# **📘 Trabajo Práctico: MongoDB con Docker y Mongoose**

**Renzo Di Laudo – Comisión 3**

## **🎯 Objetivo general**

Que los estudiantes comprendan los conceptos fundamentales de MongoDB y Mongoose, y sean capaces de utilizar Docker para levantar una base de datos MongoDB y conectarse a ella desde una aplicación Node.js utilizando Mongoose.

## **🧠 Parte Teórica (Responder con sus propias palabras)**

**1. ¿Qué es MongoDB? ¿En qué se diferencia de una base de datos relacional como MySQL o PostgreSQL?**

MongoDB es un sistema de gestión de bases de datos NoSQL orientado a documentos, que utiliza un modelo de datos flexible basado en documentos BSON (Binary JSON), lo que permite almacenar datos en estructuras jerárquicas y anidadas. A diferencia de las bases de datos relacionales tradicionales, MongoDB no requiere un esquema fijo, lo que proporciona una gran flexibilidad para manejar datos en constante evolución.

**2. ¿Qué es una colección en MongoDB? ¿Y qué es un documento? Dibuja un ejemplo básico.**

Una **colección** es un conjunto de documentos dentro de una base de datos. Es equivalente a una tabla en una base de datos relacional, pero con mayor flexibilidad, ya que los documentos dentro de una misma colección no necesitan tener la misma estructura o esquemas fijos. Un **documento** es la unidad básica de datos en MongoDB, y se almacena en formato JSON (aunque internamente MongoDB usa BSON, una versión binaria de JSON). Un documento está compuesto por pares clave-valor y puede contener objetos anidados o matrices.

Como ejemplo utilizaremos una colección **Usuarios** con los siguientes **Documentos**:

{

"nombre": "Juan Pérez",

"edad": 30,

"email": "juan.perez@example.com",

"hobbies": ["fútbol", "lectura"]

}

{

"nombre": "María López",

"edad": 25,

"email": "maria.lopez@example.com",

"hobbies": ["viajar", "fotografía"]

}

**3. ¿Qué es Mongoose? ¿Para qué se utiliza en una app de Node.js?**

**Mongoose** es una biblioteca de Node.js que permite trabajar de manera sencilla y eficiente con MongoDB. Sirve como una herramienta de **modelado de datos** para establecer y gestionar la conexión entre una aplicación Node.js y una base de datos MongoDB. Mongoose es ampliamente utilizado en aplicaciones de **NodeJS** para casos como definir esquemas y modeos, realizar consultas, validar datos, optimizar la conexión con la base de datos, entre otras.

**4. ¿Qué es un esquema en Mongoose? ¿Y qué es un modelo? ¿Cuál es la relación entre ambos?**

Un **esquema** en Mongoose es una definición de la estructura de los documentos que se guardarán en una colección de MongoDB. Básicamente, actúa como un plano (o "molde") que especifica los campos que tendrán los documentos, sus tipos de datos, valores predeterminados, reglas de validación, y más. Un **modelo**, en cambio, es una clase creada a partir de un esquema. Representa una colección de la base de datos y proporciona la interfaz para interactuar con ella. Con un modelo se puede crear, leer, actualizar o eliminar documentos en la colección. La relación es simple: un **esquema define la estructura** de los documentos, mientras que el **modelo se basa en ese esquema para interactuar con la base de datos**. En otras palabras, el esquema es como un contrato que dicta cómo deben ser los datos, y el modelo es la herramienta que te permite trabajar con esos datos.

**5. ¿Qué métodos de Mongoose se pueden usar para:**

* **Insertar un nuevo documento:** Se utiliza el método **create().** Ejemplo:

const usuario = new Usuario({ nombre: "Juan", edad: 30 });

await Usuario.create({ nombre: "Juan", edad: 30 });

* **Buscar documentos:** Se utiliza el método **find()** y sus variantes:

const usuarios = await Usuario.**find**({ edad: { $gte: 18 } }); 🡪 Encuentra todos los documentos que coincidan con un criterio

const usuario = await Usuario.**findOne**({ nombre: "Juan" }); 🡪 Encuentra un único documento que coincida con un criterio

const usuario = await Usuario.**findById**("ID\_DEL\_DOCUMENTO"); 🡪 Encuentra un documento por su id

* **Actualizar un documento:** Se utiliza el método **update()** y sus variantes:

await Usuario.**updateOne**({ nombre: "Juan" }, { $set: { edad: 31 } }); 🡪 Actualiza un único documento que coincida con el criterio.

await Usuario.**updateMany**({ edad: { $lt: 18 } }, { $set: { edad: 18 } }); 🡪 Actualiza todos los documentos que coincidan con el criterio.

await Usuario.**findByIdAndUpdate**("ID\_DEL\_DOCUMENTO", { edad: 32 }); 🡪 Encuentra un documento por su ID y lo actualiza.

* **Eliminar un documento:** Se utiliza el método **delete()** y sus variantes:

await Usuario.**deleteOne**({ nombre: "Juan" }); 🡪 Elimina un único documento que coincida con el criterio.

await Usuario.**deleteMany**({ edad: { $lt: 18 } }); 🡪 Elimina todos los documentos que coincidan con el criterio.

await Usuario.**findByIdAndDelete**("ID\_DEL\_DOCUMENTO"); 🡪 Encuentra un documento por su ID y lo elimina.

## **🧪 Parte Práctica (Completar)**

**Link a github:** [**https://github.com/Renchu109/LaboratorioIV**](https://github.com/Renchu109/LaboratorioIV)

### **Consigna**

Vas a crear una pequeña aplicación en Node.js que se conecta a una base de datos MongoDB corriendo en Docker. Solo te damos una base para que completes el código y pongas a prueba lo aprendido.

### **Paso 1: Levantar MongoDB con Docker**

1. Crear un archivo docker-compose.yml y completarlo para que:
   * Se use la imagen oficial de MongoDB
   * Exponga el puerto 27017
   * Mantenga los datos persistentes usando volúmenes

version: "3.8"

services:

mongodb:

image: mongo:latest

restart: always

environment:

MONGO\_INITDB\_ROOT\_USERNAME: ${MONGO\_USER}

MONGO\_INITDB\_ROOT\_PASSWORD: ${MONGO\_PASS}

volumes:

- ./mongo:/data/db

ports:

- \*\_\_\_\_\_\_\* completar

1. Levantar el contenedor con el comando correspondiente. *docker compose up -d*
2. Verificar en Docker desktop que este corriendo la bbdd

### **Paso 2: Crear la app en Node.js**

1. Crear un nuevo proyecto e instalar las siguientes dependencias:

* npm init -y
* npm i nodemon --save -dev
* npm i express mongoose body-parser dotenv

1. **Requiere los módulos necesarios**

* Importá los módulos express y mongoose.
* Inicializá una instancia de Express.
* Agregá el middleware para interpretar JSON.

1. **Conectate a MongoDB**

* Usá mongoose.connect() para conectarse a la base de datos.
* Agregá un .then() y .catch() para mostrar mensajes de conexión exitosa o error.

1. **Define el modelo de datos**

* Usá mongoose.Schema para definir el esquema de un **usuario** con los campos:  
  + nombre (tipo String)
  + edad (tipo Number)
  + email (tipo String)
* Usá mongoose.model() para crear el modelo.

1. **Crea la ruta GET /usuarios**

* Usá app.get() para definir una ruta que devuelva todos los usuarios almacenados en la base de datos.
* Usá Usuario.find() para obtener todos los documentos.
* Devolvé los resultados como JSON.

1. **Crea la ruta POST /usuarios**

* Usá app.post() para definir una ruta que reciba un objeto JSON con datos de un nuevo usuario.
* Usá el modelo para crear y guardar ese nuevo usuario.
* Devolvé como respuesta el objeto guardado.

1. **Iniciá el servidor**

* Usá app.listen() para levantar el servidor en el puerto 3000.
* Mostrá un mensaje por consola indicando que el servidor está corriendo.