Bases de datos

Mongoose

Cordero Hernández, Marco R.

En sesiones pasadas...

- MongoDB
  - Conceptos
  - Instalación
  - Uso básico
  - Cloud Atlas

CONTENIDOS

01

INTRODUCCIÓN A
Mongoose

Mongoose

Mongoose es una utilería de apoyo para *ODM* enfocada a al trabajo en conjunto de **Node.js** y **MongoDB**.

La ventaja sobre **MongoDB** nativo es que **Mongoose** proveé tipos de datos extendidos, validación de campos, construcción de queries, etc., todo en un mismo paquete.

Uso de Mongoose – Preliminar

Mongoose también requiere de su instalación a través de ***npm***.

- `npm i mongoose`

```
// Importar módulo
const mongoose = require('mongoose');

// Conexión a la base
let db_url = 'mongodb://localhost:27017/AlumnosDB'; // O tu URL
mongoose.connect(db_url, {}).then(() => {
    console.log('Conexión exitósa a MongoDB!');
}).catch(err => {
    console.log(`Error de conexión: ${err}`);
});
```

Uso de Mongoose – Esquemas

Los esquemas usualmente se definen dentro de un directorio ***models/***

```
// Importar modelo (esquema) de alumno
const alumnoModel = require('./src/models/alumno');

// Guardar documento en la colección de alumno
let newAlumno = {
   nombre: 'Marco C',
   edad: 23,
   carrera: 'IC'
};
alumnoModel.create(newAlumno);
```

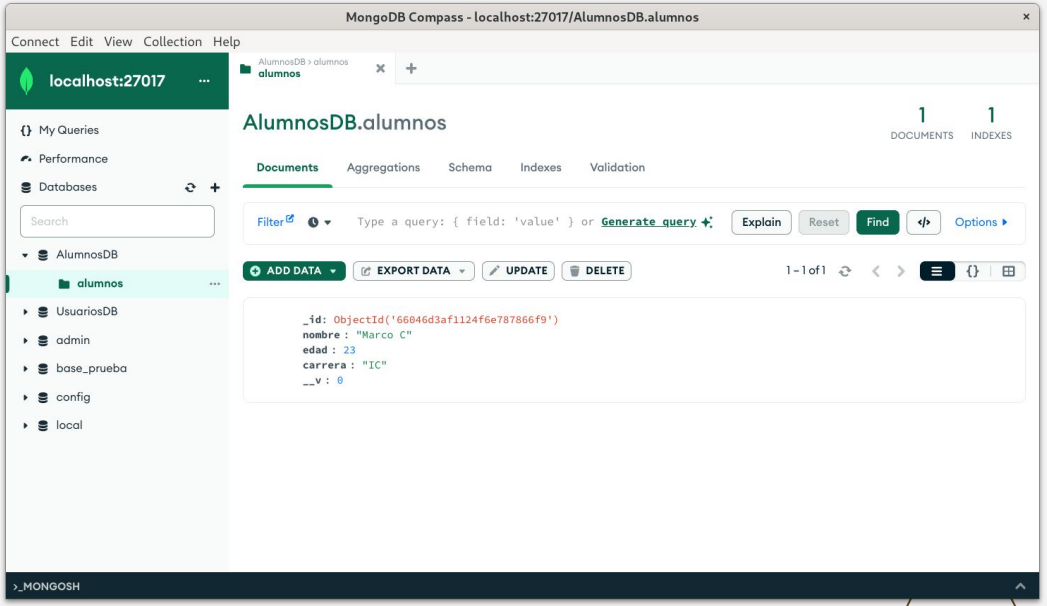
```
/* Modelo de Alumno */
// Importar mongoose
const { Schema, model } = require('mongoose');

// Esquema
const alumnoSchema = new Schema({
   nombre: String,
   edad: { type: Number, min: 18, max: 99, required: true },
   carrera: { type: String, enum: ['IE', 'PS', 'IC'], required: true }
});

// Exportar esquema
module.exports = model('alumno', alumnoSchema);
```

Uso de Mongoose

```
AlumnosDB> db.alumnos.find({})
[
  {
    _id: ObjectId('66046d3af1124f6e787866f9'),
    nombre: 'Marco C',
    edad: 23,
    carrera: 'IC',
    __v: 0
  }
]
```



Ejercicio

- Establece una conexión a tu base de datos (local o remota)
- Crea un nuevo esquema de usuario
  - Añade todos los atributos usados previamente
  - Añade el atributo *rol* con los posibles valores ['ADMIN', 'USER', 'GUEST']
- Crea un nuevo modelo **User**
- Crea un nuevo usuario y verifícalo en la base de datos
  - Local: con **mongosh** o **Compass**
  - Remoto: desde **Mongo Atlas**

02

BÚSQUEDA DE DOCUMENTOS

Buscar documentos

Búsqueda ≈ Obtener documentos bajo ciertos criterios.

- **`find(criterio)`** - Buscar documentos que cumplan con el criterio
- **`findById(id)`** - Buscar por ID
- **`findByIdAnd[Delete,Update](id)`** - Buscar por ID y borrar o actualizar
- **`findOne(criterio)`** - Regresa el primer documento que cumpla con el criterio
- **`findOneAnd[Delete,Update](criterio)`** - Busca un documento y lo borra o actualiza

Para obtener los resultados en formato **JSON**, usar *`.lean()`* después de cada operación y manejarlo como promesa.

Buscar documentos

```
// Buscar todos los documentos de una colección
alumnoModel.find({}).lean().then(response => {
    console.log('Todos los documentos');
    response.forEach(item => console.log(item));
});

// Buscar con criterio (regex para nombres que empiezan con M)
alumnoModel.find({
    nombre: /^M/
}).lean().then(response => {
    console.log('Documentos que empiezan con M');
    response.forEach(item => console.log(item));
});

// Edad en el rango de 18 a 21
alumnoModel.find({ edad: { $gte: 18, $lte: 21 } })
.lean().then(response => {
    console.log('Rango de edad de 18 a 21');
   response.forEach(item => console.log(item));
});
```

```
Todos los documentos
{
  _id: new ObjectId('66046d3af1124f6e787866f9'),
  nombre: 'Marco C',
  edad: 23,
  carrera: 'IC',
  __v: 0
}
{
  _id: new ObjectId('66048498c5e8e21e0e1852b7'),
  nombre: 'Ricardo H',
  edad: 20,
  carrera: 'PS'
}
```

```
Documentos que empiezan con M
{
  _id: new ObjectId('66046d3af1124f6e787866f9'),
  nombre: 'Marco C',
  edad: 23,
  carrera: 'IC',
  __v: 0
}
```

```
Rango de edad de 18 a 21
{
  _id: new ObjectId('66048498c5e8e21e0e1852b7'),
  nombre: 'Ricardo H',
  edad: 20,
  carrera: 'PS'
}
```

Ejercicio

Inserta dos nuevos usuarios con datos distintos y después muestra en consola:

- A todos los usuarios
- Solo los usuarios hombres
- Un usuario con ID específico

03

UPDATE / DELETE

Actualizar documentos

Para lograrlo, es posible utilizar:

- **`findOneAndUpdate(criterio, nuevos_datos, opciones)`** - Buscar un documento y actualizarlo
- **`findByIdAndUpdate(id, nuevos_datos, opciones)`** - Buscar un documento por ID y actualizarlo

De igual manera, es posible manejar estos métodos como promesas con el respectivo `.then()` y `.catch()`

Ejercicio

Cambiar el nombre de un usuario.

TIP: las ***opciones*** dentro de los métodos vistos pueden contener las siguientes configuraciones

- ***new:*** `true` - Regresa el documento modificado (útil para mostrarlo o retornarlo sin tener que buscarlo de nuevo)
- ***omitUndefined***: `true` - Puede usarse para modificar solo una parte del documento sin generar error (similar al método de HTTP ***PATCH***)

Borrar documentos

Para lograrlo, es posible utilizar:

- **`findOneAndDelete(criterio)`** - Buscar un documento y eliminarlo (regresa el documento eliminado)
- **`findByIdAndDelete(id)`** - Buscar un documento por ID y eliminarlo (regresa el documento eliminado)
- **`deleteMany(criterio)`** - Eliminar múltiples documentos
- **`deleteOne(criterio)`** - Elimina un solo documento
- ***`remove()`*** - Método deprecado (usar métodos anteriores)

Consideraciones adicionales

Para validar un ID, se puede usar

- `mongoose.Types.ObjectId.`***`isValid(id)`***

Esto se hace debido a que *no todos los objetos son IDs válidos*.

Como ya se vió, la mayoría de las operaciones pueden ser tratadas como *promesas*, por ende, es posible utilizar ***async/await*** dentro de una función para manejar las llamadas a la base de mejor manera.

Ejercicio *opcional*

Implementa el uso de **MongoDB** y **Mongoose** en el ejercicio integrador realizado hace un par de sesiones.

- Implementa la función de búsqueda
  - La estructura ya está implementada
  - El flujo de búsqueda debería ser frontend (petición **GET** con parámetros de búsqueda) -> backend (query hacía la base de datos) -> **MongoDB** (documentos resultantes) -> backend (documentos resultantes) -> frontend (vista de los resultados)
- Reemplaza la lectura/escritura de archivos locales por **MongoDB** (local o remoto)

04

RELACIÓN ENTRE COLECCIONES

Relación entre colecciones

Todas las bases *no relacionales* son ***NoSQL***, sin embargo, no todas las bases ***NoSQL*** son *no relacionales*. **MongoDB** es una base de documentos *no relacional*, por lo tanto, para establecer un vínculo entre dos o más colecciones es necesario implementar un mecanismo personalizado para lograrlo, como guardar los ids de documentos dentro de otros para "ligarlos".

Ejemplo: Cada documento de la colección "carrito(s)" tiene su propio ID, pero, también *tendría* el ID del usuario que lo creó, haciendo alusión a un documento de la colección "usuario(s)"

Relación entre colecciones

Ejemplo: Dentro de un documento perteneciente a “grupo(s)” con esquema { `asignatura: “Web”, profesor: “MRCH”, periodo: “2024A”, alumnos: [1, 2, 3, 4]` } la propiedad *`alumnos`* almacenaría IDs de “alumno(s)”.

Si se quisiera obtener la información de todos los alumnos inscritos en alguna materia, bastaría con obtener el arreglo de IDs dentro del documento y luego iterar sobre cada uno de sus elementos, realizando búsquedas individuales por cada ID.

★ Las funciones de búsqueda admiten un segundo parámetro para *proyectar* campos, es decir, cuáles sí/no se quieren en el resultado final

```
modeloGrupos.findOne({ asignatura: ‘Web’ }, { alumnos: 1 })
```

Relación entre colecciones

Como en cualquier tecnología bien establecida, lo anterior puede lograrse de forma más sencilla o al menos más compacta y directa. Para esto es posible usar **aggregate**, **$lookup** y **$project**.

https://stackoverflow.com/questions/51808279/getting-all-docutments-present-in-an-array-in-another-collection

https://www.mongodb.com/docs/manual/aggregation/

https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/

https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/