

CRUD / Express y MySql - Parte 1

En esta cuarta parte vamos a realizar otro CRUD(Create, Read, Update, Delete) esta vez con una base de datos relacional. Antes habíamos creado nuestro proyecto con una base de datos no relacional como lo era MongoDB, ahora trabajaremos con un gestor MySql en este caso usaremos **Xampp**.

Primero creamos nuestro directorio **backend-mysql** e iniciamos un nuevo proyecto node.js y descargamos las siguientes dependencias:

Instalación de dependencias

[Copiar código](#)

```
npm init -y
npm i express
npm i cors sequelize mysql2
npm i nodemon -D
```

Mira la documentación de **sequelize** [aquí](#)

En este proyecto no trabajamos con REQUIRE trabajaremos con módulos, para ello modifica nuestra package.json, este debe quedar de la siguiente manera:

package.json.js

[Copiar código](#)

```
{
  "name": "backend-mysql",
  "version": "1.0.0",
  "description": "",
```

```
"main": "index.js",
"type": "module", // le especificamos que vamos a trabajar con Módulos y no con require
"scripts": {
  "start": "nodemon src/index.js" // Modificamos para que inicie nuestra app con nodemon desde
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
  "cors": "^2.8.5",
  "express": "^4.19.2",
  "mysql2": "^3.9.6",
  "sequelize": "^6.37.3"
},
"devDependencies": {
  "nodemon": "^3.1.0"
}
}
```

Creación de base de datos relacional

En **Xampp** crea una nueva base de datos llamada **citasDB**

Bases de datos

18.04.2024 5:25:12

Crear base de datos






citasBD

utf8mb4_general_ci

Crear

☐ Seleccionar todo

 Eliminar

	Base de datos	Cotejamiento	Acción
<input type="checkbox"/>	information_schema	utf8_general_ci	 Seleccionar privilegios
<input type="checkbox"/>	mysql	utf8mb4_general_ci	 Seleccionar privilegios
<input type="checkbox"/>	performance_schema	utf8_general_ci	 Seleccionar privilegios
<input type="checkbox"/>	phpmyadmin	utf8_bin	 Seleccionar privilegios
<input type="checkbox"/>	test	latin1_swedish_ci	 Seleccionar privilegios

Total: 5

Crea una nueva tabla llamada **citas**

Crear nueva tabla

Nombre de la tabla

Número de columnas

citas

7

Crear

Verifica que los 7 campos queden con la siguiente configuración:

Estructura de tabla

Vista de relaciones

18-04-2024 5:25 2

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/>	1 id	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
<input type="checkbox"/>	2 nombre_medico	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	3 especialidad	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	4 fecha	varchar(50)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	5 duracion	varchar(50)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	6 createdAt	date			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	7 updatedAt	date			No	Ninguna			Cambiar Eliminar Más

Crea el primer registro

estructura de carpetas

Copiar código

```
INSERT INTO `citas` (
  `nombre_medico`,
  `especialidad`,
  `fecha`,
  `duracion`)
VALUES (
  'Julio Amaya',
  'Medico General',
  '18-04-2024',
  '30 min')
```

NOTA

IMPORTANTE! 'Sequelize' en su documentación nos recomienda que creamos dos campos adicionales en nuestra base de datos: 'createdAt' y 'updatedAt' de tipo 'date'. Para que no se generen errores en Xampp debes omitir esos dos campos y adicional omitir el campo de ID, por esta razón de los 7 campos de la tabla solo debes ingresar datos a 4 campos.

Estructura de carpetas

Ahora creamos los directorios y archivos necesarios para conectarnos y trabajar con nuestra base de datos

estructura de carpetas

Copiar código

```
↓ config// New
  dg.js// New
↓ controllers// New
  CitasController.js// New
↓ models// New
  Citas.js// New
→ node_modules
↓ routes// New
  RouterCitas.js// New
↓ src// New
  index.js// New
package-lock.json
package.json
```


Iniciando CRUD

En esta parte se inicia con el Crud, definiendo algunas rutas, conexión a la base de datos y creación de nuestro primer método para agregar citas a nuestra base de datos relacional, ten presente que el modelo no contine dos campos que en la base dedatos definimos inicialmente, esto por que son campos que funcionan con **Sequelize** y el automáticamente se encarga de su gestión, por lo que nosotros no debemos no debemos tocarlos.

Nuestra configuración de momento queda definida de la siguiente manera:

config/db.js

config/db.js

 Copiar código

```
// config/db.js
import { Sequelize } from "sequelize";

const BD = new Sequelize('citasbd', 'root', '', {
  host: 'localhost',
  dialect: 'mysql'
})
```

```
export default BD;
```


18.04.2024 S28-M2

NOTA

Configura la conexión a la base de datosMySQL utilizando Sequelize, una librería de Node.js para manejar bases de datos relacionales. Especifica el nombre de la base de datos 'citasbd', usuario 'root', sin contraseña(vacío). Además, se define el host como 'localhost' y se especifica el dialecto 'mysql'. La conexión establecida se exporta como un objeto BD que puede ser utilizado en otras partes de la aplicación para interactuar con la base de datos.

model/Citas.js

model/Citas.js

 Copiar código

```
// model/Citas.js
// Importamos la conexión a la BD
import BD from "../config/db.js";
import { DataTypes } from "sequelize";

const Citas = BD.define('citas', {

  nombre_medico :
  {
    type: DataTypes.STRING,
    allowNull:false
  },

  especialidad :
  {
    type: DataTypes.STRING,
    allowNull:false
  },

  fecha:
  {
    type: DataTypes.STRING,
    allowNull:false
  },

  duracion :
  {
    type: DataTypes.STRING,
    allowNull:false
  },

});
```

```
export default Citas;
```


18.04.2024 S28-M2

NOTA

Este código define un modelo 'Citas' utilizando Sequelize, un ORM para bases de datos relacionales en Node.js. El modelo 'Citas' representa una tabla en la base de datos con campos como 'nombre_medico', 'especialidad', 'fecha' y 'duracion', todos de tipo STRING y que no permiten valores nulos. La conexión a la base de datos se importa de un archivo de configuración y se define el nombre de la tabla como 'citas'. Este modelo se exporta para ser utilizado en otras partes del proyecto.

controller/CitasController.js

controller/CitasController.js

 Copiar código

```
// controller/CitasController.js
// Importamos el modelo
import Citas from '../model/Citas.js';

// Creamos los métodos CRUD


// Creamos una función para agregar citas
export const agregarCitas = async (req, res) => {
  try {
    await Citas.create(req.body)
    res.json({msg: "Cita creada con éxito 😊"});
  } catch (error) {
    res.json ({msg: error.message});
  }
}
```

NOTA

Este código importa el modelo 'Citas' definido anteriormente y define una función 'agregarCitas' para el CRUD. Dentro de esta función, se intenta crear una nueva entrada en la base de datos utilizando el método 'create' del modelo 'Citas' con los datos proporcionados en 'req.body'. Si la operación se realiza con éxito, se envía una respuesta JSON indicando que la cita se ha creado correctamente. En caso de error, se captura la excepción y se envía un mensaje JSON con el error correspondiente.

routes/RoutesCitas.js

routes/RoutesCitas.js

 Copiar código

```
// routes/RoutesCitas.js
// Importamos express
import express from 'express';

// Importamos nuestro controlador
import {agregarCitas} from '../controller/CitasController.js';

const router = express.Router();
router.post('/', agregarCitas);


export default router;
```

NOTA

Este código importa Express y el controlador 'agregarCitas' del archivo 'CitasController.js'. Luego, define un nuevo enrutador de Express con `express.Router()`. Posteriormente, se configura una ruta POST en el enrutador que, cuando se accede, llama a la función 'agregarCitas' del controlador. Finalmente, exporta este enrutador para que pueda ser utilizado en otras partes de la aplicación.

src/index.js

src/index.js

 Copiar código

```
// src/index.js
import express from "express";
import cors from "cors";
// Importamos la configuración de la BD
import BD from '../config/db.js';

// Importamos el archivo de las rutas
import citasRoutes from '../routes/RoutesCitas.js';
```



```
// Definimos la variable para trabajar con Express
const app = express();
app.use(cors());
app.use(express.json());
app.use('/citas', citasRoutes );

// Autenticación BD
try {
  await BD.authenticate();
  console.log('Connection has been established successfully.');
```

—

```
} catch (error) {
  console.error('Unable to connect to the database:', error);
}

// Muestra mensaje en el navegador
app.get('/', (req, res) => {
  res.send("Hola mundo");
})

// Configuración del puerto del servidor
app.listen(5000, () => {
  console.log("El servidor esta corriendo ⚡ en http://localhost:5000/")
});
```

NOTA

El código comienza importando Express, CORS para manejar solicitudes de diferentes orígenes y la configuración de la base de datos desde 'db.js'. También importa las rutas de 'RoutesCitas.js'. Se crea una instancia de Express y se configuran los middleware para manejar JSON y CORS. Luego, se define la ruta '/citas' usando las rutas importadas. Se intenta autenticar la conexión con la base de datos y se imprime un mensaje de éxito o error en la consola. Además, hay una ruta de inicio que responde con "Hola mundo". Finalmente, el servidor se inicia en el puerto 5000 y se imprime un mensaje en la consola para confirmar la ejecución.

Comprobación del método 'agregarCitas'

18.04.2024 S28-M2

POST http://localhost:5000/citas

Save Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "nombre_medico": "Carlos",
3   "especialidad": "Terapeuta",
4   "fecha": "18-04-2024",
5   "duracion": "45 min"
6 }
```

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 17 ms Size: 304 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Cita creada con éxito 😊"
3 }
```



Mira el repositorio en Github