TALLER DE CREACIÓN DE UNA RED SOCIAL CON REACT, REDUX, NODEJS, EXPRESS Y MONGODB

Danny Vaca

Índice

1. Objetivo general

Crear una base de datos Mongo desde NodeJS

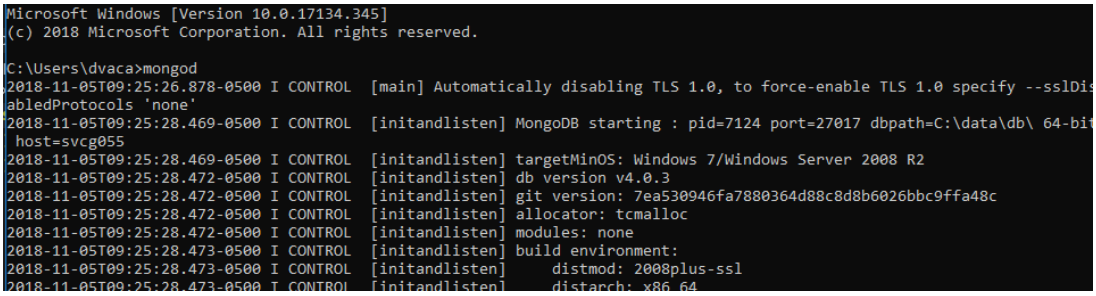
2. Objetivo específico

- Realizar una conexión desde NodeJS hasta mongo
- Crear una base de datos Mongo
- Crear colecciones de datos (tablas)
- Insertar datos en las colecciones

3. Desarrollo

3.1. Levantar el servicio de Mongo

Para levantar el servicio de Mongo es necesario correr el comando **mongod** desde una terminal, si se desea que el comando se ejecute cuando se enciende la PC, habría que registrarlo como servicio del sistema.



3.2. Creación de un proyecto

La forma más básica de crear un proyecto en NodeJS es con el comando **npm init.**

```
PS C:\Users\dvaca\Downloads\mongo\practica1> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (practica1)
version: (1.0.0)
description: practica de prueba de conexión sobre una base de datos mongo desde nodejs
entry point: (index.js)
test command:
git repository:
keywords: mongo node javascript
author: danny vaca
license: (ISC) MIT
```

Este comando nos realizará una serie de preguntas desde la terminal con lo cual nos generará un archivo llamado **package.json**

package.json

```json
{
  "name": "practica1",
  "version": "1.0.0",
  "description": "practica de prueba de conexión sobre una base de datos mongo desde nodejs",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "mongo",
    "node",
    "javascript"
  ],
  "author": "danny vaca",
  "license": "MIT"
}
```

Este archivo contiene la configuración de nuestro proyecto, el punto de entrada de nuestro proyecto es el archivo **index.js** el cual todavía no hemos creado y en el cual vamos a poner la conexión con la base de datos.

3.3. Instalación del driver de conexión

Para conectarnos a la base de datos Mongo es necesario tener un driver que se encargue de la conexión, este driver se encarga de manejar (enviar y recibir) paquetes TCP con la base de datos, escribir un driver de este tipo puede requerir de mucho tiempo, por lo que en su lugar vamos a usar el driver oficial para nodeJS, lo vamos a instalar desde la terminal usando npm, con el comando.

npm install mongodb

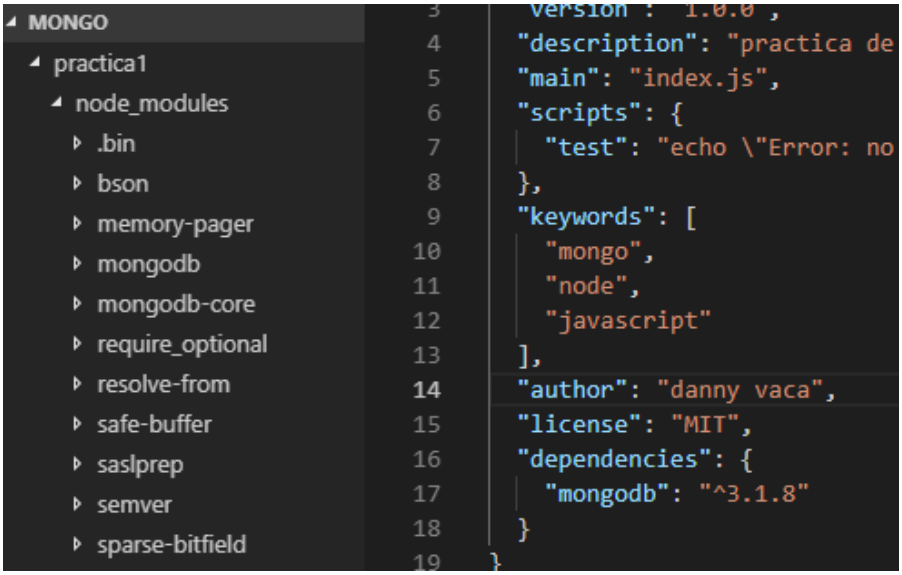
```
+ mongodb@3.1.8
added 10 packages from 7 contributors and audited 11 packages in 5.232s
found 0 vulnerabilities

PS C:\Users\dvaca\Downloads\mongo\practica1>
```

Este comando modifica el archivo **package.json** y agrega una sección llamada **dependencies** en el cual se puede ver la librería y la versión de la misma

package.json

```
{
  "name": "practica1",
  "version": "1.0.0",
  "description": "practica de prueba de conexión sobre una base de datos mongo desde nodejs",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "mongo",
    "node",
    "javascript"
  ],
  "author": "danny vaca",
  "license": "MIT",
  "dependencies": {
    "mongodb": "^3.1.8"
  }
}
```

Todas las librerías se instalan para el proyecto actual, es decir solo están disponibles para el proyecto actual y son almacenadas en una carpeta llamada **node_modules**.



Podemos apreciar que **npm** instala la librería **mongodb** y todas sus dependencias, esta es la ventaja de usar un gestor de dependencias como **npm**.

3.4. Conexión con la base de datos

En NodeJS se usa la función requiere para utilizar módulos como el instalado en el paso anterior **mongodb**, cada librería tiene su forma de utilizarse la cual debemos leer en la documentación de la librería, no existe un patrón de uso general, para crear una conexión con la base de datos mongo necesitamos un cliente de conexión y la url.

El cliente **MongoCliente** lo extraemos de la librería **mongodb** y en este cliente llamamos a la función **connect** la cual recibe una url unos parámetros de configuración y una función.

La url tiene el formato **mongodb://direcciónIP:puerto/baseDeDatos**, la **direcciónIP** es la dirección de la máquina donde hemos instalador **mongo**, el **puerto** por defecto es **27017** si se eligió otro puerto durante la instalación hay que especificar ese puerto, y finalmente la **baseDeDatos** el cual es el nombre de la base de datos que queremos crear.

El segundo argumento es un objeto ene l cual debemos especificar que se desea usar el nuevo parser asignando el valor de **true** a **useNewUrlParser**.

El último argumento que es una función, es el callback que se va a ejecutar cuando se complete la conexión, **NodeJS** es un lenguaje orientado a eventos, por eso la función **connect** no devuelve una conexión para ser utilizada, sino que recibe un callback el cual se ejecutará cuando se tenga la conexión y como argumentos recibirá el error de haber alguno y la conexión con la base de datos.

index.js

```
1   const mongodb = require('mongodb')
2   const MongoClient = mongodb.MongoClient
3   const url = 'mongodb://localhost:27017/testdatabase'
4
5   MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6       if (error) throw error
7       console.log("Database created!")
8       connection.close()
9   })
10
```

Para ejecutar el programa debemos abrir la terminal y ejecutar el comando **node index.js** el cual ejecutará el programa que acabamos de escribir.

```
PS C:\Users\dvaca\Downloads\mongo\practica1> node index.js
Database created!
PS C:\Users\dvaca\Downloads\mongo\practica1>
```

Es importante destacar que la base de datos no se crea en si hasta que no se pongo algún contenido en la misma.

3.5. Creación de colecciones

La función callback que se ejecuta cuando se establece la conexión con la base de datos nos devuelve el error de haber uno y la conexión, con esta conexión extraemos la referencia a la base de datos con la función **db** y con la instancia que nos devuelve llamamos a la función **createCollection** para crear una colección dentro de la base de datos, esta función recibe como primer argumento el nombre de la colección y una función callback para ejecutarse cuando se

haya creado la colección, esta función recibe el error de haber alguno y la referencia a la colección creada.

```
JS index.js ●
1   const mongodb = require('mongodb')
2   const MongoClient = mongodb.MongoClient
3   const url = 'mongodb://localhost:27017/testdatabase'
4
5   MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6       if (error) throw error
7       console.log("Database created!")
8       const database = connection.db()
9       database.createCollection('users', function(error, collection) {
10          if (error) throw error
11          console.log("Collection created!")
12          connection.close()
13      })
14  })
15
```

Para ejecutar el programa debemos abrir la terminal y ejecutar el comando **node index.js** el cual ejecutará el programa que acabamos de escribir.

```
PS C:\Users\dvaca\Downloads\mongo\practica1> node index.js
Database created!
Collection created!
PS C:\Users\dvaca\Downloads\mongo\practica1>
```

De la misma forma que la base de datos, hasta que no se ponga contenido en la colección esta no es creada.

3.6. Insertar documentos en la colección

Para insertar documentos en la colección necesitamos una referencia a la colección, sobre esta referencia debemos llamar a la función **insertOne** para insertar un documento, este documento es objeto javascript.

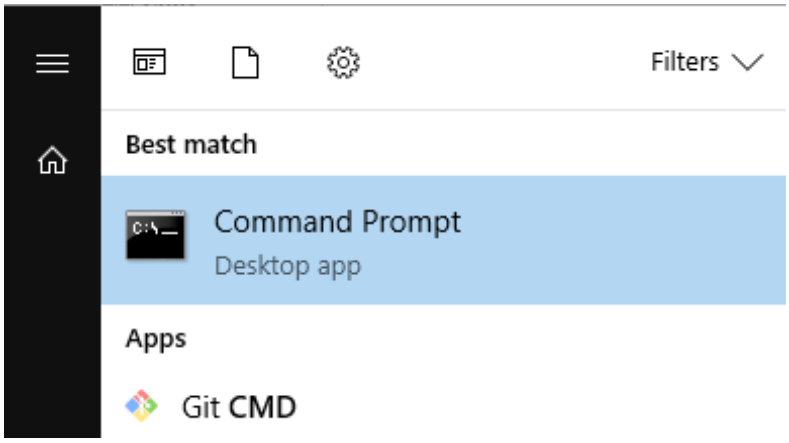
La función **insertOne** recibe como primer argumento el objeto que se desea guardar en la colección y como segundo argumento una función callback para ejecutarse cuando el documento haya sido guardado.

La función callback recibe el error de haber alguno y un **commandResult** el cual es el resultado de la ejecución del comando.

```
JS index.js ×
1   const mongodb = require('mongodb')
2   const MongoClient = mongodb.MongoClient
3   const url = 'mongodb://localhost:27017/testdatabase'
4
5   MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6       if (error) throw error
7       console.log('Database created!')
8       const database = connection.db()
9       database.createCollection('users', function(error, collection) {
10          if (error) throw error
11          console.log('Collection created!')
12          const user = {username: 'danny', email: 'danny@mail.com', name: 'Danny Vaca'}
13          collection.insertOne(user, function(error, commandResult) {
14              if (error) throw error
15              console.log('User created!')
16              connection.close()
17          })
18      })
19  })
20
```

3.7. Línea de comandos de mongo

Para verificar que los datos hayan sido ingresados correctamente procedemos a abrir una terminal en la cual ejecutamos el comando **mongo**, para acceder a la línea de comandos de mongo



```
C:\Users\dvaca>mongo
MongoDB shell version v4.0.3
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("c3ef5e7c-e24c-4ff5-83a2-8a36d93b82c3") }
MongoDB server version: 4.0.3
Server has startup warnings:
2018-11-05T09:25:30.603-0500 I CONTROL  [initandlisten]
2018-11-05T09:25:30.603-0500 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-11-05T09:25:30.605-0500 I CONTROL  [initandlisten] **          Read and write access to data and configuration is u
nrestricted.
2018-11-05T09:25:30.607-0500 I CONTROL  [initandlisten]
2018-11-05T09:25:30.607-0500 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-11-05T09:25:30.608-0500 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this ser
ver.
2018-11-05T09:25:30.608-0500 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify
 which IP
2018-11-05T09:25:30.609-0500 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --
```

3.7.1. Listar bases de datos

Con el comando **show databases** podemos ver un listado de las bases de datos, junto con el espacio en disco que estas ocupan, podemos apreciar que se creó correctamente la base de datos “testdatabase”.

```
> show databases
admin           0.000GB
config          0.000GB
local           0.000GB
social-network  0.001GB
testdatabase    0.000GB
```

Luego podemos seleccionar una base de datos para trabajar sobre ella con el comando **use**

```
> use testdatabase
switched to db testdatabase
```

3.7.2. Listar colecciones

Cuando se ha seleccionado una base de datos podemos listar las colecciones dentro de esa base de datos con el comando **show collections**

```
> show collections
users
```

3.7.3. Listar todos los documentos de una colección

Podemos usar **db** que es la referencia a la base de datos actual, para acceder a las colecciones mediante un punto, es decir con el formato **db.colección.funcion** donde colección es el nombre de la colección a la que deseamos acceder y función es la acción que deseamos realizar, para listar todos los registros usaremos la función find.

```
> db.users.find()
{ "_id" : ObjectId("5be05f8c16600a2a94f23b63"), "username" :
}
```