

API RESTFull con Express

Bases de Datos

Primero comenzamos con la creación de la Base de datos:

```
-- Creación de la Base de Datos
CREATE DATABASE tarea_08;
USE tarea_08;

-- Creación de las tablas
CREATE TABLE peliculas(
  id          INT PRIMARY KEY AUTO_INCREMENT,
  nombre      VARCHAR(45) NOT NULL,
  sinopsis     VARCHAR(300) NOT NULL,
  genero      VARCHAR(50) NOT NULL,
  director    VARCHAR(45) NOT NULL,
  calificacion ENUM('1', '2', '3', '4', '5'),
  duracion    TINYINT UNSIGNED NOT NULL COMMENT 'La duracion se esta registrando en minutos'
)ENGINE = INNODB;

-- Insercción de registros
INSERT INTO peliculas(nombre, sinopsis, genero, director, calificacion, duracion) VALUES
('Avengers: Infinity War',
'Un nuevo peligro acecha procedente de las sombras del cosmos.
Thanos, el infame tirano intergaláctico, tiene como objetivo reunir las seis Gemas del Infinito,
artefactos de poder inimaginable, y usarlas para imponer su perversa voluntad a toda la existencia.',
'Superhéroes',
'Anthony Russo y Joe Russo',
'4',
156);

-- Consulta para verificar que los datos se insertarán
SELECT * FROM peliculas;
```

Instalación y Configuración

Con la Base de datos lista, continuamos con la inicialización de nuestro proyecto en node ejecutando el siguiente comando en la raíz de nuestro proyecto:

npm init

seguimos con la instalación de los módulos necesarios para nuestra app, en este caso serían los siguiente: mysql2, express, y dotenv (para las variables de entorno).

Ejecutamos el siguiente comando:

npm install mysql2 express dotenv nodemon

Seguido creamos un archivo “.env” en la raíz de nuestro proyecto para las variables de entorno y colocamos los siguientes campos:

```
1 DB_HOST=localhost
2 DB_USER=root
3 DB_PASSWORD=
4 DB_DATABASE= tarea_08
5 DB_PORT=3306
```

Db.js

Luego en la raíz de nuestro proyecto creamos una carpeta llamada “config/” que guardara el archivo del pool de conexiones llamado “db.js”:

```
config > db.js > ...
1  /**
2  | * Usamos este archivo para crear un pool de conexiones y reutilizarlas
3  | */
4
5  //Accedemos al archivo .env para obtener las variables de entorno y establecer la conexión con la BD
6  require('dotenv').config()
7
8  //Referenciamos un modulo de mysql2 para el manejo de promesas(una acción que tardara un tiempo en resolverse)
9  const mysql = require('mysql2/promise')
10
11
12  //Pool de conexiones = acceso
13  const pool = mysql.createPool({
14    host: process.env.DB_HOST,
15    user: process.env.DB_USER,
16    password: process.env.DB_PASSWORD,
17    database: process.env.DB_DATABASE,
18    port: process.env.DB_PORT
19  })
20
21  //Aprovechar el recurso en otra parte de la App
22  module.exports = pool
```

Controller

Ahora seguimos con la carpeta que guardara la lógica de nuestro proyecto. En la raíz de nuestra app creamos la carpeta “controllers/” dentro creamos un controlador para cada tabla. En este caso solo tenemos la tabla películas.

En el controlador se creará un método para cada método HTTP (GET, POST, PUT, DELETE)

```

controllers > peliculaController.js > ...
1  /**
2  | * Este archivo tendra toda la logica de nuestra aplicación(crear, leer, modificar y eliminar)
3  | */
4
5  //Primero obtenemos el pool de conexiones de nuestro archivo db.js
6  const db = require('../config/db')
7
8  //Esta objeto nos servirar para hacer validaciones
9  > const data = {...
34  }
35
36  //Funcion para validar los campos del body
37  > function campoEsValido(dataFormat, campo){...
48  }
49
50  //Obtener todas las peliculas registradas
51  > exports.getAllPeliculas = async (req, res)⇒{...
68  }
69
70  //Obtener una pelicula por ID
71  > exports.getPeliculaById = async (req, res)⇒{...
96  }
97
98  //Crear una pelicula
99  > exports.createPelicula = async (req, res)⇒{...
154  }
155
156  //Editar una pelicula
157  > exports.editPelicula = async (req, res) ⇒ {...
180  }
181
182  //Eliminar una pelicula por ID
183  > exports.deletePelicula = async(req, res)⇒{...
208  }
209
210

```

GET:

```

//Obtener todas las peliculas registradas
exports.getAllPeliculas = async (req, res)⇒{
  //Creamos la instruccion sql para obtener los datos de la tabla peliculas
  const SQL = 'SELECT * FROM peliculas'
  try {
    const [result] = await db.query(SQL)
    if(result.length === 0){
      return res.status(402).json({
        mensaje: 'No se encotraron registros'
      })
    }
    return res.status(200).json(result)
  } catch (e) {
    console.error(e)
    return res.status(500).json({
      error: e
    })
  }
}

```

POST:

```
97 //Crear una película
98 exports.createPelícula = async (req, res) => {
99   //Obtenemos los valores que nos pasarán en el JSON mediante el body.
100   const { nombre, sinopsis, genero, director, calificacion, duracion } =
101     req.body;
102
103   const SQL =
104     "INSERT INTO peliculas(nombre, sinopsis, genero, director, calificacion, duracion) VALUES (?, ?, ?, ?, ?, ?)";
105
106   //Validaciones para los campos
107   if (!campoEsValido(data.nombre, nombre)) {
108     return res
109       .status(400)
110       .json({
111         mensaje: `El campo nombre no puede tener mas de ${data.nombre.maxLength} caracteres ni estar vacio`,
112       });
113   }
114   if (!campoEsValido(data.sinopsis, sinopsis)) {
115     return res
116       .status(400)
117       .json({
118         mensaje: `El campo sinopsis no puede tener mas de ${data.sinopsis.maxLength} caracteres ni estar vacio`,
119       });
120   }
121   if (!campoEsValido(data.genero, genero)) {
122     return res
123       .status(400)
124       .json({
125         mensaje: `El campo genero no puede tener mas de ${data.genero.maxLength} caracteres ni estar vacio`,
126       });
127   }
128   if (!campoEsValido(data.director, director)) {
129     return res
130       .status(400)
131       .json({
132         mensaje: `El campo director no puede tener mas de ${data.director.maxLength} caracteres ni estar vacio`,
133       });
134   }
135   if (!campoEsValido(data.calificacion, calificacion)) {
136     return res
137       .status(400)
138       .json({
139         mensaje: `El campo calificación solo puede tener estos valores: 1, 2, 3, 4, 5`,
140       });
141   }
142   if (!campoEsValido(data.duracion, duracion)) {
143     return res
144       .status(400)
145       .json({
146         mensaje: `El campo duración no puede ser mayor que ${data.duracion.maxValue} ni estar vacio`,
147       });
148   }
```

```

150  try {
151      //Validaciones para campos vacios
152      if (
153          !nombre ||
154          !sinopsis ||
155          !genero ||
156          !director ||
157          !calificacion ||
158          !duracion
159      ) {
160          return res.status(500).json({
161              mensaje: "Ningun campo puede estar vacio",
162          });
163      }
164      // if(comprobarTipo(nombre, tipo)  typeof(nombre) !== 'string' || typeof
165
166      //Ejecutamos la consulta y guardamos los resultados en una constante.
167      const [result] = await db.query(SQL, [
168          nombre,
169          sinopsis,
170          genero,
171          director,
172          calificacion,
173          duracion,
174      ]);

```

```

175
176      if (result.affectedRows === 0) {
177          return res.status(500).json({
178              mensaje: "No se logro insertar los datos",
179          });
180      }
181      return res.status(200).json({
182          mensaje: "Se registro correctamente",
183          pk: result.insertId,
184      });
185  } catch (e) {
186      console.error(e);
187      return res.status(500).json({
188          error: e,
189      });
190  }
191  };

```

PUT:

```

193 //Editar una pelicula
194 exports.editPelicula = async (req, res) => {
195   //Obtenemos el id que nos pasarón en la URL
196   const { id } = req.params;
197
198   //Obtenemos los valores del JSON que nos pasarón por el body.
199   const { nombre, sinopsis, genero, director, calificacion, duracion } =
200     req.body;
201
202   const SQL =
203     "UPDATE peliculas SET nombre=?, sinopsis=?, genero=?, director=?, calificacion=?, duracion=? WHERE id=?";
204
205   try {
206     //Ejecutamos la consulta y lo guardamos el resultado en una constante.
207     const [result] = await db.query(SQL, [
208       nombre,
209       sinopsis,
210       genero,
211       director,
212       calificacion,
213       duracion,
214       id,
215     ]);
216
217     if (result.affectedRows === 0) {
218       return res.status(404).json({ mensaje: "No se logro Actualizar" });
219     }
220
221     return res.status(200).json({
222       mensaje: "La pelicula se actualizo correctamente",
223     });
224   } catch (e) {
225     return res.status(500).json({ error: e });
226   }
227 }

```

DELETE:

```

228 //Eliminar una pelicula por ID
229 exports.deletePelicula = async (req, res) => {
230   //Obtenemos el id por la URL.
231   const { id } = req.params;
232
233   //Preparamos la consulta
234   const SQL = "DELETE FROM peliculas WHERE id=?";
235
236   try {
237     //Ejecutamos la consulta.
238     const [result] = await db.query(SQL, [id]);
239
240     if (result.affectedRows === 0) {
241       return res.status(404).json({
242         mensaje: "No se logro encontrar ningun registro con este id",
243       });
244     }
245
246     return res.status(200).json({
247       mensaje: "Se elimino correctamente el registro",
248     });
249   } catch (e) {
250     return res.status(500).json({
251       error: e,
252     });
253   }
254 };


```

Routes

Ya con la funcionalidad lista, debemos de configurar las rutas para llamar a cada objeto del controlador peliculaController. Creamos la carpeta: **"routes/"** en la raíz de nuestra app que será para el manejo de rutas con sus métodos HTTP y los objetos de la clase peliculaController (getAllPeliculas, getPeliculaById, createPelicula, editPelicula, deletePelicula).

Dentro de ella creamos un archivo para el manejo de rutas del controlador para nuestra tabla. En este caso peliculaRouter.

```

routes >  peliculaRouter.js > ...
1  /**
2   * Este archivo es para definir las rutas, metodos HTTP y lo que hara cada ruta
3   */
4
5   //cargamos un objeto express para acceder a sus metodos y objetos
6   const express = require('express')
7
8   //Llamamos al un objeto de express para definir las rutas
9   const router = express.Router()
10
11  //Instanciamos un objeto con nuestro clase peliculaController
12  const peliculaController = require('../controllers/peliculaController')
13
14
15  //Definimos la ruta con el metodo al que llamaremos cuando entremos a la ruta
16
17  // objeto router | metodo HTTP | metodo que llamamos al acceder a la ruta
18  router.get('/', peliculaController.getAllPeliculas)
19  router.get('/:id', peliculaController.getPeliculaById)
20  router.post('/', peliculaController.createPelicula)
21  router.put('/:id', peliculaController.editPelicula)
22  router.delete('/:id', peliculaController.deletePelicula)
23
24  //Exportamos el objeto router
25  module.exports = router

```

Server.js

Por último, creamos el archivo “**server.js**” en la raíz de nuestro proyecto. Este archivo será el que conectará todo lo que hicimos anteriormente, también será el encargado de iniciar nuestra aplicación:


```
server.js > ...
1  /**
2   * Este archivo levanta nuestra aplicación
3   */
4
5  const express = require('express')
6
7  //Llamamos a los enrutadores
8  const peliculaRouter = require('./routes/peliculaRouter')
9
10
11  const app = express()
12  const PORT = process.env.PORT || 3000 //puerto para la APP
13
14  //Definimos la comunicación en JSON
15  app.use(express.json())
16
17
18  //Rutas para las API's
19  app.use('/api/peliculas', peliculaRouter)
20
21  //iniciamos el servidor
22  app.listen(PORT, ()=>{
23    console.log(`Servidor iniciado http://localhost:${PORT}`)
24  })
```

Pruebas

Ahora para comprobar el funcionamiento del proyecto lo ejecutamos con el siguiente comando:

```
nodemon server
```

Ahora probamos el método GET:

HTTP Save

GET http://localhost:3000/api/peliculas Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (7) Test Results

200 OK • 42 ms • 4.4 KB

JSON Preview Visualize

```
1 [
2   {
3     "id": 1,
4     "nombre": "Avengers: Infinity War",
5     "sinopsis": "Un nuevo peligro acecha procedente de las sombras del cosmos. \nThanos, el infame tirano intergaláctico, tiene
6     como objetivo reunir las seis Gemas del Infinito, \nartefactos de poder inimaginable, y usarlas para imponer su
7     perversa voluntad a toda la existencia.",
8     "genero": "Superhéroes",
9     "director": "Anthony Russo y Joe Russo",
10    "calificacion": "4",
11    "duracion": 156
12  },
13 ]
```

POST:

HTTP Save

POST http://localhost:3000/api/peliculas Send

Params Authorization Headers (8) Body Scripts Settings Cookies

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

Beautify

```
1 {
2   "nombre": "Avengers EndGame",
3   "sinopsis": "La historia sigue a los vengadores supervivientes mientras intentan deshacer el daño causado por por Thanos en
4   'Avengers: Infinity War', quien logró reunir las seis Gemas del Infinito y eliminar a la mitad de la vida en el universo.",
5   "genero": "Ciencia Ficción",
6   "director": "Anthony Russo y Joe Russo",
7   "calificacion": 4,
8   "duracion": "182"
9 }
```

Body Cookies Headers (7) Test Results

200 OK • 55 ms • 282 B •

JSON Preview Visualize

```
1 {
2   "mensaje": "Se registro correctamente",
3   "pk": 13
4 }
```

PUT:

HTTP <http://localhost:3000/api/peliculas/13> Save

PUT <http://localhost:3000/api/peliculas/13> Send

Params Authorization Headers (8) **Body** Scripts Settings Cookies

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

```
1 {
2   "nombre": "Avengers EndGame",
3   "sinopsis": "Soy una sinopsis Editada",
4   "genero": "Soy un genero editado",
5   "director": "Anthony Russo y Joe Russo",
6   "calificacion": 4,
7   "duracion": "182"
8 }
```

Body Cookies Headers (7) Test Results 200 OK 50 ms 287 B ⌕ ⋮

{} JSON Preview Visualize ⌕ ⌂ 🔍 📄 🔗

```
1 {
2   "mensaje": "La pelicula se actualizo correctamente"
3 }
```

GET By ID:

HTTP <http://localhost:3000/api/peliculas/13> Save

GET <http://localhost:3000/api/peliculas/13> Send

Params Authorization Headers (8) **Body** Scripts Settings Cookies

Query Params

Key	Value	Description	⋮	Bulk Edit
-----	-------	-------------	---	-----------

Body Cookies Headers (7) Test Results 200 OK 11 ms 419 B ⌕ ⋮

{} JSON Preview Visualize ⌕ ⌂ 🔍 📄 🔗

```
1 [
2   {
3     "id": 13,
4     "nombre": "Avengers EndGame",
5     "sinopsis": "Soy una sinopsis Editada",
6     "genero": "Soy un genero editado",
7     "director": "Anthony Russo y Joe Russo",
8     "calificacion": "4",
9     "duracion": 182
10  }
11 ]
```

DELETE:

HTTP <http://localhost:3000/api/peliculas/13> Save

DELETE <http://localhost:3000/api/peliculas/13> Send

Params Authorization Headers (8) **Body** Scripts Settings Cookies

Query Params

Key	Value	Description	⋮	Bulk Edit
-----	-------	-------------	---	-----------

Body Cookies Headers (7) Test Results 200 OK 176 ms 285 B ⌕ ⋮

{} JSON Preview Visualize ⌕ ⌂ 🔍 📄 🔗

```
1 {
2   "mensaje": "Se elimino correctamente el registro"
3 }
```