

Agencia de  
Aprendizaje  
a lo largo  
de la vida

# Desarrollo Fullstack



# Les damos la bienvenida

Vamos a comenzar a grabar la clase

## Clase 31

### Base de Datos

- ▶ DML (SELECT, INSERT, UPDATE, DELETE)
- ▶ Manipulando datos de nuestra base
- ▶ Relaciones + JOINS

## Clase 32

### Base de Datos

- ▶ Servicios
- ▶ Modelos
- ▶ Conexión con la BBDD

## Clase 33

### Workshop

- ▶ Integración de las capas MVC vistas hasta el momento.

# NODE JS

MySQL + Models



# Repaso Base de Datos

Es un conjunto de datos almacenados y organizados que pueden estar o no relacionados entre sí.

Estos datos pertenecen a un mismo contexto y su almacenamiento surge de la necesidad de persistencia de datos, es decir, guardar para usar después.

# Servidor de Base de Datos

Para poder interactuar con nuestra base de datos es preciso montar un servidor.

Además podemos recurrir a un sistema gestor de base de datos para que esa interacción sea más sencilla.

**En esta clase veremos  
cómo trabajar con una,  
pero a través de nuestro  
programa hecho con  
Node y Express.**

# MySQL en Node

Lo primero es descargar el paquete **'mysql2'** desde **NPM**.

```
npm install mysql2
```

```
"dependencies": {  
  "express": "^4.18.2",  
  "mysql2": "^3.3.5"  
},  
"devDependencies": {  
  "nodemon": "^2.0.20"  
}
```

```
• >> clase_node 01:16 npm i mysql2  
  
added 11 packages, and audited 103 packages in 3s  
  
12 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

*Ahora podemos utilizar este paquete para conectarnos con nuestra base de datos.*



# Iniciamos nuestro servidor de base de datos.

# Conexión simple a la BBDD

Dentro de la carpeta **config**, crearemos un archivo llamado **conn.js** en él escribiremos lo necesario para poder conectarnos con nuestra base de datos.



```
const mysql = require('mysql2');

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

connection.connect();
module.exports = connection;
```

El método `.createConnection()` nos permite iniciar una instancia de una conexión individual.

***Esto implica que cada consulta con la Base de Datos debe **iniciar** y **cerrar** una conexión en cada interacción.***

# Pool de conexiones a la BBDD

Con el método `.createPool()` creamos un vínculo recurrente que admite hasta 10 **conexiones simultáneas** y **no debe cerrarse** para poder volver a realizar otra consulta.

```
const mysql = require('mysql2');

const pool = mysql.createPool({
  host: 'localhost',
  user: 'elpepe',
  password: 'admin123',
  database: 'latiendita',
  port: 3306,
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0,
});
```

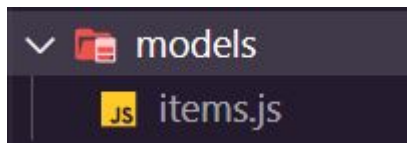
```
// Exportamos la conexión como una promesa
module.exports = {
  conn: pool.promise()
};
```

El método `.promise()` es propio de la librería **mysql2**

Ahora podemos usar **conn**  
cada vez que necesitamos  
interactuar con la BBDD.

# Models

Es una capa de nuestra app que **contendrá los archivos** encargados de **realizar consultas** a la **BBDD**



**Requerimos** nuestra **conexión** y a través de un **try/catch** y **.query()** **realizamos la consulta a la BBDD.**

***Finalmente liberamos la conexión.***

```
const { conn } = require('../config/conn');

const getItem = async () => {
  try {
    const [rows] = await conn.query('SELECT * FROM items;');
    return rows;
  } catch (error) {
    throw error;
  } finally {
    conn.releaseConnection();
  }
}

module.exports = {
  getItem
}
```

# Models - Consultas con parámetros

También podemos realizar consultas puntuales pasando parámetros específicos.

De esta manera creamos consultas “pre establecidas” mediante el `?` y este será reemplazado en orden por los parámetros recibidos.

**Nota:** podemos pasar los parámetros directamente en la consulta pero al utilizar los `?` nos aseguramos de no inyectar código tal y como viene desde el cliente evitando **SQLInjections**.

```
const.getItems = async (params) => {  
  const { price, limit, offset } = params;  
  try {  
    const [rows] = await conn.query(  
      'SELECT * FROM items WHERE price > ? LIMIT ? OFFSET ?;',  
      { price, limit, offset });  
    return rows;  
  } catch (error) {  
    throw error;  
  } finally {  
    conn.releaseConnection();  
  }  
}
```

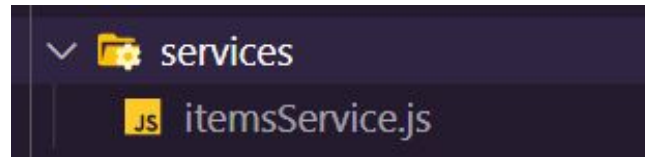
# Services

Otra capa de nuestra app, en este caso funciona como **nexo** entre los **modelos** y los **controladores**.

```
const items = require('../models/items');

const.getItems = async (params) => {
  return items.getItems(params);
};

module.exports = {
  getItems
}
```



Tomamos **valores por parámetro** desde el **controlador** y se los **enviamos al modelo** para que **realice la consulta solicitada**.

**Podemos crear un archivo por cada consulta que vayamos a necesitar o uno que junte todas y luego importarlas en donde necesitemos la información.**

***nota: las consultas a una BBDD son procesos asíncronos ya que no sabemos cuánto demoran y el resultado de la solicitud, por eso donde usemos estas funciones debemos hacerlo mediante Promesas.***



# Un momento...

Ahora que sabemos realizar consultas a nuestra base de datos podemos atender otra cuestión súper importante.

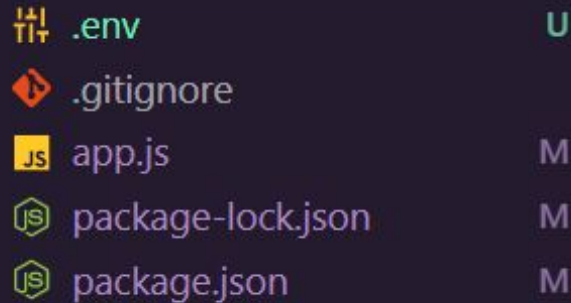
Si nos fijamos bien, estamos exponiendo la información para conectar a nuestra BBDD en el código, lo cual es una brecha de seguridad.

# Vamos a arreglarlo...

# DOTENV (repass)

En nuestro archivo **.env** creamos las variables necesarias para guardar los valores requeridos para realizar la conexión a la **BBDD**.

```
PORT=3000  
HOST=localhost  
USER=elpepe  
DBPASS=admin123  
DB=latiendita
```



A screenshot of a file explorer interface with a dark background. It lists several files with icons and status letters on the right. The files are: .env (with a Tilt icon and status 'U'), .gitignore (with a VS Code icon), app.js (with a JS icon and status 'M'), package-lock.json (with a JS icon and status 'M'), and package.json (with a JS icon and status 'M').

File	Status
.env	U
.gitignore	
app.js	M
package-lock.json	M
package.json	M

# Leemos las variables de entorno

En nuestro archivo **connection.js** requerimos la librería **dotenv** y utilizamos las variables creadas.

```
const mysql = require('mysql');  
require('dotenv').config(); // requerimos el módulo dotenv  
  
const connection = mysql.createConnection({  
  host: process.env.HOST, // usamos las variables de entorno  
  user: process.env.USER,  
  password: process.env.DBPASS,  
  database: process.env.DB  
});  
  
connection.connect();  
module.exports = connection;
```

```
HOST=localhost  
USER=root  
DBPASS=root  
DB=characters
```

# No te olvides de dar el presente

## **Recordá:**

- **Revisar la Cartelera de Novedades.**
- **Hacer tus consultas en el Foro.**

**Todo en el Aula Virtual.**

# Gracias