Use MongoDB with Node.JS

Author: Francis Ndungu

Last Updated: Thu, Sep 1, 2022 MongoDB Node.js Programming

Introduction

Node.js is a runtime environment and server-side scripting language used to build highly scalable and responsive applications like blogs, commerce websites, and corporate portals. MongoDB, like other NoSQL databases, is non-tabular. This means it stores data differently th relational databases and has data types like document, key-value, wide-column, and graph. It has a flexible schema and scales easily.

You can use the power of Node.js and MongoDB to build modern data-driven applications. This guide shows you how to implement a sam application using the Node.js MongoDB driver on Ubuntu 20.04 server.

Prerequisites

To follow along with this guide:

1. Deploy an Ubuntu 20.04 server.
2. Install MongoDB and configure an administrator account.
3. Install Node.js (Option 2 : Install via PPA Version).

1. Set a MongoDB Database

In this guide, the sample application permanently stores documents (records) in a MongoDB collection. Follow the following steps to initi the database and insert some sample documents:

1. Log in to the MongoDB server using the administrator account credentials. Replace `mongo_db_admin` with the correct username.

   ```
   $ mongosh -u mongo_db_admin -p --authenticationDatabase admin
   ```

2. Enter your MongoDB password and press `ENTER` to proceed.

   Output.

   ```
   test>
   ```

3. Run the following `use` command to create a sample `my_company` database.

   ```
   test> use my_company
   ```

   Output.

   ```
   switched to db my_company
   ```

4. Create a new `employees` collection and insert three sample documents into the document by running the following command.

   ```
   my_company> db.employees.insertMany([{
                   "employee_id" : 1,
                   "first_name" : "JOHN",
                   "last_name" : "DOE"
               },
               {
                   "employee_id" : 2,
   ```

```
        "first_name" : "MARY",
        "last_name" : "SMITH"
    },
    {
        "employee_id" : 3,
        "first_name" : "DAVID",
        "last_name" : "JACK"
    }]);
```

Output.

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62f6088b1c072dfeff3a41b8"),
    '1': ObjectId("62f6088b1c072dfeff3a41b9"),
    '2': ObjectId("62f6088b1c072dfeff3a41ba")
  }
}
```

5. Query the `employees` collection to ensure the documents are in place.

```
my_company> db.employees.find()
```

Output.

```
[
  {
    _id: ObjectId("62f6088b1c072dfeff3a41b8"),
    employee_id: 1,
    first_name: 'JOHN',
    last_name: 'DOE'
  },
  {
    _id: ObjectId("62f6088b1c072dfeff3a41b9"),
    employee_id: 2,
    first_name: 'MARY',
    last_name: 'SMITH'
  },
  {
    _id: ObjectId("62f6088b1c072dfeff3a41ba"),
    employee_id: 3,
    first_name: 'DAVID',
    last_name: 'JACK'
  }
]
```

6. Log out from the MongoDB server.

```
my_company> quit
```

After setting up the database and inserting the sample records, proceed to the next step and create a database class that allows you to in with the MongoDB database inside a Node.js code.

2. Create a `mongo_db_gateway` Class

Every Node.js application requires a separate directory to avoid mixing the source code and system files. You should put all your applicati inside that directory. For this application, you should start by coding a `mongo_db_gateway.js` file. This class file hosts methods for ma the database connection and performing other operations like inserting, finding, updating, and deleting documents.

1. Start by creating a new `project` directory.

```
$ mkdir project
```

2. Navigate to the new `project` directory.

```
$ cd project
```

3. Open a new `mongo_db_gateway.js` file in a text editor.

```
$ nano mongo_db_gateway.js
```

4. Enter the following information into the `mongo_db_gateway.js` file. Replace the `dbPass` value (`EXAMPLE_PASSWORD`) with the correct MongoDB password.

```
class mongo_db_gateway {

    connect(callBack) {

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

        const dbHost = 'localhost';
        const dbUser = 'mongo_db_admin';
        const dbPass = 'EXAMPLE_PASSWORD';
        const dbPort = 27017;
        const dbName = "my_company";

        const conString = "mongodb://" + dbUser + ":" + dbPass + "@" + dbHost + ":" + dbPort;

        MongoClient.connect(conString, function(err, dbClient) {

            if (!err) {
                const mongoDb = dbClient.db(dbName);
                callBack(null, mongoDb);
            }
        });
    }

    insertDocument(data, callBack) {

        this.connect(function (dbErr, mongoDb) {

            if (!dbErr) {

                mongoDb.collection("employees").insertOne(data, function(err, result) {
                    if (err) {
                        callBack(err, null);
                    } else {
                        callBack(null, result);
                    }
                });
            }
        });
    }

    findDocuments(resourceId, callBack) {

        this.connect(function (dbErr, mongoDb) {

            if (!dbErr) {

                var query = {};

                if (resourceId != "") {
                    query = {"employee_id": parseInt(resourceId)};
                }

                mongoDb.collection("employees").find(query).toArray(function(err, result) {
                    if (err) {
                        callBack(err, null);
                    } else {
                        callBack(null, result);
                    }
                });
            }
        });
    }

    updateDocument(resourceId, data, callBack) {

        this.connect(function (dbErr, mongoDb) {

            if (!dbErr) {

                var query = {"employee_id": parseInt(resourceId)};
```

```
                    data = {$set: data};

                    mongoDb.collection("employees").updateOne(query, data, function(err, result) {
                        if (err) {
                            callBack(err, null);
                        } else {
                            callBack(null, result);
                        }
                    });
                }
            });
        }

        deleteDocument(resourceId, callBack) {

            this.connect(function (dbErr, mongoDb) {

                if (!dbErr) {

                    var query = {"employee_id": parseInt(resourceId)};

                    mongoDb.collection("employees").deleteOne(query, function(err, result) {
                        if (err) {
                            callBack(err, null);
                        } else {
                            callBack(null, result);
                        }
                    });
                }
            });
        }
    }

    module.exports = mongo_db_gateway;
```

5. Save and close the `mongo_db_gateway.js` file.

The `mongo_db_gateway.js` file explained:

1. The `mongo_db_gateway.js` class file contains five methods illustrated below.

```
class mongo_db_gateway {

    connect(callBack) {
        ...
    }

    insertDocument(data, callBack) {
        ...
    }

    findDocuments(resourceId, callBack) {
        ...
    }

    updateDocument(resourceId, data, callBack) {
        ...
    }

    deleteDocument(resourceId, callBack) {
        ...
    }

}
```

2. The functions of the five different methods are as follows:

`connect()` : This method uses database credentials to connect to the MongoDB database using the `MongoClient.connect()` function.

`insertDocument(data, callBack)` : This methods accepts two arguments. The `data` argument is a JSON payload fro HTTP client requesting to insert a document into the `employees` collection. The `callBack` argument is a special functic runs when the `insertDocument(...)` function completes.
The `mongoDb.collection("employees").insertOne(..)` statement is the actual function that inserts the data into the MongoDB database.

`findDocuments(resourceId, callBack)` : This function is like the SQL `SELECT * FROM SAMPLE_TABLE WHERE CONDI SOME_VALUE` statement. The `findDocuments()` function accepts two values. The `resourceId` is the `employee_id` f employee you want to return from the collection. The `callBack` argument is a function that runs when the code executes. the `findDocuments(...)` function, you're examining the value of the `resourceId` to craft the suitable `query` for the `mongoDb.collection("employees").find(query).toArray(...)` function as illustrated below. An empty filter (`{` query returns all documents in the collection. Otherwise, the `{"employee_id": parseInt(resourceId)}` filter query retu specific document.

```
...
var query = {};

if (resourceId != "") {
   query = {"employee_id": parseInt(resourceId)};
}

mongoDb.collection("employees").find(query).toArray(...)
...
```

`updateDocument(resourceId, data, callBack)` : The `updateDocument(...)` function takes three arguments. The `resourceId` argument defines a filter for the document you want to update. The `data` is a JSON payload with new document values. The `callBack` argument is a function that the `updateDocument(...)` function calls after completion The `updateDocument(...)` function runs the `mongoDb.collection("employees").updateOne(query, data, ...` function to update the document matching the filter query.

`deleteDocument(resourceId, callBack)` : The `deleteDocument(...)` function takes two arguments. The `resourceId` argument allows you to craft a filter query to delete a document per the following illustration.

```
...
var query = {"employee_id": parseInt(resourceId)};

mongoDb.collection("employees").deleteOne(query...)
...
```

3. The `module.exports = mongo_db_gateway;` line at the end of the `mongo_db_gateway.js` file allows you to import the modu other Node.js files using the `require('./mongo_db_gateway.js')` statement.

Your `mongo_db_gateway.js` module is now ready. The next step shows you how to call the `mongo_db_gateway` class methods to ins find, update, and delete documents.

3. Create an `index.js` File

Every Node.js application requires an entry point that executes when the application starts. This guide uses an `index.js` file as the ent point. Follow the steps below to create the file:

1. Open a new `index.js` file in a text editor.

```
$ nano index.js
```

2. Enter the following information into the `index.js` file.

```
const http = require('http');
const url  = require("url");
const mongo_db_gateway = require('./mongo_db_gateway');

const httpHost = '127.0.0.1';
const httpPort = 8080;

const httpServer = http.createServer(httpHandler);

httpServer.listen(httpPort, httpHost, () => {
    console.log(`HTTP server running at http://${httpHost}:${httpPort}/`);
});

function httpHandler(req, res) {

    var pathname = url.parse(req.url).pathname;
    var resourcePath = pathname.split("/");

    resourceId   = "";

    if (resourcePath.length >= 3) {
        resourceId = resourcePath[2]
```

```
    }

    const dg = new mongo_db_gateway();

    switch (req.method) {

        case "POST":

            var jsonData = "";

            req.on('data', function (data) {
                jsonData += data;
            });

            req.on('end', function () {
                dg.insertDocument(JSON.parse(jsonData), callBack);
            });

            break;

        case "PUT":

            var jsonData = "";

            req.on('data', function (data) {
                jsonData += data;
            });

            req.on('end', function () {
                dg.updateDocument(resourceId, JSON.parse(jsonData), callBack);
            });

            break;

        case "DELETE":

            dg.deleteDocument(resourceId, callBack);

            break;

        case "GET":

            dg.findDocuments(resourceId, callBack);

            break;
    }

    function callBack(err, result) {

        res.writeHead(200, {'Content-Type': 'application/json'});

        if (!err) {
            res.write(JSON.stringify(result, null, 4));
        } else {
            res.write(err);
        }

        res.end();
    }
}
```

3. Save and close the `index.js` file.

The `index.js` file explained:

1. The first three lines in the `index.js` file import the modules required to run this sample application together with the custom `mongo_db_gateway.js` module you coded earlier. The `http` module provides HTTP functionalities to your application allows you to use and run the Node.js inbuilt web server. The `url` module splits the web address into different parts and returns a resource path.

```
const http = require('http');
const url  = require("url");
const mongo_db_gateway = require('./mongo_db_gateway');

...
```

2. The following two lines define the address and port for the HTTP server.

```
...

const httpHost = '127.0.0.1';
const httpPort = 8080;

...
```

3. The following line defines a new HTTP server and specifies a callback function (`httpHandler`).

```
...
const httpServer = http.createServer(httpHandler);
...
```

4. The line below instructs the web server to listen for incoming HTTP requests and print a message on the console when the code ru

```
...

httpServer.listen(httpPort, httpHost, () => {
    console.log(`HTTP server running at http://${httpHost}:${httpPort}/`);
});

...
```

5. The following line defines a `httpHandler(req, res) {...}` function.

```
...
httpHandler(req, res) {
    ...
}
...
```

6. The following lines retrieve the `resourceId` from the request URL. For instance, when an HTTP client requests the `http://127.0.0.1:8080/employees/4` URL, the logic below retrieves `4` as a `resourceId` . You're using the `resourceId` value to find, update, and delete documents by crafting a search filter that MongoDB understands.

```
...
var pathname = url.parse(req.url).pathname;
var resourcePath = pathname.split("/");

resourceId   = "";

if (resourcePath.length >= 3) {
    resourceId = resourcePath[2]
}
...
```

7. The next line in the `index.js` file initializes the `mongo_db_gateway` module you created earlier. Then, you're using the Node.js `switch(...){...}` statement to map the HTTP `req.methods` (`POST` , `GET` , `PUT` , and `DELETE`) to the correspor database functions (`dg.insertDocument(...)` , `dg.getDocuments(...)` , `dg.updateDocument(...)` , and `dg.deleteDocument(...)`).

```
...

const dg = new mongo_db_gateway();

switch (req.method) {
   ...
}
...
```

8. The `callBack()` function towards the end of the `index.js` file runs when MongoDB completes the database operations. Ther using the `res.write(JSON.stringify(result, null, 4));` statement to write a response to the calling HTTP client.

```
...

function callBack(err, result) {

    res.writeHead(200, {'Content-Type': 'application/json'});

    if (!err) {
        res.write(JSON.stringify(result, null, 4));
    } else {
        res.write(err);
    }
```

```
        res.end();
    }
    ...
```

You now have all the source code files required to run your application. The next step focuses on testing your application.

4. Test the Application

After creating the MongoDB database and coding all the source code files, your application is now ready for testing. Follow the steps belc test the application:

1. Ensure you have got the latest `npm` version.

   ```
   $ sudo npm install npm -g
   ```

   Output.

   ```
   changed 41 packages, and audited 206 packages in 4s

   ...

   found 0 vulnerabilities
   ```

2. Use `npm` to initialize your `project` directory.

   ```
   $ npm init
   ```

3. Enter the following responses when prompted.

   ```
   package name: (project) project ENTER
   version: (1.0.0) 1.0.0 ENTER
   description: Node.js and MongoDB ENTER
   entry point: (index.js) index.js ENTER
   test command: ENTER
   git repository: ENTER
   keywords: Node.js, MongoDB ENTER
   author: Test author ENTER
   license: (ISC) ENTER

   About to write to ...package.json:
   ...

   Is this OK? (yes) yes
   ```

4. Install the MongoDB Node.js driver module (`mongodb`).

   ```
   $ npm install mongodb
   ```

5. Run the application. Remember, index.js is the entry point to your application.

   ```
   $ node index.js
   ```

   The application starts a web server and displays the output below. Don't enter any other command in your active terminal window.

   ```
   HTTP server running at http://127.0.0.1:8080/
   ```

6. Establish another `SSH` connection to your server and use the Linux `curl` command to run the following test commands.

   Create a new document:

   ```
   $ curl -X POST http://127.0.0.1:8080/employees -H 'Content-Type: application/json' -d '{"employee_
   ```

Output.

```
{
    "acknowledged": true,
    "insertedId": "62f77136ebf1445162cbd181"
}
```

Retrieve all documents:

```
$ curl -X GET http://localhost:8080/employees
```

Output.

```
[
    {
        "_id": "62f770c2740ed2290a62d7ad",
        "employee_id": 1,
        "first_name": "JOHN",
        "last_name": "DOE"
    },
    {
        "_id": "62f770c2740ed2290a62d7ae",
        "employee_id": 2,
        "first_name": "MARY",
        "last_name": "SMITH"
    },
    {
        "_id": "62f770c2740ed2290a62d7af",
       "employee_id": 3,
        "first_name": "DAVID",
        "last_name": "JACK"
    },
    {
        "_id": "62f77136ebf1445162cbd181",
        "employee_id": 4,
        "first_name": "HENRY",
        "last_name": "JACKSON"
    }
]
```

Retrieve a specific document:

```
$ curl -X GET http://localhost:8080/employees/4
```

Output.

```
[
    {
        "_id": "62f77136ebf1445162cbd181",
        "employee_id": 4,
        "first_name": "HENRY",
        "last_name": "JACKSON"
    }
]
```

Update a document (`resouceId 4`):

```
$ curl -X PUT http://127.0.0.1:8080/employees/4 -H 'Content-Type: application/json' -d '{"employee
```

Output.

```
{
    "acknowledged": true,
    "modifiedCount": 1,
    "upsertedId": null,
    "upsertedCount": 0,
    "matchedCount": 1
}
```

Confirm the update command:

```
$ curl -X GET http://localhost:8080/employees/4
```

Output.

```
[
    {
        "_id": "62f770c2740ed2290a62d7ad",
        "employee_id": 4,
        "first_name": " JACK",
        "last_name": "GEOFREY"
    }
]
```

Delete a document:

```
$ curl -X DELETE http://localhost:8080/employees/2
```

Output.

```
{
    "acknowledged": true,
    "deletedCount": 1
}
```

Confirm the delete command:

```
$ curl -X GET http://localhost:8080/employees/2
```

Output.

```
[]
```

Conclusion

This guide implements the MongoDB document database using Node.js and Ubuntu 20.04 server. You've set up a database, created a cer database module, and defined an entry point to the application. Then, you've run different tests using the Linux `curl` command to inser update, and delete documents. Use the knowledge in this guide to build your next data-driven application when working with Node.js and MongoDB.