**Experiment No.: 09**

**Title:** Demonstrate CURD operation on Mongodb

**Objectives:**

1. To demonstrate middlewares in expressjs.

**Theory:**

MongoDB is a popular open-source NoSQL database that uses a document-oriented data model. Instead of using tables and rows like in traditional relational databases, MongoDB stores data in flexible, JSON-like documents with dynamic schemas, making it easy to store and retrieve complex data structures.

Key features of MongoDB include:

* Document-Oriented: MongoDB stores data in flexible, JSON-like documents called BSON (Binary JSON). Each document can have its own unique structure, making it easy to represent complex data.
* Scalable: MongoDB is designed to scale out horizontally across multiple servers, allowing it to handle large volumes of data and high throughput applications.
* High Availability: MongoDB provides built-in replication and automatic failover capabilities, ensuring data availability and reliability.
* Schemaless: MongoDB's dynamic schema allows you to change the structure of your documents without modifying the existing data, providing flexibility and agility in application development.
* Rich Query Language: MongoDB supports a powerful query language with support for ad-hoc queries, indexing, aggregation, and geospatial queries.

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a higher-level abstraction over the MongoDB Node.js driver, making it easier to work with MongoDB in Node.js applications.

Key features of Mongoose include:

* Schema Definition: Mongoose allows you to define schemas for your MongoDB documents, including data types, validation rules, and default values.
* Middleware: Mongoose supports middleware functions that allow you to intercept and modify document operations like validation, saving, and querying.
* Query Building: Mongoose provides a fluent API for building MongoDB queries, making it easy to perform CRUD operations and complex queries.
* Data Validation: Mongoose allows you to define validation rules for your schemas, ensuring that data is consistent and meets the specified criteria.
* Population: Mongoose supports population, which allows you to reference documents in other collections and automatically retrieve their data when querying.

To create, read, update, and delete (CRUD) records in MongoDB, you can use the MongoDB shell or a programming language like JavaScript (Node.js) with a MongoDB driver such as mongodb or Mongoose. Here is example of how to perform CRUD operations using the mongodb package in Node.js.

**1. Install the MongoDB Node.js Driver:**

First, you need to install the mongodb package in your Node.js project:

*npm install mongodb*

**2. Connect to MongoDB:**

Create a JavaScript file (e.g., app.js) and establish a connection to your MongoDB database:

*const { MongoClient } = require('mongodb');*

*const uri = 'mongodb://localhost:27017'; // MongoDB connection URI*

*async function connectToMongoDB() {*

*try {*

*const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });*

*await client.connect();*

*console.log('Connected to MongoDB');*

*return client;*

*} catch (error) {*

*console.error('Error connecting to MongoDB:', error);*

*process.exit(1);*

*}*

*}*

*// Call the function to connect to MongoDB*

*const client = await connectToMongoDB();*

3. Insert a Record:

To create (insert) a new record into a MongoDB collection, you can use the insertOne() method:

const db = client.db('my\_database'); // Specify the name of your database

const collection = db.collection('my\_collection'); // Specify the name of your collection

const document = { name: 'John', age: 30, city: 'New York' };

const result = await collection.insertOne(document);

console.log('Inserted document:', result.ops[0]);

**4. Read Records:**

To read (retrieve) records from a MongoDB collection, you can use methods like find():

const documents = await collection.find({}).toArray();

console.log('Retrieved documents:', documents);

**5. Update a Record:**

To update an existing record in a MongoDB collection, you can use methods like updateOne() or updateMany():

const filter = { name: 'John' };

const update = { $set: { age: 35 } }; // Update the age field

const result = await collection.updateOne(filter, update);

console.log('Updated document:', result.modifiedCount);

**6. Delete a Record:**

To delete a record from a MongoDB collection, you can use methods like deleteOne() or deleteMany():

const filter = { name: 'John' };

const result = await collection.deleteOne(filter);

console.log('Deleted document:', result.deletedCount);

We can also make connect from index.js.

**1.Set Up Express.js and MongoDB Connection:**

First, set up an Express.js application and establish a connection to MongoDB. Create a file named app.js:

const express = require('express');

const { MongoClient } = require('mongodb');

const app = express();

const port = 3000;

const uri = 'mongodb://localhost:27017'; // MongoDB connection URI

const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

async function connectToMongoDB() {

try {

await client.connect();

console.log('Connected to MongoDB');

} catch (error) {

console.error('Error connecting to MongoDB:', error);

process.exit(1);

}

}

connectToMongoDB();

app.listen(port, () => {

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

});

2. Create Routes for CRUD Operations:Define routes in Express.js for each CRUD operation (create, read, update, delete):

// Insert a new record

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

try {

const db = client.db('my\_database');

const collection = db.collection('my\_collection');

const result = await collection.insertOne(req.body);

res.status(201).send(result.ops[0]);

} catch (error) {

res.status(500).send(error);

}

});

// Retrieve all records

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

try {

const db = client.db('my\_database');

const collection = db.collection('my\_collection');

const documents = await collection.find({}).toArray();

res.send(documents);

} catch (error) {

res.status(500).send(error);

}

});

// Update a record

app.put('/items/:id', async (req, res) => {

try {

const db = client.db('my\_database');

const collection = db.collection('my\_collection');

const filter = { \_id: req.params.id }; // Assuming item ID is passed in the request URL

const update = { $set: req.body }; // Update the entire document with the request body

const result = await collection.updateOne(filter, update);

res.send(result.modifiedCount ? 'Record updated successfully' : 'Record not found');

} catch (error) {

res.status(500).send(error);

}

});

// Delete a record

app.delete('/items/:id', async (req, res) => {

try {

const db = client.db('my\_database');

const collection = db.collection('my\_collection');

const filter = { \_id: req.params.id }; // Assuming item ID is passed in the request URL

const result = await collection.deleteOne(filter);

res.send(result.deletedCount ? 'Record deleted successfully' : 'Record not found');

} catch (error) {

res.status(500).send(error);

}

});

3. Start the Express.js Server:Start the Express.js server:

app.listen(port, () => {

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

});

**Key Concept:** Mongodb, Mongoose

**Steps:**

1. Create folder with name myapp and execute npm init in the command prompt at the folder location

2. Create index.js file with db connection and write CURD operation for db handling.

3. Listen on port no 3000

3. Use postman application to send the request to <http://localhost:3000> with method GET, and POST.

4. Verify the result.