<codoa codo/>

Agencia de Aprendizaje a lo largo de la vida

# FULL STACK
# Clase 29

Node 5

BA

# Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 28

Clase 29

Clase 30

**Node**

- Express Router
  Postman
  GET
  POST
  PUT
  DELETE

**Node**

- Modulo mysql2
  Estructura del proyecto
  Controladores
  Altas
  Modificaciones
  Bajas

**Node**

# NodeJS

## Conectando con la Base de Datos

# Conectar con la base de datos

Como hemos visto con anterioridad, guardar datos en una base de datos al trabajar en el backend es crucial por varias razones: Persistencia de Datos, Acceso Eficiente y Rápido, Seguridad, Gestión de Concurrencia, Flexibilidad y Escalabilidad, Recuperación ante Fallos entre otros.

En resumen, almacenar datos en una base de datos proporciona un entorno seguro, eficiente y escalable para gestionar la información, lo cual es esencial para el funcionamiento confiable y sostenible de aplicaciones backend.

El módulo mysql2 para Node.js es una biblioteca que permite interactuar con bases de datos MySQL y MariaDB desde aplicaciones Node.js. Está diseñado para ser rápido, compatible con el protocolo MySQL, y fácil de usar en combinación con frameworks como Express.

# Instalación en el proyecto

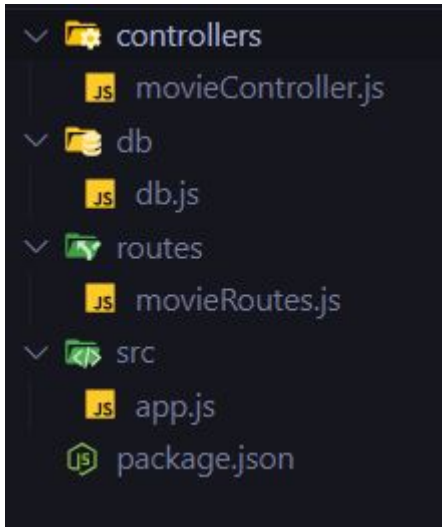
Para poder instalar mysql en nuestro proyecto, simplemente debemos ejecutar el siguiente comando:

```
npm install mysql2
```

**Importante:** Para que funcione correctamente el paquete de mysql2 tiene que estar instalado MySQL Server en la computadora local y debe estar en funcionamiento, cómo lo hemos visto en las clases de bases de datos.

# Estructura del proyecto

Reestructuremos el proyecto de la clase pasada de la siguiente manera:



# Archivo app.js

El archivo que comienza todo el proceso:

src > app.js > ...

```
1  // src/app.js
2  const express = require('express');
3  const movieRoutes = require('../routes/movieRoutes');
4
5  const app = express();
6
7  app.use(express.json());
8  app.use('/movies', movieRoutes);
9
10 const PORT = 3000;
11
12 app.listen(PORT, () => {
13     console.log(`Server is running on port ${PORT}`);
14 });
```

La declaración del servidor se mantiene de la misma manera.

Este archivo utiliza el módulo **movieRoutes**

# Archivo movieRoutes.js

Vamos a quitar toda la lógica y tratamiento de datos que teníamos en este archivo.Dejaremos sólo la declaración de las rutas, con sus métodos y el llamado al **movieController con el método específico para cada opción.**

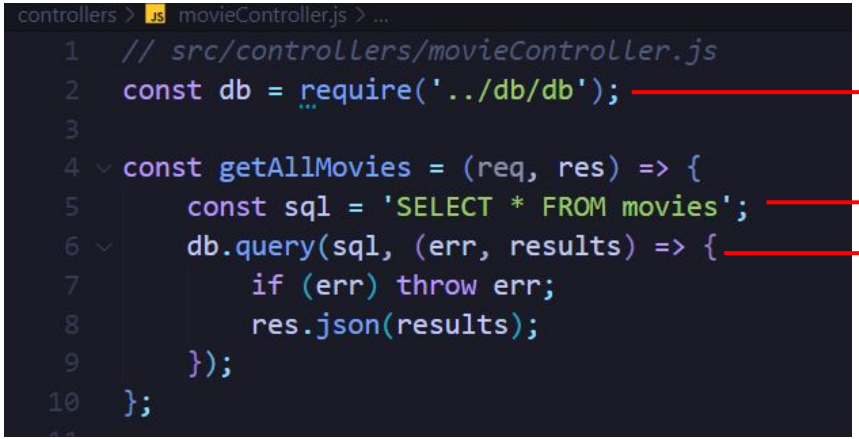
routes > movieRoutes.js > ...

```
1  // src/routes/movieRoutes.js
2  const express = require('express');
3  const router = express.Router();
4  const movieController = require('../controllers/movieController');
5
6  router.get('/', movieController.getAllMovies);
7  router.get('/:id', movieController.getMovieById);
8  router.post('/', movieController.createMovie);
9  router.put('/:id', movieController.updateMovie);
10 router.delete('/:id', movieController.deleteMovie);
11
12 module.exports = router;
```

De esta manera simplificamos y ordenamos nuestro código. Haciendo más fácil el mantenimiento posterior.

# Archivo movieController.js

El controlador es el que tendrá los cambios más importantes y es el que hará el tratamiento de la información.
El primer controlador que haremos será **getAllMovies** que devolverá todas las películas cargadas en la base de datos.



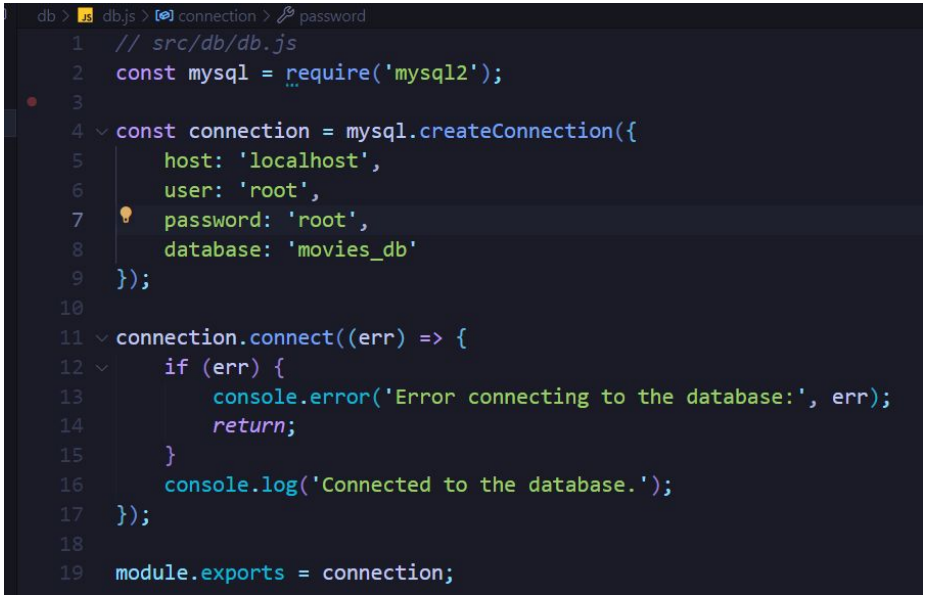

El objeto **db** posee los métodos para conectar con la base de datos. Es la conexión a la base de datos.

Generamos el string que se utilizará en la base de datos.

Utilizamos el método query del objeto db que recibe por parámetro la consulta que generamos y una función de flecha que tendrá el resultado de la consulta en formato json.

# Archivo db.js

Finalmente el archivo db.js será el que cree el objeto que conecta con la base de datos. Esa conexión utilizará el objeto mysql provisto en en el módulo **mysql2**

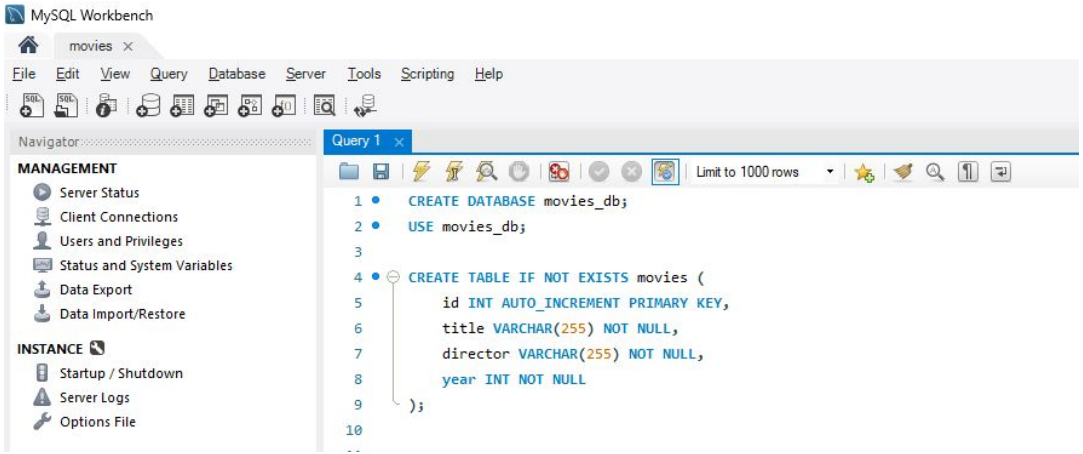



Al utilizar el método **createConnection** debemos indicar el host, user, password y base de datos que deberá utilizar el objeto para conectar con la base de datos.

***Si estos datos son incorrectos, no será posible acceder a la base de datos.***

# Creación de la base de datos

Para hacer la prueba a la base datos, primeramente tendremos que crearla si no existe. Podemos hacerlo dirctamente a través del Workbench de Mysql, por ejemplo:



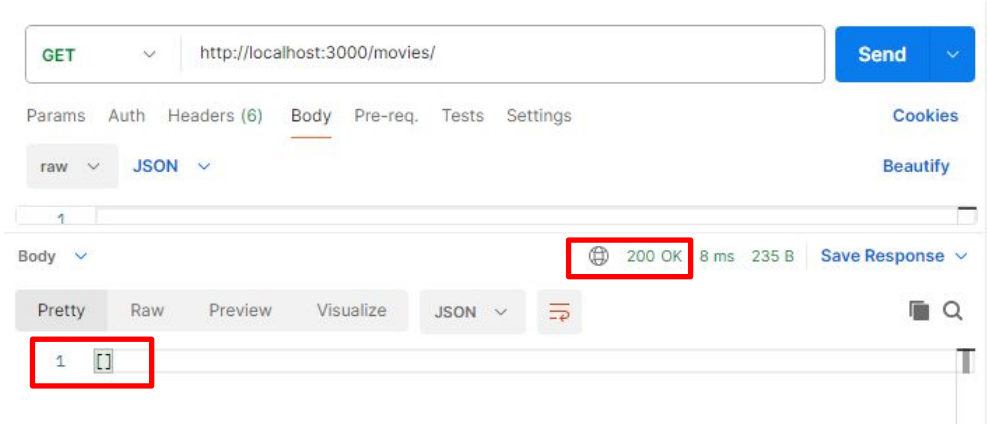
# Levantando el proyecto

Ahora sí, deberemos levantar el Proyecto en Visual Studio y si todo es correcto, recibiremos el siguiente mensaje:

```
> movies@1.0.0 start
> node src/app.js

Server is running on port 3000
Connected to the database.
```

# Probando desde Postman:

Ahora, al ejecutar la petición GET movies desde Postman, recibiremos un **Status 200 Ok**, con un **array vacío**, porque la tabla no tiene registros. Si ingresamos manualmente datos a través de MySql, vamos a obtener el JSON de respuesta de la misma forma en que obteníamos los datos cuando estaban cargados en el array de películas en memoria.



# Continúamos con el desarrollo de los controladores

# Parámetros en la consulta

Vamos a trabajar ahora en el controlador **getMovieById**, en el cual recibimos como entrada, el valor del **id** de la película que queremos recuperar de la base de datos.

```
12  const getMovieById = (req, res) => {
13      const { id } = req.params;
14      const sql = 'SELECT * FROM movies WHERE id = ?';
15      db.query(sql, [id], (err, result) => {
16          if (err) throw err;
17          res.json(result);
18      });
19  };
```

El **?** es un marcador de posición que será reemplazado por el valor de **id** para evitar inyecciones SQL.

La inyección de SQL (SQL Injection) es una técnica de ataque que explota una vulnerabilidad en una aplicación que interactúa con una base de datos. Este tipo de ataque permite a un atacante ejecutar comandos SQL arbitrarios en la base de datos, lo que puede llevar a la exposición, manipulación o eliminación de datos sensibles.

# Alta de película

Crearemos el controlador **createMovie** para realizar el alta de la película.

```
21  const createMovie = (req, res) => {
22      const { title, director, year } = req.body;
23      const sql = 'INSERT INTO movies (title, director, year) VALUES (?, ?, ?)';
24      db.query(sql, [title, director, year], (err, result) => {
25          if (err) throw err;
26          res.json({ message: 'Movie created', movieId: result.insertId });
27      });
28  };
```

Nuevamente, a través de los marcadores de posiciones evitamos la inyección de código malicioso.

Si quisiéramos hacer algún control respecto a los datos recibidos, este es el lugar donde podríamos desarrollarlo y tomar la decisión de grabar en la base de datos o devolver un error al usuario.

# Modificación de película

Crearemos el controlador **updateMovie** para realizar la modificación.

```
30  const updateMovie = (req, res) => {
31      const { id } = req.params;
32      const { title, director, year } = req.body;
33      const sql = 'UPDATE movies SET title = ?, director = ?, year = ? WHERE id = ?';
34      db.query(sql, [title, director, year, id], (err, result) => {
35          if (err) throw err;
36          res.json({ message: 'Movie updated' });
37      });
38  };
```

Al igual que en el alta, si quisieramos hacer algún control respecto a los datos recibidos, este es el lugar donde podríamos desarrollarlo y tomar la decision de grabar en la base de datos o devolver un error al usuario.

# Eliminación de película

Crearemos el controlador **deleteMovie** para realizar la eliminación del dato en la base. En este caso hemos optado por un borrado físico de la información, pero podríamos haber optado por un borrado lógico.

```
40  const deleteMovie = (req, res) => {
41      const { id } = req.params;
42      const sql = 'DELETE FROM movies WHERE id = ?';
43      db.query(sql, [id], (err, result) => {
44          if (err) throw err;
45          res.json({ message: 'Movie deleted' });
46      });
47  };
```

# movieController.js

controllers > movieController.js > ...

```
// src/controllers/movieController.js
const db = require('../db/db');

const getAllMovies = (req, res) => {
    const sql = 'SELECT * FROM movies';
    db.query(sql, (err, results) => {
        if (err) throw err;
        res.json(results);
    });
};

const getMovieById = (req, res) => {
    const { id } = req.params;
    const sql = 'SELECT * FROM movies WHERE id = ?';
    db.query(sql, [id], (err, result) => {
        if (err) throw err;
        res.json(result);
    });
};

const createMovie = (req, res) => {
    const { title, director, year } = req.body;
    const sql = 'INSERT INTO movies (title, director, year) VALUES (?, ?, ?)';
    db.query(sql, [title, director, year], (err, result) => {
        if (err) throw err;
        res.json({ message: 'Movie created', movieId: result.insertId });
    });
};
```

controllers > movieController.js > ...

```

const updateMovie = (req, res) => {
    const { id } = req.params;
    const { title, director, year } = req.body;
    const sql = 'UPDATE movies SET title = ?, director = ?, year = ? WHERE id = ?';
    db.query(sql, [title, director, year, id], (err, result) => {
        if (err) throw err;
        res.json({ message: 'Movie updated' });
    });
};

const deleteMovie = (req, res) => {
    const { id } = req.params;
    const sql = 'DELETE FROM movies WHERE id = ?';
    db.query(sql, [id], (err, result) => {
        if (err) throw err;
        res.json({ message: 'Movie deleted' });
    });
};

module.exports = {
    getAllMovies,
    getMovieById,
    createMovie,
    updateMovie,
    deleteMovie
};
```

# Ejecutando el Código por primera vez

Si es la primera vez que estamos ejecutando nuestro código y no temenos creada la base de datos, tendremos un error que no podremos manejar a menos que manualmente realizamos el **CREATE TABLE**.

Para resolver esa situación, podemos modificar nuestro archivo **db.js** para que en el intento de conectar con la base de datos, cree la base y las tablas en caso que no existan. De esta manera nos aseguramos que aunque borremos la base de datos en MySQL, **la misma se creará automáticamente cuando se ejecute la aplicación**.

# Actualización de db.js

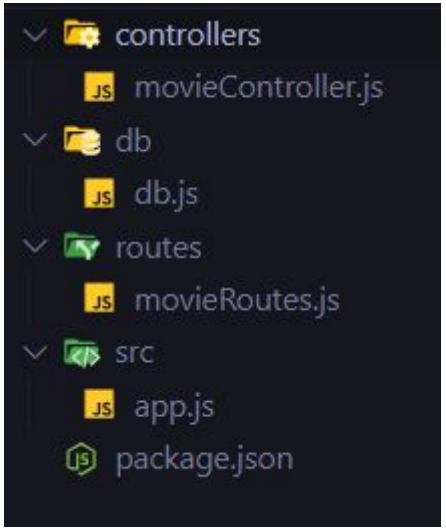
db > db.js > ...

```
// src/db/db.js
const mysql = require('mysql2');

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'root'
});

connection.connect((err) => {
    if (err) {
        console.error('Error connecting to the database:', err);
        return;
    }
    console.log('Connected to the database.');

    connection.query('CREATE DATABASE IF NOT EXISTS movies_db', (err, results) => {
        if (err) {
            console.error('Error creating database:', err);
            return;
        }
        console.log('Database ensured.');

        connection.changeUser({ database: 'movies_db' }, (err) => {
            if (err) {
                console.error('Error switching to movies_db:', err);
                return;
            }
```

```
            const createTableQuery = `
                CREATE TABLE IF NOT EXISTS movies (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    title VARCHAR(255) NOT NULL,
                    director VARCHAR(255) NOT NULL,
                    year INT NOT NULL
                );
            `;
            connection.query(createTableQuery, (err, results) => {
                if (err) {
                    console.error('Error creating table:', err);
                    return;
                }
                console.log('Table ensured.');
            });
        });
    });
});

module.exports = connection;
```

# Estructura del proyecto

Al tener el Proyecto estructurado en archivos y carpetas, es más fácil realizar el mantenimiento y seguimiento del Proyecto.




Por ejemplo:

Si cambia la contraseña o la base de datos, deberemos modificar solamente el archive **db.js**

Si queremos agregar un tratamiento de datos para usuarios, agregaremos en un archivo **userController.js** los controladores del usuario, en **userRoutes.js** las rutas que podrán ser llamadas desde el frontend y en **app.js** incorporaremos el modulo de userRoutes.

# No te olvides de dar el presente

# Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizá los ejercicios obligatorios.

Todo en el Aula Virtual.