


CRUD / Express y MySql - Parte 2

En esta sesión o entrega del día, 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 configuración con los cambios de hoy quedaría de la siguiente forma:

Controller/CitasController.js

 Copiar código

```
// 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});
  }
}

// Sesión 29
// Función para mostrar todas las citas
export const getAllCitas = async (req, res) =>{
  try {
    const citas = await Citas.findAll();
    res.json(citas);
  } catch (error) {
    res.json ({msg: error.message});
  }
}

// Función para mostrar un sol cliente por ID
export const getCita = async (req, res) => {
  try {
    const cita = await Citas.findAll({
      where:{id:req.params.id}
    });
    res.json(cita[0]);
  } catch (error) {
```

```

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

// Función para modificar una cita
export const modificarCita = async (req, res) =>{
  try {
    await Citas.update(req.body, {
      where:{id: req.params.id}
    })
    res.json({msg: "Se modifiko una cita"})

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

// Función para eliminar una cita
export const eliminarCita = async (req, res) => {
  try {
    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})
  }
}


```

NOTA

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`

 Copiar código

```

// 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.

Servidor: 127.0.0.1 » Base de datos: citasbd » Tabla: usuarios

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones Seg

Estructura de tabla Vista de relaciones

#	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_user	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 3	edad	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 4	doctor	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 5	createdAt	date			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/> 6	updatedAt	date			No	Ninguna			Cambiar Eliminar Más

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

```
↓ 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 serían iguales a los anteriores usados para **citas** solo que aquí se le modificarían algunos nombres de variables y algunas rutas

```
// 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 los 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});
  }
}
```

```
// Función para mostrar un usuario por ID
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})
  }
}

// Función para modificar un usuario
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 modifiko un usuario"})

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

// Función para eliminar un usuario
export const eliminarUser = async (req, res) => {
  try {
    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})
  }
}
```

NOTA

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.

```
// model/Usuarios.js
// Importamos la conexión a la BD
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;
```

NOTA

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)
```

19.04.2024 S28-M2


```
export default router;
```

NOTA

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

```
// 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';
// Importamos el archivo de las rutas de nuestro nuevo mod.
import userRoutes from '../routes/RoutesUsuarios.js';

// Definimos la variable para trabajar con Express
const app = express();

app.use(cors());
app.use(express.json());
app.use('/citas', citasRoutes );
app.use('/user', userRoutes );

// Autenticación BD
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);
}

// 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

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'

The screenshot shows the Postman interface for a POST request to `http://localhost:5000/user/`. The request body is a JSON object: `{ "nombre_user": "Martha Rojas", "edad": "55", "doctor": "Juan" }`. The response status is `200 OK` with a message: `"msg": "Usuario creado con éxito 😊"`. A yellow checkmark is drawn next to the response.

Request Details:

- Method: POST
- URL: `http://localhost:5000/user/`
- Body Type: raw (JSON)
- Body Content:

```
{
  "nombre_user": "Martha Rojas",
  "edad": "55",
  "doctor": "Juan"
}
```

Response Details:

- Status: 200 OK
- Time: 9 ms
- Size: 307 B
- Body Content:

```
{
  "msg": "Usuario creado con éxito 😊"
}
```


'getAllUser'

19.04.2024 S28-M2

GET http://localhost:5000/user/

Send

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

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 9 ms Size: 505 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "nombre_user": "Yaneth Prieto",
5     "edad": "47",
6     "doctor": "Ernesto",
7     "createdAt": "2024-04-24",
8     "updatedAt": "2024-04-24"
9   },
10  {
11    "id": 2,
12    "nombre_user": "Martha Rojas",
13    "edad": "55",
14    "doctor": "Juan",
15    "createdAt": "2024-04-24",
16    "updatedAt": "2024-04-24"
17  }
18 ]
```

'getUser'

GET http://localhost:5000/user/2

Send

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

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 9 ms Size: 383 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "nombre_user": "Martha Rojas",
4   "edad": "55",
5   "doctor": "Juan",
6   "createdAt": "2024-04-24",
7   "updatedAt": "2024-04-24"
8 }
```

GET http://localhost:5000/user/ID-INVALIDO! 19.04.2024 5:28-M2

http://localhost:5000/user/ID-INVALIDO!

GET http://localhost:5000/user/ID-INVALIDO! Send

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

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 8 ms Size: 314 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "No se encontró el usuario con ese ID"
3 }
```

modificarUser

PUT http://localhost:5000/user/2

PUT http://localhost:5000/user/2 Send

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

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

```
1 {
2   "nombre_user": "Rocio Cuellar",
3   "edad": "23",
4   "doctor": "Jhon"
5 }
```

Body Cookies Headers (8) Test Results Status: 200 OK Time: 27 ms Size: 299 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Se modifiko un usuario"
3 }
```

GET http://localhost:5000/user/2

Save Send

GET http://localhost:5000/user/2

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 7 ms Size: 384 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "nombre_user": "Rocio Cuellar",
4   "edad": "23",
5   "doctor": "Jhon",
6   "createdAt": "2024-04-24",
7   "updatedAt": "2024-04-24"
8 }
```

eliminarUser

DEL http://localhost:5000/user/2

Save Send

DELETE http://localhost:5000/user/2

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 9 ms Size: 298 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Se elimino un usuario"
3 }
```

GET http://localhost:5000/user/

Save Send

GET http://localhost:5000/user/ Send

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 8 ms Size: 389 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "nombre_user": "Yaneth Prieto",
5     "edad": "47",
6     "doctor": "Ernesto",
7     "createdAt": "2024-04-24",
8     "updatedAt": "2024-04-24"
9   }
10 ]
```

#2 eliminado

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



Mira el repositorio en Github

Bryan Hernández | Telento Tech DWFSV2-42 | 2024