Práctico 2: Interacción con MongoDB desde Node.js usando Mongoose

En este segundo ejercicio, aprenderás a conectarte a MongoDB desde Node.js utilizando la librería Mongoose, crear una estructura para una colección de superhéroes y desarrollar métodos para interactuar con la base de datos. Este ejercicio abarca operaciones CRUD básicas: insertar, actualizar, eliminar y buscar.

Enunciado del Ejercicio

1. **Conectarse a la base de datos con Node.js** usando Mongoose.
2. **Crear un esquema y un modelo** para la colección de superhéroes.
3. **Desarrollar métodos CRUD** para:
    - Insertar (`insert`).
    - Actualizar (`update`).
    - Eliminar (`delete`).
    - Buscar (`find`).

Resolución Paso a Paso

Cada uno de los pasos incluye el **requerimiento específico**, el **proceso detallado para completarlo**, y una **justificación teórica** para entender mejor el contexto y la utilidad de cada operación.

Paso 1: Conectarse a la Base de Datos con Node.js usando Mongoose

Requerimiento: Configura una conexión a la base de datos MongoDB desde Node.js utilizando la librería Mongoose.

Proceso:

1. Instala Mongoose ejecutando el siguiente comando en la terminal:

    ```
    npm install mongoose
    ```

2. Crea un archivo `app.js` para configurar la conexión.

3. Agrega el siguiente código en `app.js`:

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

    mongoose.connect('mongodb+srv://usuario:contraseña@cursadanodejs.ls9ii.mongodb
        .net/Node-js')
    .then(() => console.log('Conexión exitosa a MongoDB'))
    .catch(error => console.error('Error al conectar a MongoDB:', error));
    ```

Asegúrate de reemplazar `usuario` y `contraseña` con tus credenciales y `PracticaSuperHeroes` con el nombre de tu base de datos.

Justificación teórica:
Mongoose es una **herramienta de modelado de datos** para MongoDB y Node.js, que permite manejar bases de datos no relacionales de manera estructurada. La conexión inicial establece una ruta entre la aplicación y el clúster MongoDB, y Mongoose facilita las interacciones de manera más segura y organizada.

Paso 2: Crear un Esquema y un Modelo para Superhéroes

Requerimiento: Define un esquema y un modelo en Mongoose para estructurar la colección de superhéroes en la base de datos.

Proceso:

1. En `app.js`, define un esquema para los superhéroes:

```
const superheroSchema = new mongoose.Schema({
    nombreSuperHeroe: { type: String, required: true },
    nombreReal: { type: String, required: true },
    edad: { type: Number, min: 0 },
    planetaOrigen: { type: String, default: 'Desconocido' },
    debilidad: String,
    poderes: [String],
    aliados: [String],
    enemigos: [String],
    createdAt: { type: Date, default: Date.now },
      creador: String
}, { collection: 'Grupo-XX' });

const SuperHero = mongoose.model('SuperHero', superheroSchema);
```

Justificación teórica:
Un **esquema** en Mongoose define la estructura y reglas de los documentos dentro de una colección, lo cual permite realizar validaciones de datos antes de almacenarlos. Un **modelo** es la implementación del esquema y permite realizar operaciones sobre los datos almacenados. Esto aporta consistencia a la base de datos y permite mantener la integridad de los datos en aplicaciones complejas.

Paso 3: Desarrollar Métodos CRUD para Insertar, Actualizar, Eliminar y Buscar Documentos

Requerimiento: Implementa métodos para insertar, actualizar, eliminar y buscar documentos en la colección `SuperHero`.

Método 1: Insertar un Documento

Proceso:

1. En `app.js`, agrega la siguiente función para insertar un nuevo superhéroe:

```
async function insertSuperHero() {
    const hero = new SuperHero({
        nombreSuperHeroe: 'Spiderman',
        nombreReal: 'Peter Parker',
        edad: 25,
        planetaOrigen: 'Tierra',
        debilidad: 'Radioactiva',
        poderes: ['Trepar paredes', 'Sentido arácnido', 'Super fuerza', 'Agilidad'],
        aliados: ['Ironman'],
        enemigos: ['Duende Verde'],
         creador: 'Martin'
    });
    await hero.save();
    console.log('Superhéroe insertado:', hero);
}

insertSuperHero();
```

Justificación teórica:
La inserción de documentos permite añadir nuevas instancias a la colección. Usar Mongoose facilita esta operación mediante el método `.save()`, que valida el documento antes de guardarlo, asegurando que cumple con las reglas del esquema.

Método 2: Actualizar un Documento

Proceso:

1. Agrega la siguiente función en `app.js` para actualizar un superhéroe existente:

```
async function updateSuperHero(nombreSuperHeroe) {
    const result = await SuperHero.updateOne(
        { nombreSuperHeroe: nombreSuperHeroe },
        { $set: { edad: 26 } }
    );
    console.log('Resultado de la actualización:', result);
}

updateSuperHero('Spiderman');
```

Justificación teórica:
La actualización de documentos es fundamental para mantener la información actualizada. En este ejemplo, `updateOne` busca un documento por el campo `nombreSuperHeroe` y actualiza el campo `edad` a 26. Este método permite modificar los datos sin necesidad de reemplazar el documento completo.

Método 3: Eliminar un Documento

Proceso:

1. Agrega esta función en `app.js` para eliminar un superhéroe de la colección:

```
async function deleteSuperHero(nombreSuperHeroe) {
    const result = await SuperHero.deleteOne({ nombreSuperHeroe: nombreSuperHeroe });
    console.log('Superhéroe eliminado:', result);
}

deleteSuperHero('Spiderman');
```

Justificación teórica:
La eliminación de documentos es una operación básica de limpieza de datos. `deleteOne` encuentra el primer documento que coincide con el criterio especificado (en este caso, `nombreSuperHeroe`) y lo elimina de la colección. Esto ayuda a gestionar el espacio y a eliminar datos obsoletos.

Método 4: Buscar Documentos

Proceso:

1. En `app.js`, agrega una función para buscar todos los superhéroes cuyo planeta de origen sea "Tierra":

```
async function findSuperHeroes() {
    const heroes = await SuperHero.find({ planetaOrigen: 'Tierra' });
    console.log('Superhéroes encontrados:', heroes);
}

findSuperHeroes();
```

Justificación teórica:
El método `find` permite recuperar documentos que cumplen con los criterios especificados, en este caso, aquellos superhéroes cuyo `planetaOrigen` es "Tierra". La búsqueda es esencial para acceder a la información almacenada y realizar operaciones posteriores.

Resumen del Ejercicio

En este segundo ejercicio, has aprendido a:

- Conectar una aplicación Node.js a MongoDB usando Mongoose.
- Crear un esquema y un modelo para estructurar y validar los datos.
- Implementar métodos básicos para insertar, actualizar, eliminar y buscar documentos.

Estos pasos te ayudan a comprender cómo manipular datos de una base de datos MongoDB desde una aplicación Node.js, utilizando Mongoose para gestionar la estructura y validación de datos de manera eficiente y flexible.