../models');
const Product = db.Product;

// Crear un nuevo producto
exports.create = async (req, res) => {
 try {
  const { name, description, price } = req.body;
  if (!name || !price) {
   return res.status(400).send({ message: 'El nombre y el precio son campos requeridos.' });
  }

  const product = await Product.create({ name, description, price });
  res.status(201).send(product);
 } catch (error) {
  console.error('Error al crear el producto:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al crear el producto.'
  });
 }
};

// Obtener todos los productos
exports.findAll = async (req, res) => {
 try {
  const products = await Product.findAll();
  res.status(200).send(products);
 } catch (error) {
  console.error('Error al recuperar los productos:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al recuperar los productos.'
  });
 }
};

// Obtener un producto por ID
exports.findOne = async (req, res) => {
 const id = req.params.id;
 try {
  const product = await Product.findByPk(id);
  if (!product) {
   return res.status(404).send({ message: `No se encontró un producto con id=${id}.` });
  }
  res.status(200).send(product);
 } catch (error) {
  console.error('Error al recuperar el producto por ID:', error);
  res.status(500).send({
   message: 'Error al recuperar el producto con id=' + id
  });
 }
};

// Actualizar un producto por ID
exports.update = async (req, res) => {
 const id = req.params.id;
 try {
  const [num] = await Product.update(req.body, {
```

```
    where: { id: id }
   });

   if (num === 1) {
    res.status(200).send({ message: 'El producto se actualizó correctamente.' });
   } else {
    res.status(404).send({
     message: `No se puede actualizar el producto con id=${id}. Tal vez el producto no fue
encontrado o req.body está vacío.`
    });
   }
  } catch (error) {
   console.error('Error al actualizar el producto:', error);
   res.status(500).send({
    message: 'Error al actualizar el producto con id=' + id
   });
  }
};

// Eliminar un producto por ID
exports.delete = async (req, res) => {
 const id = req.params.id;
 try {
  const num = await Product.destroy({
   where: { id: id }
  });

  if (num === 1) {
   res.status(200).send({ message: 'El producto se eliminó correctamente!' });
  } else {
   res.status(404).send({
    message: `No se pudo eliminar el producto con id=${id}. Tal vez el producto no fue
encontrado!`
    });
  }
 } catch (error) {
  console.error('Error al eliminar el producto:', error);
  res.status(500).send({
   message: 'No se pudo eliminar el producto con id=' + id
  });
 }
};

// Eliminar todos los productos
exports.deleteAll = async (req, res) => {
 try {
  const num = await Product.destroy({
   where: {},
   truncate: false // No truncar la tabla, solo eliminar filas
  });
  res.status(200).send({ message: `${num} productos se eliminaron correctamente!` });
 } catch (error) {
  console.error('Error al eliminar todos los productos:', error);
  res.status(500).send({
   message: error.message || 'Ocurrió un error al eliminar todos los productos.'
  });
 }
```

};

5. Definir las Rutas (ProductRoutes)

Este archivo mapeará las URL de la API a las funciones del controlador.

Crea la carpeta `routes` y dentro, el archivo `productRoutes.js`:

`routes/productRoutes.js`:

```
const express = require('express');
const router = express.Router();
const products = require('../controllers/productController.js');

// Rutas para las operaciones CRUD en productos

// Crear un nuevo producto
router.post('/', products.create);

// Obtener todos los productos
router.get('/', products.findAll);

// Obtener un producto por ID
router.get('/:id', products.findOne);

// Actualizar un producto por ID
router.put('/:id', products.update);

// Eliminar un producto por ID
router.delete('/:id', products.delete);

// Eliminar todos los productos
router.delete('/', products.deleteAll);

module.exports = router;
```

6. Configurar la Aplicación Principal (app.js)

Este es el punto de entrada de nuestra aplicación Express.

`app.js`:

```
require('dotenv').config(); // Cargar variables de entorno al inicio

const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const db = require('./models'); // Importa la configuración de la base de datos y los modelos

// Opciones de CORS (puedes configurarlo para permitir solo orígenes específicos)
var corsOptions = {
  origin: '*' // Permite todas las solicitudes. En producción, limita a tus dominios.
};
```

```
app.use(cors(corsOptions));

// Analizar solicitudes de tipo content-type - application/json
app.use(bodyParser.json());

// Analizar solicitudes de tipo content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// **IMPORTANTE**: No uses `db.sequelize.sync({ force: true })` si la base de datos ya existe
// y quieres preservar los datos. `sync()` sin `force: true` intentará crear tablas si no existen,
// pero no las borrará ni recreará si ya lo hacen.
// Para una base de datos existente, es más seguro simplemente conectarse.
db.sequelize.authenticate()
  .then(() => {
    console.log('Conexión a la base de datos establecida exitosamente.');
  })
  .catch(err => {
    console.error('No se pudo conectar a la base de datos:', err);
    process.exit(); // Salir si no hay conexión a la DB
  });


// Ruta simple de bienvenida
app.get('/', (req, res) => {
  res.json({ message: 'Bienvenido a la API RESTful de productos con Node.js, Express y Sequelize
(DB Existente).' });
});

// Rutas de productos
const productRoutes = require('./routes/productRoutes');
app.use('/api/products', productRoutes);

// Establecer puerto, escuchar solicitudes
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`El servidor está corriendo en el puerto ${PORT}.`);
});
```

8. Ejecutar la Aplicación y Probar

Para iniciar el servidor, agrega un script a tu `package.json`:

`package.json`:

```
{
  "name": "nodejs-sequelize-crud-api",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

```
  "dependencies": {
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "pg": "^8.12.0",
    "sequelize": "^6.37.3"
  },
  "devDependencies": {
    "nodemon": "^3.1.4",
    "sequelize-cli": "^6.6.2"
  }
}
```