**Batch: C1**

**Roll No.: 16010122221**

**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:**

The objective of this experiment is to explore and test the core concepts and components involved in the development of a full-stack web application, utilizing various front-end and back-end technologies. This includes understanding how to integrate Node.js for server-side logic, Express.js for building RESTful APIs, and MongoDB for data storage, while implementing best practices for effective web application architecture.

**Resources used:**

**Development Tools:**

* Code Editor (e.g., Visual Studio Code)
* Postman (for API testing)
* MongoDB Atlas (for cloud database hosting)

**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.

**Websites:**

* Node.js Official Documentation: Node.js Docs
* Express.js Official Documentation: [Express.js Docs](https://expressjs.com/)
* MongoDB Official Documentation: [MongoDB Docs](https://docs.mongodb.com/)

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

* Mongo DB

**Overview:** MongoDB is a popular NoSQL database designed for high performance, scalability, and flexibility. Unlike traditional relational databases, MongoDB stores data in a JSON-like format called BSON (Binary JSON), allowing for dynamic schemas and the ability to handle unstructured data. Key features of MongoDB include:

* **Document-Based Storage:** Data is stored in documents, which can contain arrays and nested objects, making it easy to represent complex data structures.
* **Scalability:** MongoDB supports horizontal scaling through sharding, allowing you to distribute data across multiple servers.
* **Rich Query Language:** MongoDB provides a powerful query language that supports ad-hoc queries, indexing, and real-time aggregation.
* **High Availability:** Through replica sets, MongoDB ensures data redundancy and fault tolerance, providing automatic failover in case of server failures.
* **Flexible Schema:** The schema can be modified on-the-fly, allowing for agile development and easy adaptation to changing requirements.
* Connection using node js Express js And MongoDB

To connect a Node.js application to MongoDB using Express.js, follow these steps:

1. *Prerequisites*
   * Ensure you have Node.js installed on your machine.
   * Set up a MongoDB database instance. You can use MongoDB Atlas for a cloud solution or install MongoDB locally.
2. *Create a New Node.js Project*
3. Create a new directory for your project and navigate into it:
4. Initialize a new Node.js application:
5. Install the required packages:
6. *Set Up Your Project Structure*

Create the following files in your project directory:

* + server.js (for the server logic)
  + .env (for environment variables)

1. *Configure the .env File*

In your .env file, define your MongoDB connection string and port: MONGO\_URI="your\_mongodb\_connection\_string"

PORT=4000

Replace your\_mongodb\_connection\_string with your actual MongoDB URI (e.g., from MongoDB Atlas).

1. *Create the Server in server.js*

In server.js, set up the Express server and connect to MongoDB: const express = require("express");

const mongoose = require("mongoose"); require("dotenv").config();

const app = express();

app.use(express.json()); // To parse JSON requests

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

const mongoURI = process.env.MONGO\_URI;

if (!mongoURI) {

console.error("MongoDB URI is not defined. Please check your .env file."); process.exit(1);

}

// Connect to MongoDB mongoose

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

.then(() => console.log("Database connected successfully"))

.catch((err) => {

console.error("MongoDB connection error:", err); process.exit(1);

});

// Define a simple route app.get("/", (req, res) => {

res.send({ message: "API working" });

});

// Start the server app.listen(port, () => {

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

});

1. *Run Your Application*
2. Start your Node.js application: node server.js
3. Open your browser or an API client like Postman and send a GET request to http://localhost:4000/. You should receive a response indicating that the API is working.

**Methodology and Implementation Details:**

## Install Node, Express, and MongoDB

First, ensure that you have Node.js and MongoDB installed on your machine. Then, create a new project directory and navigate to it.

## Initialize Node Application

In your project directory, run the following command to initialize a new Node.js application:

Bash – npm I

# Create your server

const express = require("express"); const mongoose = require("mongoose"); require("dotenv").config();

const port = process.env.PORT || 3000; const mongoURI = process.env.MONGO\_URI;

if (!mongoURI) {

console.error("MongoDB URI is not defined. Please check your .env file."); process.exit(1);

}

const app = express(); app.use(express.json());

app.get("/", (req, res) => { res.send({ message: "API working" });

});

mongoose

.connect(mongoURI, { serverSelectionTimeoutMS: 5000,

})

.then(() => console.log("Database connected"))

.catch((err) => {

console.log("MongoDB connection error: ", err); process.exit(1);

});

app.listen(port, () => {

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

});

1. **Define mongoose model**

const mongoose = require("mongoose");

require("dotenv").config();

const connection = mongoose.connection(); module.exports = {

connection,

};

1. **Create Your .env File**

In the root of your project, create a file named .env and add your MongoDB URI and port configuration:

MONGO\_URI="mongodb+srv://sharwarp:password@cluster0.u5pu5.mongodb.net/?retryWrites=true &w=majority&appName=Cluster0"

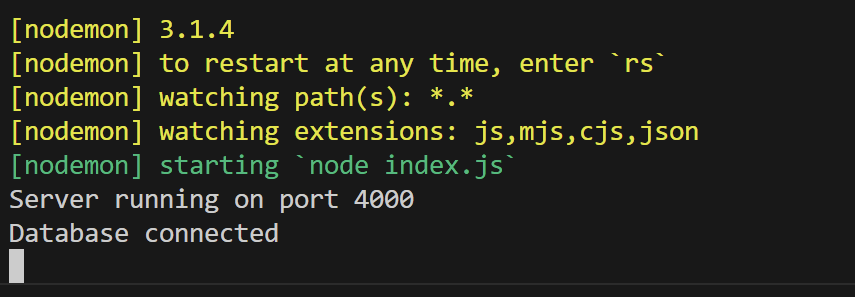
PORT=4000

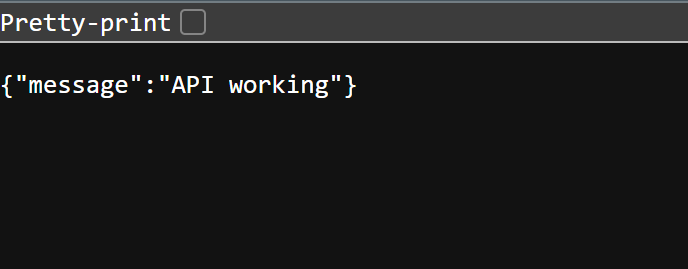
# Checking if Server is Working

app.get("/", (req, res) => { res.send({ message: "API working" });

});

**Output**





**Steps for execution:**

* 1. Open your terminal and navigate to your project directory
  2. Start the server by running: Nodemon app.js
  3. Open your browser or an API client (like Postman) and send a GET request to http://localhost:4000/

**Conclusion:** In conclusion, this setup provides a robust foundation for developing a Node.js application with Express and MongoDB, ensuring efficient API management and database connectivity. With the server running successfully, you can now build upon this structure to implement additional features and functionalities.