**Batch: TY A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No.\_\_7\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of MongoDB, Node.js and Express js. |

**AIM:** Implementation of MongoDB, Node.js and Express js.

**Problem Definition:**

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 4:** **Test the concepts and components of various front-end, back-end web app**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

* Mongo DB
* Connection using node js Express js And MongoDB

**MongoDB**

**MongoDB** is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. Unlike traditional relational databases that store data in rows and columns, MongoDB uses collections and documents to manage unstructured data. Each document in MongoDB is a key-value pair (BSON format, a binary JSON) that allows for flexibility in the type of data that can be stored.

**Key Features of MongoDB:**

1. **Schema-less Database**: It does not require a predefined schema. You can store different types of data structures in the same collection.
2. **Scalability**: Supports horizontal scaling, meaning you can scale the database by adding more servers.
3. **High Availability**: Supports replication, ensuring the database remains available even if a server goes down.
4. **Document-Oriented**: Data is stored in documents that can easily be nested and indexed, allowing for more efficient queries.
5. **Sharding**: Distributes data across different machines to enable horizontal scaling, maintaining high performance.
6. **Indexing**: Provides powerful indexing options that improve the speed of search operations.
7. **Aggregation**: Offers an aggregation framework that enables advanced data processing.

MongoDB is ideal for applications that require fast, iterative development and handle large volumes of data that are subject to rapid change.

**Connecting Node.js and Express.js to MongoDB**

To connect a **Node.js** application using **Express.js** to **MongoDB**, you typically use the official MongoDB driver called **Mongoose**, which is a popular Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straight-forward schema-based solution to model your data and easily interact with MongoDB.

Here are the steps for connecting MongoDB to a Node.js application using Express.js and Mongoose:

**1. Install Required Packages**

You will need to install the necessary npm packages:

npm install express mongoose

**2. Set Up the Express Server**

Create a basic Express.js application by creating a file (e.g., server.js or app.js) with the following content:

// Importing required modules

const express = require('express');

const mongoose = require('mongoose');

// Create an instance of Express

const app = express();

// Middleware to parse incoming JSON requests

app.use(express.json());

// MongoDB connection URI (Replace with your MongoDB Atlas or local URL)

const mongoURI = 'mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority';

// Connecting to MongoDB

mongoose.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch((err) => console.log('Error connecting to MongoDB:', err));

// Define a sample route

app.get('/', (req, res) => {

res.send('Hello, MongoDB with Express.js');

});

// Start the server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**3. Configure MongoDB URI**

In the above code, replace <username>, <password>, and <dbname> in the mongoURI with your MongoDB credentials and database name.

If you are running MongoDB locally, the connection string would look like this:

mongodb://localhost:27017/mydatabase

**4. Running the Application**

You can now run your Node.js application:

node app.js

After starting the server, visit http://localhost:3000/ in your browser, and you should see a message like "Hello, MongoDB with Express.js."

**5. Define Mongoose Schema and Models**

Now that you're connected to MongoDB, you can define Mongoose schemas and models to structure and interact with your database collections:

// Define a Schema

const userSchema = new mongoose.Schema({

name: String,

email: String,

age: Number

});

// Create a Model from the schema

const User = mongoose.model('User', userSchema);

// Create a new User and save it to MongoDB

app.post('/users', async (req, res) => {

const { name, email, age } = req.body;

try {

const newUser = new User({ name, email, age });

await newUser.save();

res.status(201).json({ message: 'User created', newUser });

} catch (err) {

res.status(500).json({ message: 'Error creating user', error: err });

}

});

In this code, we define a User schema with fields for name, email, and age. You can now POST user data to /users to create and store new users in MongoDB.

**6. Testing with POSTMAN or CURL**

You can test your API routes using tools like Postman or cURL. For example, to add a user, make a POST request to http://localhost:3000/users with a JSON body:

{

"name": "John Doe",

"email": "john@example.com",

"age": 25

}

**Implementation Details:**

App.js

import React, { useState, useEffect } from 'react'; import './App.css';

function App() { const [task, setTask] = useState(''); const [tasks, setTasks] = useState([]);

useEffect(() => { fetchTasks();

}, []);

const fetchTasks = async () => { const response = await

fetch('http://localhost:5000/api/tasks'); const

data = await response.json(); setTasks(data);

};

const handleAddTask = async () => { const response = await

fetch('http://localhost:5000/api/tasks', { method:

'POST', headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ task })

});

const data = await response.json(); setTasks([...tasks, data]); setTask('');

};

const handleDeleteTask = async (id) => { await

fetch(`http://localhost:5000/api/tasks/${id}`, {

method: 'DELETE'

});

const updatedTasks = tasks.filter(task => task.\_id

!== id); setTasks(updatedTasks);

};

return (

<div className="App">

<h1>Task List</h1>

<div>

<input type="text"

placeholder="Enter a task" value={task}

onChange={(e) => setTask(e.target.value)}

/>

<button onClick={handleAddTask}>Add Task</button>

</div>

<ul>

{tasks.map((task) => (

<li key={task.\_id}>

{task.task}{' '}

<button onClick={() => handleDeleteTask(task.\_id)}>Delete</button>

</li>

))}

</ul>

</div>

);

}

export default App;

server.js

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

const dotenv = require('dotenv') dotenv.config();

const app = express(); const PORT

= 5000;

const uri = process.env.MONGODB\_URI;

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

mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true

});

const taskSchema = new mongoose.Schema({ task: { type: String,

required: true

}

});

const Task = mongoose.model('Task', taskSchema);

app.post('/api/tasks', async (req, res) => { const { task } = req.body;

const newTask = new Task({ task

}); await newTask.save(); res.json(newTask);

});

app.get('/api/tasks', async (req, res) => { const tasks = await Task.find(); res.json(tasks);

});

app.delete('/api/tasks/:id', async (req, res) =>

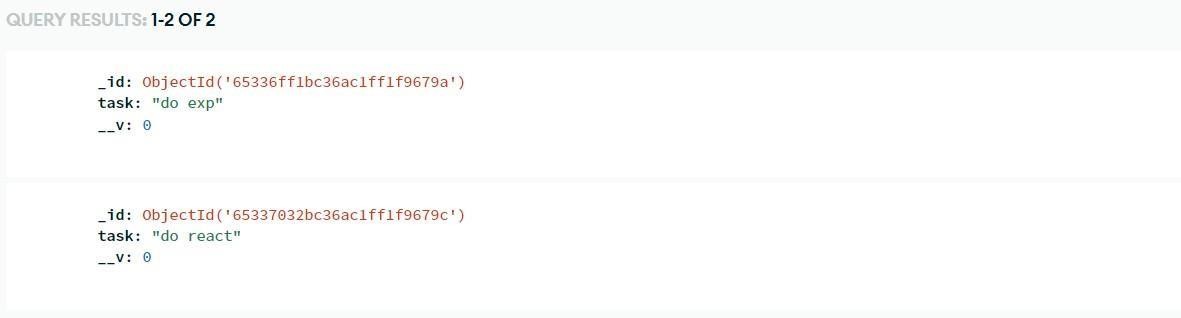
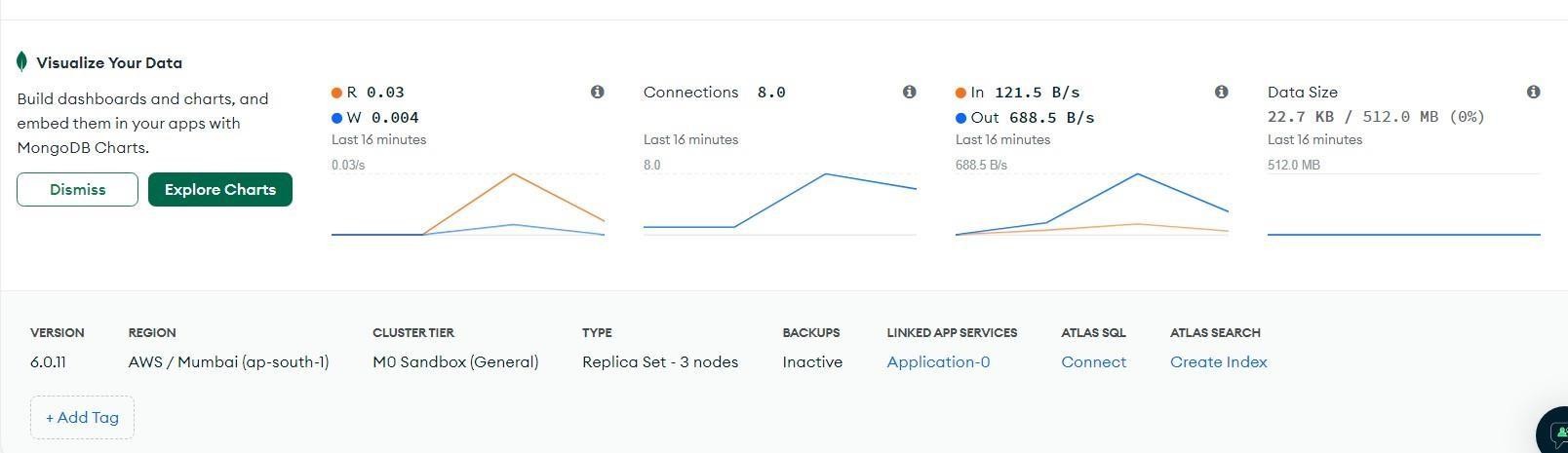
{ const { id } = req.params; await Task.findByIdAndDelete(id);

res.json({ message: 'Task deleted successfully' }); });

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`); });

Mongo dB overview



Output:



**Steps for execution:**

**Conclusion:**

Learnt and implemented connections using MongoDB, Express and React