El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

CICLO IV:

Desarrollo de Aplicaciones Web

MisionTIC2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Sesión 13:
Desarrollo de Aplicaciones Web

Bases de datos no relacionales

MisiónTIC2022

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Comprender el concepto de base de datos no relacionales.
2. Diseñar una base de datos no relacional.
3. Implementar una base de datos no relacional.
4. Gestionar los datos de una base de datos no relacional a través de una aplicación de Node.js.

Bases de Datos NoSQL - Conceptos

- Por lo general las bases de datos NoSQL se utilizan en contraste con respecto a las bases de datos SQL.
- Por lo general dentro las principales razones para adoptar una base de datos NoSQL encontramos:
  - El ritmo de desarrollo con bases de datos NoSQL puede ser mejor respecto a bases de datos SQL.
  - Hay una mayor versatilidad en las bases de datos NoSQL, puesto que permite manejar datos con diversas estructuras con más facilidad.
  - La cantidad de datos en muchas aplicaciones no puede ser consumida de una forma asequible usando bases de datos SQL.
  - Son más fáciles y rápidas de escalar que una base de datos SQL.
  - Nuevos paradigmas pueden ser soportados de una forma más sencilla.

Bases de Datos NoSQL - Tipos

- Dentro de los tipos de base de datos NoSQL encontramos las siguientes:
  - Basadas en documentos.
  - Almacenamiento de tuplas de valor.
  - Basadas en columnas.
  - Basadas en grafos.

Bases de Datos NoSQL - Tipos

- Basadas en documentos:
  - Se basa en schemas o esquemas.
  - Dichos schemas se representan como documents o documentos.
  - Los documentos se agrupan bajo una colección o colecciones.
  - Se conforma por un conjunto de collections.
- Como ejemplo de estas bases de datos nos encontramos con:
  - MongoDB.
  - DynamoDB.
  - CosmosDB.
  - Firestore.
  - Entre otras.

Bases de Datos NoSQL - Tipos

- Almacenamiento de tuplas llave-valor:
  - Son la base de datos más sencilla debido a que solo almacenan objetos en formato de tupla (llave-valor).
  - Suelen ser usadas como almacenamiento local o caché.
  - Suelen ser usadas para el manejo de sesiones de usuario.
- Como ejemplo de estas bases de datos nos encontramos con:
  - Redis.
  - KeyDB.
  - Entre otras.

Bases de Datos NoSQL - Tipos

- Basadas en columnas:
  - Similar a las bases de datos relacionales estas almacenan su información por columnas.
  - Esto las hace excepcionales para trabajar con análisis de datos ya que estas bases de datos se encuentran optimizadas para ello.
  - Son de uso complejo ya que no son fáciles de escalar debido a que toda su información se basa en columnas.

- Como ejemplo de estas bases de datos nos encontramos con:
  - Redshift.
  - BigQuery.
  - Snowflake.
  - Entre otras.

Bases de Datos NoSQL - Tipos

- Basadas en grafos:
  - Se encargan de relacionar nodos de información entre sí.
  - Usualmente se usan para construir redes sociales dado que un nodo (usuario) se relaciona con otros nodos mediante un grafo.
  - Así se facilita el relacionar información de uno o más nodos y hacer análisis de por ejemplo información en común como lo sería los amigos en común de un usuario.

- Como ejemplo de estas bases de datos nos encontramos con:
  - Neo4j.
  - Amazon Neptune.
  - Apache Cassandra.
  - Entre otras.

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Bases de Datos NoSQL - JSON

- En estas sesiones estaremos trabajando con bases de datos NoSQL orientadas a documentos.
- Estos documentos por lo general almacenan su información en formato JSON donde el motor de bases de datos posee tipos especiales de datos.
- Por lo que estos JSON son definidos con un schema para estructurar su información.
- Es posible definir un schema dinámico donde la estructura del JSON sea variable.

MisionTIC2022

Bases de Datos NoSQL - Firebase

- Firebase es un gestor de proyectos en la nube donde se pueden agregar varios servicios a un mismo proyecto.
- Dentro de estos servicios nos encontramos con Firestore.
- Firestore nos permite crear JSON que serán alojados a modo de base de datos.
- Firebase es una plataforma desarrollada y sostenida por Google.
- Firestore es un servicio que cuenta con una modalidad tanto paga como gratuita.

Bases de Datos NoSQL - DynamoDB

- DynamoDB es el servicio de bases de datos no relacional propio de AWS.
- Principalmente DynamoDB suele ser usada como un key-value store de rápido acceso y fácil uso.
- DynamoDB así mismo como MongoDB Atlas y Firestore cuenta con la característica de auto escalado a medida que se vayan requiriendo más recursos para su funcionamiento.
- Para más información sobre DynamoDB observar el siguiente enlace.

Bases de Datos NoSQL - MongoDB

- MongoDB es un motor de bases de datos NoSQL basado en documentos.
- MongoDB tiene una suite en la nube llamada Atlas, esta nube se conforma por alianzas con los siguientes proveedores:
  - AWS
  - Google Cloud Platform
  - Azure
- MongoDB Atlas tiene una capa donde se nos deja crear una base de datos NoSQL gratuita sin hacer ninguna clase de pago ni registrar ninguna clase de método de pago.

Bases de Datos NoSQL - Node & Mongo

- Normalmente las aplicaciones web se dividen en 3 capas por así decirlo:
  - Interfaz (Front-End).
  - Lógica de negocio (Back-End).
  - Capa de datos (Base de datos).
- Por temas de seguridad la interfaz nunca se conecta directamente con la base de datos, por lo que se tiene que comunicar con esta a través del Back-End.
- Por lo que explicaremos cómo se realiza una conexión con MongoDB, utilizando la nube atlas de forma gratuita.

Bases de Datos NoSQL - Node & Mongo

- Para eso crearemos una API en express para un to-do list, o tareas pendientes a modo de demostración.
- Tomaremos como base la plantilla que incluye la configuración de eslint, prettier y babel.
- Definiremos un archivo de configuración para la conexión a base de datos.
- Un archivo que se encargue de conectarse a la base de datos.
- Un controlador donde agrupamos toda la lógica de nuestros recursos REST.

Bases de Datos NoSQL - Node & Mongo

- Tendremos la siguiente estructura:

```
|-- src/
| |-- config/
| | `-- db.json
| |-- controllers/
| | |-- to-do/
| | | |-- guards.js
| | | |-- methods.js
| | | `-- index.js
| | `index.js
| |-- db/
| | |--client.js
| | `-- watchers.js
| `--index.js
```

- db.js: Archivo de configuración.
- guards.js: Archivo para validaciones de nuestros endpoints.
- methods.js: Archivo donde definiremos la lógica de negocio de nuestro controlador.
- to-do/index.js: Archivo donde definiremos los endpoints del controlador

Bases de Datos NoSQL - Node & Mongo

- Tendremos la siguiente estructura:

```
|-- src/
| |-- config/
| | `-- db.json
| |-- controllers/
| | |-- to-do/
| | | |-- guards.js
| | | |-- methods.js
| | | `-- index.js
| | `index.js
| |-- db/
| | |--client.js
| | `-- watchers.js
| `--index.js
```

- client.js: Conexión con la base de datos.
- watchers.js: Interrupción de la conexión con la base de datos.
- src/index.js: Definición del servidor WEB.
- Adicionalmente, ejecutaremos el comando **npm install mongodb**

Bases de Datos NoSQL - Node & Mongo

- Lo primero será definir nuestro servidor web, para esto observaremos el archivo src/index.js:
  - Primero veremos nuestros imports:

```
import express from 'express';
import compression from 'compression';
import { json, urlencoded } from 'body-parser';
import { connectToMongoDB } from './db/client';
import { setUpControllers } from './controllers';
```

- Importamos express para definir el servidor Web.
- Importamos compression para reducir nuestras peticiones.
- Importamos json y urlencoded de body-parser para que express ajuste el formato de nuestras solicitudes.
- Finalmente importamos utilidades que veremos más adelante.

Bases de Datos NoSQL - Node & Mongo

- Lo primero será definir nuestro servidor web, para esto observaremos el archivo src/index.js:
  - Ahora veremos la configuración de nuestro servidor

```
const main = async () => {
  try {
    const PORT = process.env.PORT || 3000;
    const app = express();
    await connectToMongoDB();
    app.use(compression());
    app.use(urlencoded({ extended: false }));
    app.use(json());
    app.get('/', (req, res) => res.json({ message: 'ok' }));
    setUpControllers(app);
    app.listen(PORT, () =>
      console.log(`Sevidor esperando por peticiones en localhost:${PORT}`)
    );
  } catch (error) {
    console.error(error);
  }
};
main();
```

Bases de Datos NoSQL - Node & Mongo

- Como se observamos en nuestro servidor web usamos el método connectToMongoDB, el cual se encuentra en el archivo src/db/client.js.

- Observemos nuestro archivo client.js:

```
import DB_CONFIG from '../config/db.json';
import { MongoClient } from 'mongodb';
import { setUpMongoDBProcessWatchers } from './watchers';

let mongoDBClient = null;
const listDatabases = async () => {
  // Consulta de la base de datos en nuestro cluster.
};
export const connectToMongoDB = async () => {
  // Conexion a la Base de datos.
};
export { mongoDBClient };
```

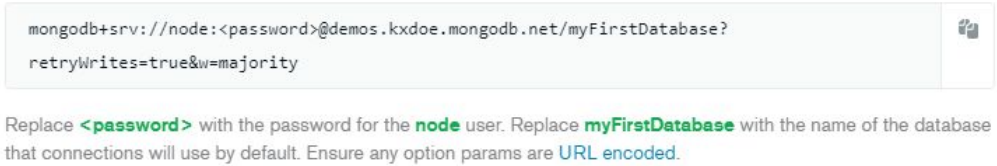
- Como podemos observar exportamos nuestro cliente y el metodo de conexion.

Bases de Datos NoSQL - Node & Mongo

- Observemos la configuración que importamos de db.json

```
{
  "CONNECTION_URL":
"mongodb+srv://<user>:<password>@demos.kxdoe.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"
}
```

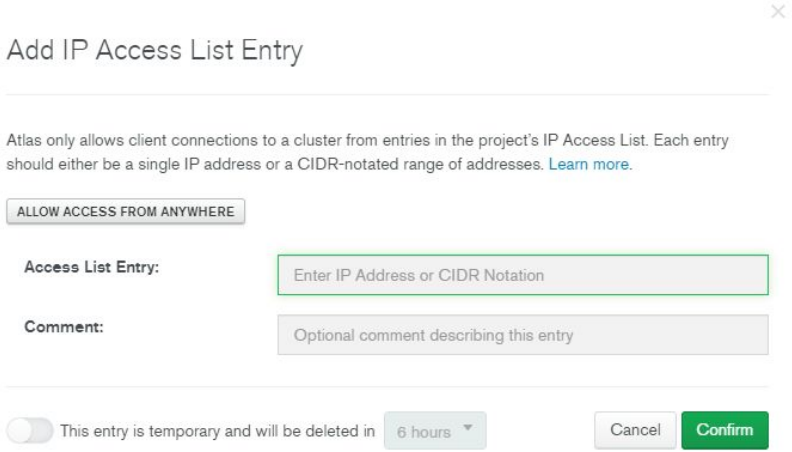
- En la consola de MongoDB Atlas, accedemos a nuestro cluster para ver esta configuración.
- Siguiendo el formulario nos encontraremos con:



- Cabe resaltar que hay que reemplazar **<user>** y **<password>**.

Bases de Datos NoSQL - Node & Mongo

- De igual forma es importante resaltar que se debe de dar acceso a la máquina o dirección IP que quiere acceder al cluster, esto se puede hacer desde la pestaña de acceso de red:



Bases de Datos NoSQL - Node & Mongo

- Observemos el método de conexión **connectToMongoDB**

```
export const connectToMongoDB = async () => {
  const connectionString = DB_CONFIG.CONNECTION_URL;
  mongoDBClient = new MongoClient(connectionString);
  try {
    console.log('Conectandose a MongoDB...');
    await mongoDBClient.connect();
    console.log('Conexion con MongoDB establecida.');
    await listDatabases();
  } catch (e) {
    console.log('Error conectandose a MongoDB');
    console.error(e);
  } finally {
    setUpMongoDBProcessWatchers();
  }
}
```

- Solo inicializamos nuestro cliente, listamos nuestras bases de datos y configuramos los eventos de interrupción de nuestro programa.

Bases de Datos NoSQL - Node & Mongo

- Para finalizar la configuración de nuestro cliente observaremos **src/db/watchers.js**:

```
import { mongoDBClient } from './client';
const gracefulShutdown = () => {
  console.log('Conexión con MongoDB cerrada.');
  mongoDBClient.close();
};
export const setUpMongoDBProcessWatchers = () => {
  // cerrar conexion en process.exit()
  process.on('exit', gracefulShutdown);
  // cerrar conexion en comandos del CLI.
  process.on('SIGINT', gracefulShutdown);
  process.on('SIGTERM', gracefulShutdown);
  process.on('SIGKILL', gracefulShutdown);
  // cerrar conexion en excepciones no controladas.
  process.on('uncaughtException', gracefulShutdown);
};
```

- Definimos un método para cerrar la conexión y lo pasamos como callback en los diferentes eventos de node.

Bases de Datos NoSQL - Node & Mongo

- Luego observamos como se definen las rutas de nuestros controladores en **src/controller/index.js:**

```
import TodoRouter from './to-do';

export const setUpControllers = (app) => {
  app.use('/to-do', TodoRouter);
};
```

- Este archivo se encarga de definir todos nuestros controladores.
- Como podemos observar creamos el controlador ‘to-do’ con el TodoRouter que es importado de la carpeta ‘./to-do’.
- Es decir que para acceder a todos los endpoints del controlador ‘to-do’, debemos de utilizar como prefijo el texto ‘to-do/’.

Bases de Datos NoSQL - Node & Mongo

- Así mismo, en el archivo **src/controllers/to-do/index.js** podemos observar la definición de nuestros endpoints:
  - Inicialmente observaremos nuestros imports:

```
import { Router } from 'express';
import { verifyTodoPayload } from './guards';
import {
  completeTodo,
  createTodo,
  deleteTodo,
  getAllTodos,
  getAllTodosByState,
  getTodoById,
  updateTodo,
} from './methods';
const router = Router();
```

- Adicionalmente definimos un router el cual será el que exportamos como controlador, intermediario o middleware de express.

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos GET:

```
router.get('/list', async (_req, res) => {
  const todos = await getAllTodos();
  res.json(todos);
});

router.get('/list-pending', async (_req, res) => {
  const todos = await getAllTodosByState(false);
  res.json(todos);
});

router.get('/list-completed', async (_req, res) => {
  const todos = await getAllTodosByState(true);
  res.json(todos);
});

router.get('/:todoId/detail', async (req, res) => {
  const detail = await getTodoById(req.params.todoId);
  res.json(detail);
});
```

- Cabe resaltar que en el último método definimos un parámetro url con el uso de '**`/:todoId`**', para más información ver este enlace.
- Estos métodos son para consultar contenido.

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos usados en los endpoint GET:

```
export const getAllTodos = async () => {
  const cursor = await mongoDBClient.db(DB).collection(COLLECTION).find();
  return await cursor.toArray();
};

export const getAllTodosByState = async (isCompleted) => {
  const cursor = await mongoDBClient.db(DB).collection(COLLECTION).find({ isCompleted });
  return await cursor.toArray();
};

export const getTodoById = async (todoId) => {
  const queryId = new ObjectId(todoId);
  const result = await mongoDBClient.db(DB).collection(COLLECTION).findOne({ _id: queryId
});
  return result;
};
```

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos POST:

```
router.post('/create', verifyTodoPayload, async (req, res) => {
  const newTodo = await createTodo(req.body);
  res.json({
    message: 'Se ha creado un nuevo todo',
    newTodoId: newTodo.insertedId,
  });
});
```

- Como podemos observar a este método /create, le pasamos de parámetro 2 funciones, esto lo que logra es que definimos por así decirlo un pipeline de funciones.
- En otras palabras la función **verifyTodoPayload** es la que se encargará de decidir si la siguiente función proporcionada como parámetro se ejecuta o no.
- Estos métodos son para agregar contenido.

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos usados en los endpoint POST:

```
export const createTodo = async (newTodo) => {
  const { createdBy, title, task } = newTodo;
  const result = await mongoDBClient.db(DB).collection(COLLECTION).insertOne({
    isCompleted: false,
    createdBy,
    title,
    task,
    createdAt: new Date(),
    completedAt: null,
  });
  return result;
};
```

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos PUT:

```
router.put('/:todoId/update', verifyTodoPayload, async (req, res) => {
  await updateTodo(req.params.todoId, req.body);
  res.json({ message: `Se ha actualizado el todo ${req.params.todoId}` });
});

router.put('/:todoId/complete', async (req, res) => {
  await completeTodo(req.params.todoId);
  res.json({ message: `Se ha completado el todo ${req.params.todoId}` });
});
```

- Estos métodos son para modificar contenido ya existente.

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos usados en los endpoint PUT:

```
export const updateTodo = async (todoId, updatedTodo) => {
  const queryId = new ObjectId(todoId);
  const { title, task } = updatedTodo;
  const result = await mongoDBClient.db(DB) .collection(COLLECTION).updateOne(
    { _id: queryId },
    { $set: { title, task } }
  );
  return result;
};

export const completeTodo = async (todoId) => {
  const queryId = new ObjectId(todoId);
  const result = await mongoDBClient.db(DB).collection(COLLECTION).updateOne(
      { _id: queryId },
      { $set: { isCompleted: true, completedAt: new Date() } }
    );
  return result;
};
```

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos DELETE:

```
router.delete('/:todoId/delete', async (req, res) => {
  await deleteTodo(req.params.todoId);
  res.json({ message: `Se ha eliminado el todo ${req.params.todoId}` });
});
```

- Estos métodos son para eliminar contenido.

Bases de Datos NoSQL - Node & Mongo

- Observaremos la definición de los métodos usados en los endpoint DELETE:

```
export const deleteTodo = async (todoId) => {
  const queryId = new ObjectId(todoId);

  const result = await mongoDBClient
    .db(DB)
    .collection(COLLECTION)
    .deleteOne({ _id: queryId });

  return result;
};
```

- Se puede ver el repositorio en este enlace.

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

Ejercicios de práctica

Mision TIC 2022

El futuro digital es de todos MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Referencias

- https://www.mongodb.com/scale/types-of-nosql-databases#document-databases
- https://www.mongodb.com/scale/types-of-nosql-databases#key-value-stores
- https://www.mongodb.com/scale/types-of-nosql-databases#column-orientated-databases
- https://www.mongodb.com/scale/types-of-nosql-databases#graph-databases
- https://aws.amazon.com/es/dynamodb/

MisionTIC2022

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

¡GRACIAS

POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Mision TIC 2022