Clase 22: Desarrollo de Aplicaciones con MongoDB y Node.js

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:54
Libro:	Clase 22: Desarrollo de Aplicaciones con MongoDB y Node.js		

Descripción

Objetivos

- Conectar una base de datos MongoDB a una aplicación Node.js y crear un API RESTful con Node.js y MongoDB.
- Utilizar Mongoose como ODM (Object Document Mapper) para modelar esquemas, validar datos y realizar consultas a MongoDB.
- Integrar MongoDB con Express.js y Node.js para autenticar y autorizar usuarios.

Tabla de contenidos

1. Introducción

2. .Conexión a MongoDB desde una aplicación Node.js

3. Creación de un API RESTful con Node.js y MongoDB

4. Mongoose

5. Integración de MongoDB

1. Introducción

En esta sección exploraremos la integración de MongoDB y Node.js para el desarrollo de aplicaciones web sólidas. Comenzaremos con la conexión a una base de datos MongoDB desde Node.js y crearemos un API RESTful para operaciones CRUD. Utilizaremos Mongoose como ODM para estructurar datos y validarlos, realizaremos consultas eficientes a MongoDB y aprenderemos a integrar MongoDB con Express.js. Además, abordaremos la autenticación y autorización de usuarios, esenciales para la seguridad en aplicaciones web.

2. .Conexión a MongoDB desde una aplicación Node.js

Conexión a MongoDB desde una aplicación Node.js

La combinación de Node.js y MongoDB en el desarrollo de aplicaciones ofrece ventajas como el rendimiento, escalabilidad, flexibilidad, bajo acoplamiento y un flujo de trabajo eficiente debido al uso de JavaScript en ambos lados. Esto facilita la creación de aplicaciones modernas de manera rápida y efectiva, respaldada por comunidades activas y soporte sólido.

Uso del controlador oficial de MongoDB en Node.js.

El controlador oficial de MongoDB para Node.js es una biblioteca de software que facilita la comunicación entre una aplicación Node.js y una base de datos MongoDB. A continuación, profundicemos en algunos aspectos clave de su uso:

Interacción con MongoDB

El controlador permite a los desarrolladores de Node.js conectarse a una base de datos MongoDB, realizar consultas y modificar datos de manera eficiente. Proporciona métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en documentos almacenados en colecciones de MongoDB.

API Coherente

El controlador oficial de MongoDB ofrece una API consistente y fácil de usar. Los desarrolladores pueden acceder a funciones y métodos para realizar tareas comunes de base de datos sin tener que escribir código complejo desde cero.

Rendimiento y Escalabilidad

El controlador está optimizado para trabajar eficazmente con MongoDB. Esto es crucial para aplicaciones que necesitan manejar grandes cantidades de datos y escalabilidad horizontal. El controlador permite a Node.js administrar varias conexiones a MongoDB de manera eficiente y ejecutar operaciones en paralelo.

Mapeo de Objetos-Documento (ODM)

El controlador oficial no solo permite interactuar con MongoDB sino que también incluye un módulo llamado "Mongoose", que actúa como un ODM (Object Document Mapper). Mongoose facilita el modelado de esquemas, la validación de datos y la definición de relaciones entre documentos, lo que agrega una capa adicional de abstracción y facilita la gestión de datos en aplicaciones Node.js.

Documentación Completa

MongoDB ofrece una documentación detallada y ejemplos de uso del controlador para Node.js. Los desarrolladores pueden consultar fácilmente esta documentación para aprender a utilizar el controlador y resolver problemas específicos.

Configuración de la cadena de conexión y las opciones para la autenticación.

La configuración de la cadena de conexión en MongoDB implica definir una URL de conexión que incluye detalles como nombre de usuario, contraseña, host y puerto del servidor, y la base de datos a la que te conectarás. Además, se pueden establecer opciones de autenticación como `authSource`, `authMechanism`, y opciones SSL/TLS para garantizar una conexión segura. La conexión se realiza mediante el controlador oficial de MongoDB para Node.js, como `MongoClient.connect()`. La correcta configuración de estos aspectos es esencial para la seguridad y la integridad de tus datos en MongoDB.

Conexión a una base de datos MongoDB desde una aplicación Node.js.

La conexión a una base de datos MongoDB desde Node.js es un paso fundamental en el desarrollo de aplicaciones que utilizan esta combinación de tecnologías. Para lograrlo, puedes seguir estos pasos:

1. **Instala el controlador de MongoDB para Node.js:** Asegúrate de que tengas el controlador oficial de MongoDB para Node.js instalado en tu proyecto. Puedes hacerlo utilizando npm (Node Package Manager) ejecutando el siguiente comando en la terminal:

```
npm install mongodb
```

2. **Importa el módulo MongoClient:** En tu archivo de código Node.js, importa el módulo MongoClient que proporciona el controlador de MongoDB. Debes hacerlo al comienzo de tu archivo:

```
const MongoClient = require('mongodb').MongoClient;
```

3. **Configura la cadena de conexión:** Define una cadena de conexión que incluya la dirección del servidor MongoDB, el puerto, y la base de datos a la que te conectarás. Puedes incluir opciones de autenticación y configuración de SSL/TLS si es necesario.

```
const url = 'mongodb://localhost:27017/mi_basededatos';
```

4. **Conéctate a la base de datos:** Utiliza el método `connect` del objeto `MongoClient` para establecer una conexión con la base de datos:

```
MongoClient.connect(url, (err, db) => {
  if (err) throw err;
```

```
  console.log('Conexión a MongoDB establecida con éxito');
  // Realiza operaciones en la base de datos
  // ...
  // Cierra la conexión cuando hayas terminado
  db.close();
});
```

5. **Realiza operaciones en la base de datos:** Una vez que la conexión se haya establecido con éxito, puedes realizar operaciones en la base de datos, como inserción, consulta, actualización y eliminación de documentos.

6. **Cierra la conexión:** Es importante cerrar la conexión a la base de datos cuando hayas terminado de utilizarla para liberar recursos.

Recuerda que es esencial gestionar los errores adecuadamente, y también puedes aprovechar las ventajas de usar promesas o async/await para un código más limpio y legible.

3. Creación de un API RESTful con Node.js y MongoDB

Creación de un API RESTful con Node.js y MongoDB

Un API RESTful permite la comunicación entre clientes (como aplicaciones web, móviles o de escritorio) y una base de datos MongoDB a través de solicitudes HTTP. Aquí se describen los pasos básicos para crear un API RESTful con Node.js y MongoDB:

Configuración del Proyecto

· Crea un nuevo directorio para tu proyecto y, dentro de él, inicia un proyecto Node.js utilizando `npm init`.

· Instala las dependencias necesarias, como Express (para crear el servidor web) y Mongoose (para interactuar con MongoDB).

```
npm install express mongoose
```

Definición de Rutas y Controladores

· Crea rutas que representen las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) que deseas para tu API. Por ejemplo, puedes tener rutas para `/api/usuarios` o `/api/productos`.

· Crea controladores para manejar las solicitudes entrantes y comunicarse con la base de datos MongoDB. Los controladores deben definir funciones para crear, leer, actualizar y eliminar documentos en MongoDB.

```
// Ejemplo de un controlador para usuarios
const Usuario = require('../modelos/Usuario');

const crearUsuario = async (req, res) => {
  try {
    const nuevoUsuario = new Usuario(req.body);
    const usuarioGuardado = await nuevoUsuario.save();
    res.status(201).json(usuarioGuardado);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
}
```

```
// Define otras funciones para leer, actualizar y eliminar usuarios
```

Modelado de Datos con Mongoose

· Define los modelos de datos utilizando Mongoose. Los modelos representan las estructuras de datos que se almacenan en MongoDB. Por ejemplo, para un modelo de usuario:

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

const usuarioSchema = new mongoose.Schema({
  nombre: String,
  correo: String,
  edad: Number,
});

module.exports = mongoose.model('Usuario', usuarioSchema);
```

Creación del Servidor Express

· Configura un servidor Express y define las rutas que responden a las solicitudes API.

```
const express = require('express');
const app = express();

// Middleware para el manejo de solicitudes JSON
app.use(express.json());

// Rutas de API
const usuariosRutas = require('./rutas/usuarios');
app.use('/api/usuarios', usuariosRutas);

// Inicia el servidor
```

```
const puerto = process.env.PUERTO || 3000;
app.listen(puerto, () => {
  console.log(`Servidor en ejecución en el puerto ${puerto}`);
});
```

Prueba de las Rutas del API

· Utiliza herramientas como Postman o Insomnia para probar las rutas del API. Deberías poder realizar solicitudes HTTP a las rutas que has definido y ver cómo interactúa tu API con la base de datos MongoDB.

Recuerda seguir prácticas de seguridad, autenticación y autorización para proteger tu API, y manejar errores adecuadamente. La creación de un API RESTful con Node.js y MongoDB te permite construir aplicaciones web escalables y flexibles.

4. Mongoose

Uso de Mongoose como ODM (Object Document Mapper)

Mongoose es una biblioteca de Node.js que actúa como un Object Document Mapper (ODM) para facilitar la interacción con bases de datos MongoDB. El uso de Mongoose como ODM (Object Data Modeling) en una aplicación Node.js implica definir la estructura de tus datos y cómo interactuar con MongoDB de manera más organizada y estructurada.

Definición de Esquemas

En lugar de trabajar directamente con documentos en MongoDB, Mongoose te permite definir esquemas que representan la estructura de tus datos. Esto significa especificar qué campos se almacenan en un documento y qué tipos de datos deben contener. Los esquemas también permiten aplicar validaciones y restricciones a los datos.

Creación de Modelos

A partir de los esquemas, creas modelos que representan colecciones en tu base de datos MongoDB. Los modelos son objetos que te permiten interactuar con los documentos de la base de datos como si fueran objetos JavaScript. Esto simplifica la creación, lectura, actualización y eliminación de datos.

Validación de Datos

Mongoose ofrece opciones de validación que te permiten asegurarte de que los datos cumplen con ciertas reglas antes de ser guardados en la base de datos. Puedes validar campos, requerir valores, definir valores por defecto y personalizar las validaciones según tus necesidades.

Consultas y Actualizaciones

Puedes utilizar Mongoose para realizar consultas y actualizaciones en la base de datos. Mongoose proporciona una API que facilita la creación de consultas para recuperar datos específicos, filtrar documentos y realizar actualizaciones en lotes.

Conexión a la Base de Datos

Mongoose facilita la conexión a tu base de datos MongoDB, lo que simplifica el proceso de configuración y gestión de la conexión en tu aplicación Node.js.

En resumen, Mongoose como ODM te brinda una capa de abstracción para trabajar con MongoDB de una manera más orientada a objetos y estructurada. Esto hace que el desarrollo de aplicaciones Node.js que utilizan MongoDB sea más eficiente y menos propenso a errores, al proporcionar herramientas para definir, validar y gestionar datos en tu base de datos.

Modelado de esquemas con Mongoose y validación de datos

Modelado de Esquemas con Mongoose

Mongoose permite definir la estructura de los datos que se almacenan en MongoDB utilizando esquemas.

Un esquema es una representación de cómo se verán los documentos en una colección de MongoDB, lo que significa especificar qué campos tendrán y qué tipos de datos albergarán.

Los esquemas se crean utilizando la clase `mongoose.Schema`. Dentro de un esquema, puedes definir campos, tipos de datos, restricciones y opciones específicas para cada campo. Esto proporciona una estructura coherente y cohesiva para tus datos.

Ejemplo de definición de esquema con Mongoose:

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


const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  age: {
    type: Number,
    min: 18,
  },
```

```
});
```

Validación de Datos

Mongoose permite aplicar validaciones a los datos antes de que se almacenen en la base de datos MongoDB. Puedes definir reglas para asegurarte de que los datos cumplan con ciertos criterios.

Por ejemplo, en el esquema anterior, hemos aplicado las siguientes validaciones:

- El campo "name" debe ser una cadena de texto y es requerido.
- El campo "email" debe ser una cadena de texto única y requerida.
- El campo "age" debe ser un número y debe tener al menos un valor de 18.

Cuando intentas guardar un documento que no cumple con estas validaciones, Mongoose generará errores que puedes manejar en tu aplicación.

Consultas a MongoDB utilizando Mongoose

Consultar MongoDB utilizando Mongoose en una aplicación Node.js es un proceso esencial para recuperar, actualizar, eliminar o agregar datos a tu base de datos. Aquí tienes una explicación de cómo realizar consultas utilizando Mongoose:

Conexión a la Base de Datos:

Antes de realizar consultas, debes establecer una conexión a tu base de datos MongoDB utilizando Mongoose. Esto se hace una sola vez en tu aplicación, generalmente en el archivo de entrada principal. Puedes usar el método `mongoose.connect()` para conectar con la base de datos y especificar la URL de conexión.

Definición de Modelos:

Los modelos en Mongoose son la representación de tus colecciones de MongoDB. Debes definir un modelo para cada colección que planeas consultar. Esto se hace creando un esquema utilizando `mongoose.Schema` y luego creando un modelo a partir de ese esquema utilizando `mongoose.model()`.

Ejemplo de definición de un modelo:

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});
const User = mongoose.model('User', userSchema);
```

Realización de Consultas:

Una vez que tengas un modelo, puedes usarlo para realizar consultas en la base de datos. Mongoose proporciona métodos de consulta que facilitan la búsqueda de documentos en una colección.

Ejemplo de consulta para encontrar todos los usuarios:

```
User.find({}, (err, users) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(users);
});
```

Filtros y Opciones

Puedes proporcionar filtros y opciones en tus consultas para refinar los resultados. Por ejemplo, para encontrar usuarios con un nombre específico:

```
User.find({ name: 'John' }, (err, users) => {
  // Resultados que coinciden con el filtro
});
```

Actualización y Eliminación:

Mongoose también permite realizar operaciones de actualización y eliminación en documentos. Puedes utilizar métodos como `updateOne()`, `updateMany()`, `deleteOne()`, y `deleteMany()` para modificar o eliminar datos.

Ejemplo de actualización:

```
User.updateOne({ name: 'John' }, { email: 'john@example.com' }, (err) => {
  if (err) {
```

```
    console.error(err);
    return;
  }
  console.log('Usuario actualizado con éxito');
});
```

Manejo de Resultados

Las consultas generalmente devuelven un objeto que contiene los resultados o un objeto `Query`. Puedes utilizar `.exec()` o promesas para ejecutar y manejar los resultados de manera más efectiva.

Ejemplo de manejo de resultados con promesas:

```
User.find({ name: 'John' })
  .then(users => {
    console.log(users);
  })
  .catch(err => {
    console.error(err);
  });
```

Realizar consultas a MongoDB utilizando Mongoose te permite interactuar de manera efectiva con tus datos almacenados en la base de datos, lo que es esencial para el desarrollo de aplicaciones Node.js con MongoDB.

5. Integración de MongoDB

Integración de MongoDB con Express.js y Node.js

La integración de MongoDB con Express.js y Node.js implica:

1. Configurar Express.js para manejar solicitudes HTTP.
2. Instalar el paquete Mongoose como ODM para MongoDB.
3. Establecer una conexión con la base de datos MongoDB.
4. Definir modelos Mongoose para representar las colecciones.
5. Crear rutas y controladores para manejar solicitudes HTTP.
6. Utilizar modelos y métodos Mongoose en la lógica de la aplicación para interactuar con la base de datos MongoDB. Esto incluye consultas, agregado, actualización y eliminación de datos.

Esta integración permite construir aplicaciones web dinámicas y escalables que aprovechan la potencia de MongoDB junto con la versatilidad de Node.js y Express.js.

Autenticación y autorización de usuarios con MongoDB y Node.js

La autenticación y autorización de usuarios con MongoDB y Node.js es esencial en muchas aplicaciones web para garantizar la seguridad y el acceso controlado a recursos. A través de este proceso, los usuarios se autentican (verifican su identidad) y se autoriza (se les otorgan permisos) para acceder a ciertas partes de la aplicación o realizar acciones específicas.

MongoDB y Node.js proporcionan herramientas y capacidades para implementar sistemas de autenticación y autorización eficaces:

· **Autenticación de Usuarios:** Node.js se utiliza para crear sistemas de autenticación donde los usuarios proporcionan credenciales (como nombre de usuario y contraseña). Se pueden aplicar técnicas de cifrado para almacenar contraseñas de manera segura en la base de datos MongoDB

· **Gestión de Sesiones:** Node.js permite la gestión de sesiones de usuario, lo que es fundamental para mantener la autenticación a lo largo de las solicitudes HTTP.

· **Autorización y Control de Acceso:** A través de Node.js, puedes definir roles y permisos de usuario y garantizar que los usuarios solo tengan acceso a las rutas o recursos para los que están autorizados.

· **Middleware de Autenticación:** Node.js y Express.js permiten el uso de middleware para autenticar a los usuarios antes de que accedan a ciertas rutas o recursos. Esto puede incluir la verificación de

tokens de sesión, verificación de roles, etc.

· **Bibliotecas y Frameworks:** Existen bibliotecas y frameworks específicos de autenticación en Node.js, como Passport, que facilitan la implementación de estrategias de autenticación, como OAuth, JWT (JSON Web Tokens), etc.

· **Almacenamiento de Datos de Usuario:** MongoDB se utiliza para almacenar información de usuario, incluidos datos de autenticación, roles y otros datos relevantes.

La combinación de MongoDB y Node.js proporciona una plataforma sólida para la autenticación y autorización en aplicaciones web, lo que es crucial para proteger los datos y garantizar la privacidad de los usuarios.

Bibliografía utilizada y sugerida

· Connection Guide — Node.js. (s. f.). https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/connect/#std-label-node-connect-to-mongodb

· Express Tutorial Part 3: Using a database (with Mongoose) - Learn Web Development | MDN. (2023, 29 octubre). https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

· GitHubber. (s. f.). Creación de una API web con Node.js y Express mediante JavaScript - training. Microsoft Learn. https://learn.microsoft.com/es-es/training/modules/build-web-api-nodejs-express/

· Integración de la base de datos de Express. (s. f.). https://expressjs.com/es/guide/database-integration.html

· Mongoose V8.0.0: Getting started. (s. f.). https://mongoosejs.com/docs/index.html