OBJETIVO

En esta guía aprenderemos a a conectarnos a la Base de Datos MongoDB usando Node, para que a futuro puedas implementarlo en tus aplicaciones.

Conexión

Para poder conectarnos a Mongodb tenemos dos opciones: El driver oficial mongodb y la librería Mongoose.

Mongoose es una libreria Object-Document-Mapping (ODM) que proporciona una solución sencilla para trabajar con Mongodb y basada en esquemas para modelar los datos de la aplicación. Incluye conversión de tipos integrada, validación, creación de consultas. En esta guía trabajaremos con la librería `Mongoose`, para ello se debe agregar la dependencia a nuestro proyecto:

```
$ npm install mongoose
```

Para crear una conección debemos hacerlo de la siguiente forma:

```
const Mongoose = require("mongoose");
Mongoose.connect(
"mongodb+srv://tecnologias:Uajms.2020@cluster0.xgwrohx.mongodb.net/?retryWrites=true&w
=majority&appName=Cluster0",
    {
       // useNewUrlParser: true,
       // useFindAndModify: false,
       // useUnifiedTopology: true
    }
).then(()=>{
    console.log("OK se conecto a mongodb");
}).catch((e)=>{
    console.log("Se produjo un error de conexión");
})
```

Creación de Esquemas

Un esquema de mongoose es una estructura JSON, sirve para representar la estructura del documento que vamos a utilizar. En un esquema también se puede almacenar información adicional como la validación y los valores por defecto.

Los esquemas se declaran de la siguiente forma:

```
const cursosSchema = mongoose.Schema({
    nombre:String,
    sigla:String
});
```

Construir Modelos

El modelo es otra pieza fundamental que nos permite interactuar con MongoDB. Un modelo se lo crea a partir de `mongoose.model`:

```
const CursosModel = mongoose.model('cursos',cursosSchema);
```

Una vez definido el modelo ya podemos empezar a insertar, buscar, actualizar y eliminar los documentos de una colección.

Crear un Documento

Existen varias formas de insertar documentos en una colección. La primera es instanciar el modelo y utilizar el método `save`:

```
const cursoNuevo = new CursosModel({
        nombre:"INFORMATICA 1",
        sigla:"INF-101"
    });
 cursoNuevo.save();
```

Otra forma de crear documentos es:

```
 CursosModel.create({nombre:"QUIMICA 1",sigla:"QMC2"});
```

Para crear muchos documentos

```
CursosModel.insertMany([
  { nombre: "CALCULO 1", sigla: "MAT-102" },
  { nombre: "CALCULO 2", sigla: "MAT-202" }
  ]);
```

Buscar Documentos

Para buscar los documentos de una colección usamos función `find`

```
const listarCursos =  async ()=>{
    const cursos = await CursosModel.find({});
    console.log(cursos);
}

listarCursos();
```

Para hacer búsquedas con condiciones lo podemos hacer de la siguiente forma:

```
EstudiantesModel.find({ nota: { $gte: 80, $lte: 100 } });
```

Actualizar documentos

Para actualizar documentos se ejecuta el método `updateOne` y `updateMany`:

```
CursosModel.updateOne({_id:id},{
        $set:{nombre:'INFORMATICA 1 MODIFICADO'}
    });
```

Eliminar un documento

Para eliminar documentos se usa el método `deleteOne` y `deleteMany`

```
CursosModel.deleteOne({_id:"60c78b205b97db1f249d4c37"});
```

CRUD completo de personas

El siguiente ejemplo muestra un CRUD de personas usando MongoDB y NodeJS.

Creamos la carpeta `crud-personas`

```
mkdir crud-personas
cd crud-personas
```

inicializamos el proyecto:

```
npm init -y
```

```
npm install express mongoose body-parser
```

Creamos el archivo `app.js` con el siguiente contenido:

app.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

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

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

// Conectar a MongoDB
mongoose.connect('mongodb://localhost:27017/sicrama', {
    useNewUrlParser: true,
    useUnifiedTopology: true
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'Error de conexion:'));
db.once('open', () => {
    console.log('Contado a mongoDB correctamente');
});

// Definir el esquema y el modelo de Persona
const personaSchema = new mongoose.Schema({
    nombres: String,
    apellidos: String,
    ci: Number,
    nrocelular: Number,
    direccion: String,
    nrorecibo: Number,
    edad: Number
});

const Persona = mongoose.model('personas', personaSchema);

// Crear una nueva persona
app.post('/personas', async (req, res) => {
    const persona = new Persona(req.body);
    try {
        await persona.save();
        res.status(201).send(persona);
    } catch (error) {
        res.status(400).send(error);
    }
});

// Obtener todas las personas
app.get('/personas', async (req, res) => {
    try {
        const personas = await Persona.find();
        res.status(200).send(personas);
    } catch (error) {
        res.status(500).send(error);
    }
});

// Obtener una persona por ID
app.get('/personas/:id', async (req, res) => {
    try {
```

```
        const persona = await Persona.findById(req.params.id);
        if (!persona) {
            return res.status(404).send();
        }
        res.status(200).send(persona);
    } catch (error) {
        res.status(500).send(error);
    }
});

// Actualizar una persona por ID
app.put('/personas/:id', async (req, res) => {
    try {
        const persona = await Persona.findByIdAndUpdate(req.params.id, req.body,
{ new: true, runValidators: true });
        if (!persona) {
            return res.status(404).send();
        }
        res.status(200).send(persona);
    } catch (error) {
        res.status(400).send(error);
    }
});

// Eliminar una persona por ID
app.delete('/personas/:id', async (req, res) => {
    try {
        const persona = await Persona.findByIdAndDelete(req.params.id);
        if (!persona) {
            return res.status(404).send();
        }
        res.status(200).send(persona);
    } catch (error) {
        res.status(500).send(error);
    }
});

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

Iniciamos el servidor:

```
node app.js
```

Finalmente hacemos las pruebas correspondientes con la herramienta postman para probar los endpoints:

Utiliza herramientas como Postman o curl para probar los endpoints del CRUD:

- POST /personas: Crear una nueva persona.
- GET /personas: Obtener todas las personas.

- GET /personas/:id: Obtener una persona por ID.
- PUT /personas/:id: Actualizar una persona por ID.
- DELETE /personas/:id: Eliminar una persona por ID.

Ejemplo de dato de prueba:

```
{
  "nombres": "Omar",
  "apellidos": "Franco Sanchez",
  "ci": 5059008,
  "nrocelular": 67202080,
  "direccion": "Palos Blancos",
  "nrorecibo": 25652,
  "edad": 38
}
```