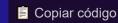
# **SESIÓN 29**

# CRUD / Express y MySql - Parte 2

En esta sesión o entrega del dia, se da continuidad con las funciones o los métodos de nuestro CRUD con Express y MySql configuración que venimos trabajando en nuestro Controller/CitasController.js, recordando que en la anterior sesión ya habíamos iniciado con el método de ingresar nuevas citas. La nueva configuraron con los cambios de hoy quedaría de la siguiente forma:

Controller/CitasController.js



```
import Citas from '../model/Citas.js';
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});
}
export const getAllCitas = async (req, res) =>{
 try {
   const citas = await Citas.findAll();
   res.json(citas);
  } catch (error) {
   res.json ({msg: error.message});
export const getCita = async (req, res) => {
   const cita = await Citas.findAll({
     where:{id:req.params.id}
   res.json(cita[0]);
  } catch (error) {
```

```
res.json({msg: error.message})
export const modificarCita = async (req, res) =>{
   await Citas.update(req.body, {
     where:{id: req.params.id}
   })
   res.json({msg: "Se modifico una cita"})
 } catch (error) {
   res.json({msg: error.message})
export const eliminarCita = async (req, res) => {
   let citas = await Citas.findAll({where:{id: req.params.id}});
   if (!citas[0]){
     res.json({msg: "No se encuentra la cita"});
   await citas[0].destroy();
   res.json({msg: "Se elimino una cita"})
 } catch (error) {
   res.json({msg: error.message})
}
```

Este código define un conjunto de funciones CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar citas médicas utilizando un modelo llamado Citas. La función agregarCitas permite crear una nueva cita utilizando los datos proporcionados en el cuerpo de la solicitud HTTP. getAllCitas devuelve todas las citas almacenadas en la base de datos. getCita busca una cita específica por su ID. modificarCita actualiza una cita existente con los datos proporcionados. Finalmente, eliminarCita elimina una cita por su ID después de verificar su existencia. Cada función maneja los errores de manera similar, respondiendo con un mensaje de error si ocurre alguna excepción.

Ahora configuramos las rutas en routes/RoutesCitas.js

```
routes/RoutesCitas.js

// Importamos express
import express from 'express';

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

```
import {getCita} from '../controller/CitasController.js';
import {modificarCita} from '../controller/CitasController.js';
import {eliminarCita} from '../controller/CitasController.js';

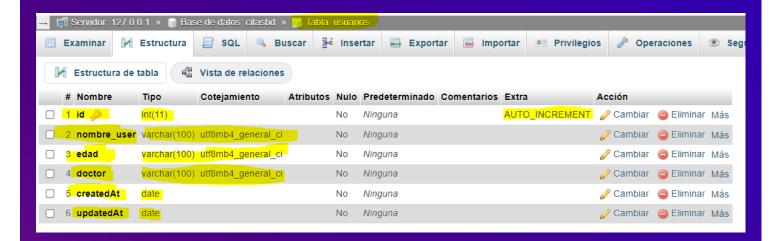
const router = express.Router();
router.post('/', agregarCitas);
router.get('/', getAllCitas);
router.get('/'id', getCita);
router.put('/:id', modificarCita);
router.delete('/:id', eliminarCita);

export default router;
```

NOTA
Este código define un enrutador en Express que maneja las operaciones CRUD
para citas médicas. Importa funciones específicas del controlador
CitasController.js para agregar, obtener, modificar y eliminar citas. Las
rutas configuradas incluyen agregar una nueva cita, obtener todas las citas,
obtener una cita por su ID, modificar una cita existente y eliminar una cita
por su ID. Cada ruta se asocia con su función correspondiente del controlador,
asegurando una gestión adecuada de las solicitudes HTTP.

# Actividad - Añadiendo nuevo modulo a nuestro CRUD con MySql

Crearemos una nueva tabla llamada 'usuarios' donde se trabajaran las mismas funciones para ingresar, mostrar, eliminar y modificar datos de nuestra base de datos.



Ahora en nuestro proyecto crearemos 3 archivos nuevos, estos serán necesarios para que nuestro nuevo modulo para usuarios funcione correctamente:

```
Estructura de carpetas
                                                                                       🖹 Copiancódigo
 ↓ config
     db.js
 ↓ controllers
     CitasController.js
     UsuariosController.js // New
 ↓ models
     Citas.js
     Usuarios.js // New
 → node modules
 ↓ routes
     RouterCitas.js
     RouterUsuarios.js // New
 ↓ src
     index.js
   package-lock.json
   package.json
```

Ahora en cuanto a la configuración de los archivos estos serian iguales a los anteriores usados para **citas** solo que aquí se le modificarían algunos nombres de variables y algunas rutas

controller/UsuariosController.js

```
Copiar código
```

```
// controller/UsuariosController.js
// Importamos el modelo
import Usuarios from '../model/Usuarios.js';

// Creamos los métodos CRUD

// Creamos una función para agregar usuarios
export const agregarUser = async (req, res) => {
    try {
        await Usuarios.create(req.body)
        res.json({msg: "Usuario creado con éxito ** "});

    } catch (error) {
        res.json ({msg: error.message});
    }

// Función para mostrar todos lss usuarios
export const getAllUser = async (req, res) => {
        try {
            const usuarios = await Usuarios.findAll();
            return res.json(usuarios);
    }
} catch (error) {
        return res.json ({msg: error.message});
}
}
```

```
export const getUser = async (req, res) => {
  try {
   let user = await Usuarios.findAll({where:{id:req.params.id}});
   if (!user[0]){
      return res.json({msg: "No se encontró el usuario con ese ID"});
   await res.json(user[0]);
 } catch (error) {
      return res.json({msg: error.message})
}
export const modificarUser = async (req, res) =>{
 try {
   let user = await Usuarios.update(req.body, { where:{id: req.params.id}})
   if(!user[0]){
      return res.json({msg: "El ID de usuario no es valido"})
   await res.json({msg: "Se modifico un usuario"})
  } catch (error) {
   return res.json({msg: error.message})
export const eliminarUser = async (req, res) => {
   let user = await Usuarios.findAll({where:{id: req.params.id}});
   if (!user[0]){
     res.json({msg: "No se encuentra el usuario para eliminar"});
   await user[0].destroy();
   res.json({msg: "Se elimino un usuario"})
  } catch (error) {
   res.json({msg: error.message})
}
```

Este código define un controlador para la gestión de usuarios. Importa el modelo Usuarios que representa la estructura de los usuarios en la base de datos. Ofrece funciones para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre estos usuarios. La función agregarUser permite agregar un nuevo usuario a la base de datos. getAllUser recupera todos los usuarios existentes. getUser obtiene un usuario específico por su ID. modificarUser actualiza la información de un usuario existente. Finalmente, eliminarUser elimina un usuario por su ID. Cada función maneja posibles errores y retorna mensajes correspondientes en caso de éxito o fallo en la operación.

```
import BD from "../config/db.js";
import { DataTypes } from "sequelize";
const Usuarios = BD.define('usuarios', {
  nombre_user :
    type: DataTypes.STRING,
    allowNull:false
  },
  edad:
    type: DataTypes.STRING,
    allowNull:false
  },
  doctor:
    type: DataTypes.STRING,
    allowNull:false
});
export default Usuarios;
```

Este código define un modelo Usuarios que representa la estructura de la tabla de usuarios en la base de datos. Utiliza Sequelize para definir los campos y tipos de datos que tendrá la tabla. Los campos incluyen nombre\_user para el nombre del usuario, edad para la edad del usuario, y doctor para el tipo de doctor. Todos los campos son requeridos (allowNull: false), lo que significa que deben tener un valor antes de poder ser almacenados en la base de datos.

routes/RoutesUsuarios.js

Copiar código

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

// Importamos nuestros controladores para nuestro nuevo mod.
import {agregarUser} from '../controller/UsuariosController.js';
import {getAllUser} from '../controller/UsuariosController.js';
import {getUser} from '../controller/UsuariosController.js';
import {modificarUser} from '../controller/UsuariosController.js';
import {eliminarUser} from '../controller/UsuariosController.js';

// Definimos las rutas para nuestro nuevo mod.
const router = express.Router();
router.post('/', agregarUser);
router.get('/', getAllUser);
router.get('/':id', getUser);
```

```
router.put('/:id', modificarUser);
router.delete('/:id', eliminarUser)

export default router;
```

Este código define las rutas para el módulo de usuarios utilizando Express. Importa las funciones del controlador UsuariosController.js para cada operación CRUD: agregar, obtener todos, obtener por ID, modificar y eliminar usuarios. Crea un router de Express y asigna las funciones correspondientes a cada ruta HTTP: POST para agregar un usuario, GET para obtener todos los usuarios y uno por ID, PUT para modificar un usuario y DELETE para eliminar un usuario. Finalmente, exporta el router para ser utilizado en la configuración principal de Express.

Y ya para finalizar con esta parte importamos las rutas de nuestro modulo a nuestro index.js, quedando éste, configurado de la siguiente manera:

src/index.js

Copiar código

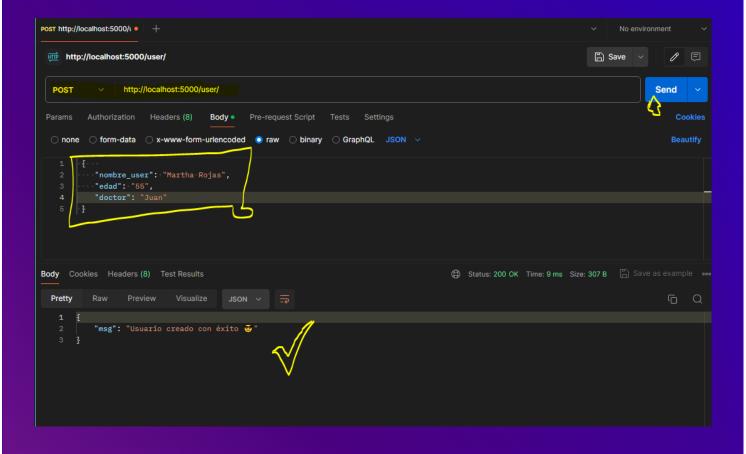
```
import express from "express";
import cors from "cors";
import BD from '../config/db.js';
import citasRoutes from '../routes/RoutesCitas.js';
import userRoutes from '.../routes/RoutesUsuarios.js';
const app = express();
app.use(cors());
app.use(express.json());
app.use('/citas', citasRoutes );
app.use('/user', userRoutes );
try {
  await BD.authenticate();
  console.log('Conexión con la base de datos exitosa. Puedes sonreír 😇');
} catch (error) {
  console.error('Unable to connect to the database:', error);
app.get('/', (req, res) => {
 res.send("Hola mundo");
```

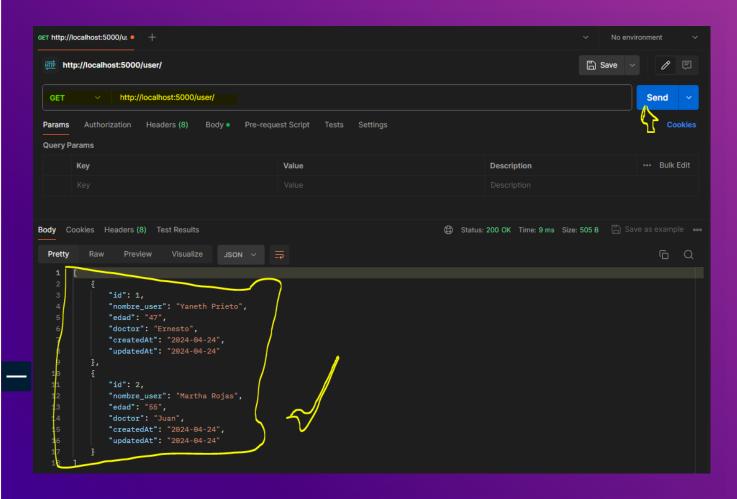
Este código configura un servidor Express que utiliza CORS para manejar solicitudes de diferentes dominios, y también middleware para analizar JSON en las solicitudes. Importa las rutas de dos módulos diferentes: RoutesCitas.js para citas y RoutesUsuarios.js para usuarios. Se autentica con la base de datos y muestra un mensaje de conexión exitosa o un error. Define una ruta principal que responde con "Hola mundo" y luego inicia el servidor en el puerto 5000, mostrando un mensaje de éxito en la consola.

### Probando funcionalidad de nuestro nuevo modulo de Usuarios

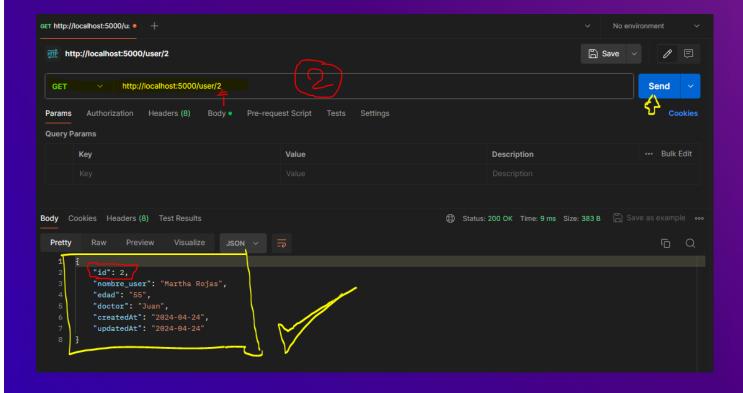
Vas a comprobar con **Postam** que nuestro nuevo modulo de Usuarios funciona de forma correcta, empezando con :

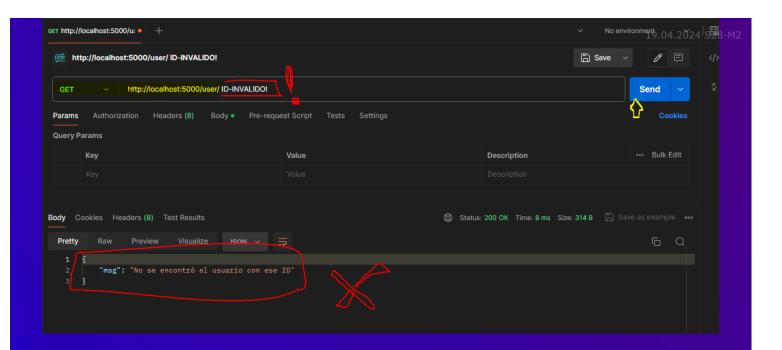
### 'agregarUser'



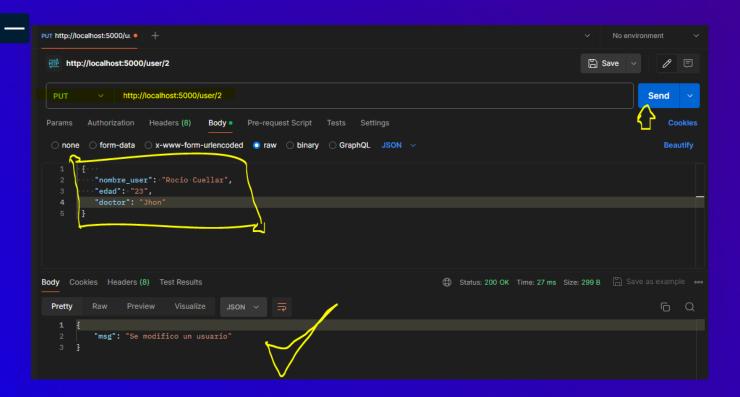


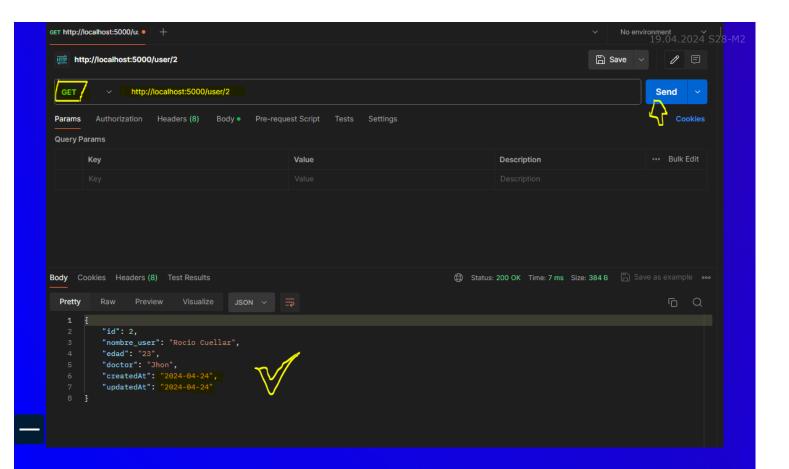
## 'getUser'



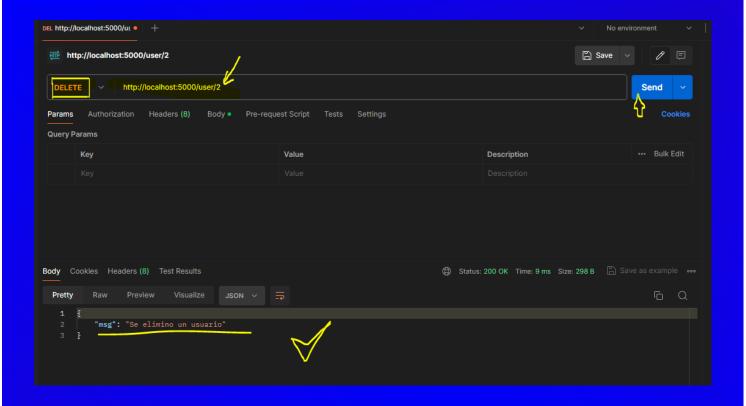


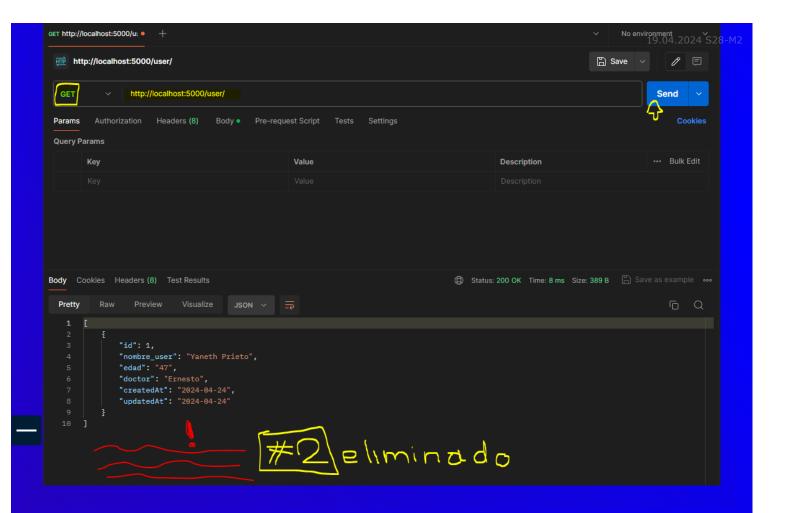
#### modificarUser





### eliminarUser





Fin de la actividad y de nuestro CRUD CON EXPRESS Y MYSQL



Mira el repositorio en Github

Bryan Hernández | Telento Tech DWFSV2-42 | 2024