«Talento Tech»

Desarrollo Web 4

Clase 05

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 5 | Check point

Temario:

- Trabajo integrador para consolidar los contenidos adquiridos hasta el momento.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

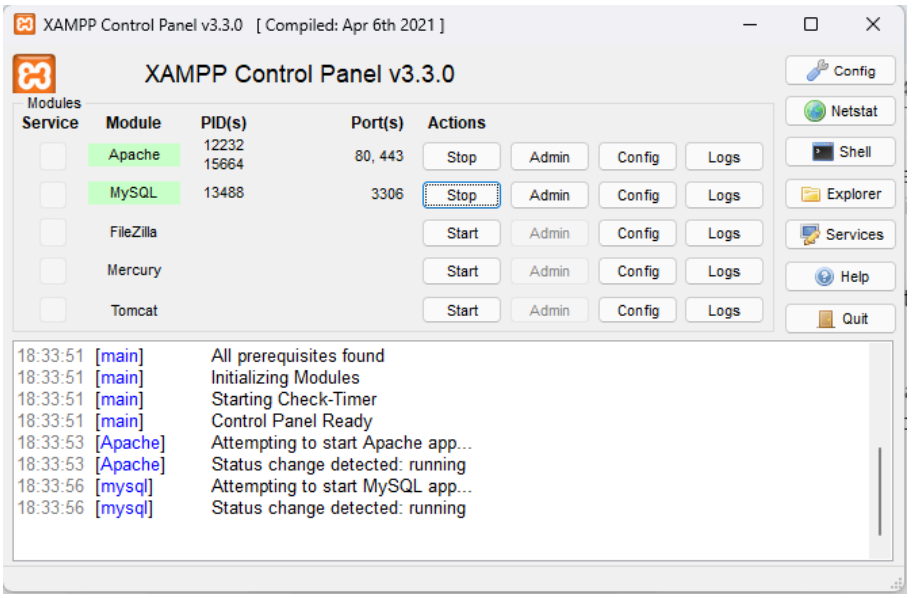
Repaso de los contenidos vistos

Hasta el momento aprendimos cómo inicializar un proyecto en node js, aprendimos a crear un servidor en express js y a su vez a consumir una base de datos **SQL**. Ahora vamos a poner todos esos conocimientos en práctica para poder llevarlos a cabo.

Para ello vamos a hacer una rest api que sea de videojuegos. Es decir un sistema que nos permite crear, leer y eliminar juegos y a su vez esas acciones realizarlas en una base de datos relacional (**SQL**).

Iniciando Proyecto

⚠ Es importante tener instalado Xampp y tener seteada la configuración en xampp para poder conectar todo.



Recuerden de tener inicializado **Apache** (primero) y **mysql** después. Solo tienen que apretar start en el primer botón donde dice Actions y cuando les aparezca tanto apache como mysql en verde implica que ya está funcionando correctamente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Iniciando el Servidor

- Vamos a crear una carpeta y abrir visual studio code en esa carpeta.
- Luego agregamos algunos comandos por consola para poder empezar a setear nuestro proyecto.
- Creamos dentro de esa carpeta un archivo que sea **index.js**

Los primeros dos comandos son:

Para montar nuestro archivo package.json

```
npm init -y
```

Para utilizar las librerías que nos van a permitir desarrollar nuestro sistema

```
npm install express mysql
```

Ahora realizamos las importaciones de nuestras librerías. En este caso vamos a implementar **require** para realizar las importaciones pero ustedes pueden usar **import** modificando la configuración en package.json agregando a la configuración la opción **“type”: “module”**

Importación de las librerías

Con express vamos a crear el ambiente para el servidor:

```
const express = require('express');
```

Con sql vamos a crear la base de datos:

```
const mysql = require('mysql');
```

Comenzamos creando nuestro entorno en express y luego creamos una variable la cual almacenará el número de puerto al que debe ir:

```
const app = express();
const port = 3000;
```

Configuramos y conectamos nuestra DB

Debajo vamos a establecer las configuraciones de la base de datos creando el siguiente objeto :

```
const db_config = {
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'phpmyadmin'
}
```

Este objeto va a ser pasado por parámetro para el método createConnection de la librería mysql. Esto lo que hace es poder crear la conexión con la base de datos:

```
const db = mysql.createConnection(db_config);
```

Una vez almacenando en una variable la conexión vamos a prevenir errores dado que la misma podría no ser exitosa y para poder conocer bien por que pueden suceder estos errores debemos **catchearlos.**

```
db.connect((err) => {
```

```
  if (err) {
    console.error('Error de conexión a la base de datos:', err);
  } else {
    console.log('Conexión exitosa a la base de datos');
  }
});
```

Middleware para json

Debajo vamos a establecer un middleware que nos permitirá admitir solicitudes en formato de tipo json:

```
app.use(express.json());
```

Creación de la tabla

A continuación debemos crear la tabla por si no existe y la crearemos de la siguiente forma:

```
function createTableIfNotExists() {
  const createTableQuery = `
    CREATE TABLE IF NOT EXISTS videojuegos (
      id INT AUTO_INCREMENT PRIMARY KEY,
      nombre VARCHAR(255) NOT NULL,
      genero VARCHAR(255) NOT NULL,
      plataforma VARCHAR(255) NOT NULL
    )
  `;

  db.query(createTableQuery, (err) => {
    if (err) {
      console.error('Error al crear la tabla de videojuegos:', err);
    } else {
```

```
      console.log('Tabla de videojuegos creada correctamente');
    }
  });
}

createTableIfNotExists();
```

createTableQuery viene a ser la solicitud en lenguaje **SQL** que enviamos a la base de datos para que se cree la tabla en el caso de que no esté creada. Esto obviamente que se va a crear una sola vez pero es importante que lo tengamos en nuestro código dado que cuando lo hagamos por primera vez nos va a ser muy útil. Recuerden que cuando desarrollamos un sistema tenemos que pensar en todos los casos de usos donde nuestro sistema va a estar desarrollándose.

En **db.query** vamos a pasar como primer parámetro la solicitud sql y luego como segundo parámetro vamos a tener una función callback que nos servirá para capturar errores en el caso que hayan errores.

Por último **createTableIfNotExist()** lo que hace es ejecutar la función que contiene todos los procedimientos mencionados anteriormente.

Creación de los métodos HTTP

A continuación lo que haremos será crear el método que nos permitirá agregar videojuegos a nuestra base de datos.

POST

El método post recuerden que está contenido dentro de app y es el que nos permitirá realizar la acción mencionada anteriormente. Recuerden que todos los métodos http en **express js** necesitan dos parámetros, el primero es la ruta a la cual queremos direccionar esa acción y la segunda es una función que vamos a detallar a continuación:

```
app.post('/videojuegos', (req, res) => {
```

La función **callback** que vemos como segundo parámetro tiene dos parámetros req (request) , res (response). Esos parámetros nos van a servir dependiendo del tipo de petición con el cual estemos trabajando.

En este caso como vamos a hacer una petición **POST** es importante desestructurar el objeto que está dentro de req el cual obtendremos nombre, genero y plataforma.

```
const { nombre, genero, plataforma } = req.body;
```

Luego vamos a crear la query que nos permitirá justamente poder hacer la inyección sql.

```
const query = 'INSERT INTO videojuegos (nombre, genero, plataforma) VALUES (?, ?, ?)';
```

Por último trabajaremos con la query que le haremos a la db donde tomaremos los errores y la respuesta en el caso de ser correcta:

```
db.query(query, [nombre, genero, plataforma], (err, result) => {
    if (err) {
      console.error('Error al crear un nuevo videojuego:', err);
      res.status(500).json({ error: 'Error al crear un nuevo videojuego' });
    } else {
      res.json({ id: result.insertId, nombre, genero, plataforma });
    }
  });
});
```

Así debería quedarnos el método POST:

```
app.post('/videojuegos', (req, res) => {
  const { nombre, genero, plataforma } = req.body;
  const query = 'INSERT INTO videojuegos (nombre, genero, plataforma) VALUES
(?, ?, ?)';

  db.query(query, [nombre, genero, plataforma], (err, result) => {
    if (err) {
      console.error('Error al crear un nuevo videojuego:', err);
      res.status(500).json({ error: 'Error al crear un nuevo videojuego' });
    } else {
      res.json({ id: result.insertId, nombre, genero, plataforma });
    }
  });
});
```

GET

El caso para trabajar la traída de todos los videojuegos es más sencilla dado que como verán la query es estática y figura con comillas simples solo hay que traer todo de la tabla videojuegos:

```
app.get('/videojuegos', (req, res) => {
  db.query('SELECT * FROM videojuegos', (err, results) => {
    if (err) {
      console.error('Error al obtener videojuegos:', err);
      res.status(500).json({ error: 'Error al obtener videojuegos' });
    } else {
      res.json(results);
    }
  });
});
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

DELETE

Por último trabajaremos con el método **DELETE.**
Este método trae algo que no habíamos visto hasta el momento que es el parámetro **:id** en la url. Esto hace que nuestra url sea dinámica y nos permitirá eliminar el juego acorde a su id. Recuerden que **SQL** trabaja con id autoincremental, es decir que los id de cada elemento van sumando de a uno .

Como primer paso vamos a crear el endpoint con su ruta y su función que tendrá dos parámetros de la siguiente forma:

```
app.delete('/videojuegos/:id', (req, res) => {});
```

Dentro de las llaves de la función del segundo parámetro agregaremos esta linea que nos permitirá capturar el id de la url.

```
const { id } = req.params;
```

Luego escribiremos la solicitud solicitud SQL de la siguiente forma y la almacenaremos en una variable:

```
const query = 'DELETE FROM videojuegos WHERE id = ?';
```

Luego realizaremos la solicitud a la base de datos. Recuerden que el primer parámetro es la solicitud, luego le pasamos el id entre corchetes y por último realizaremos la función que verifica si el resultado fue correcto dado que podemos estar queriendo eliminar un videojuego que no exista, puede haber un error de servidor tipo 500 que no permita realizar la solicitud o puede suceder que se encuentre y se elimine correctamente el videojuego.

```
db.query(query, [id], (err, result) => {
    if (err) {
      console.error('Error al eliminar el videojuego:', err);
      res.status(500).json({ error: 'Error al eliminar el videojuego' });
```

```
    } else {
      if (result.affectedRows > 0) {
        res.json({ message: 'Videojuego eliminado correctamente' });
      } else {
        res.status(404).json({ error: 'Videojuego no encontrado' });
      }
    }
  });
```

Nuestra petición deberá quedar de la siguiente forma:

```
app.delete('/videojuegos/:id', (req, res) => {
  const { id } = req.params;
  const query = 'DELETE FROM videojuegos WHERE id = ?';

  db.query(query, [id], (err, result) => {
    if (err) {
      console.error('Error al eliminar el videojuego:', err);
      res.status(500).json({ error: 'Error al eliminar el videojuego' });
    } else {
      if (result.affectedRows > 0) {
        res.json({ message: 'Videojuego eliminado correctamente' });
      } else {
        res.status(404).json({ error: 'Videojuego no encontrado' });
      }
    }
  });
});
```

Luego de tener todas nuestros métodos http en funcionamiento vamos a realizar la conexión de nuestro servidor en el puerto que habíamos establecido al comienzo del a clase de la siguiente forma:

```
app.listen(port, () => {
  console.log(`Servidor escuchando en http://localhost:${port}`);
});
```

Por último ejecutamos en consola el comando:

```
node index.js
```

Y si la consola nos devuelve el mensaje "**Servidor escuchando en** http://localhost:3000" significa que nuestra conexión fue exitosa.

Para poder probar los **endpoints** pueden usar postman, contenido que hemos abordado en la **clase 3** pueden usar estos ejemplos para hacer inyecciones **SQL** a su base de datos:

POST :
http://localhost:3000/videojuegos

```
{
  "nombre": "The Legend of Zelda: Breath of the Wild",
  "genero": "Acción/Aventura",
  "plataforma": "Nintendo Switch"
}
```

```
{
  "nombre": "Red Dead Redemption 2",
  "genero": "Acción/Aventura",
  "plataforma": "PlayStation 4"
}
```

```
{
  "nombre": "Overwatch",
  "genero": "Shooter",
  "plataforma": "PC"
}
```

```
{
  "nombre": "Super Mario Odyssey",
  "genero": "Plataformas",
  "plataforma": "Nintendo Switch"
}
```

GET : http://localhost:3000

DELETE : (HACER SI HAY VIDEOJUEGOS CREADOS) http://localhost:3000/(ingresar numero entero que este dentro de la cantidad de elementos que hayan creado)


Buenos Aires
aprende
Buenos
Aires
Ciudad