Experiment – 7: MongoDB

Name of Student	Ronak Katariya
Class Roll No	D15A /23
D.O.P.	
D.O.S.	
Sign and Grade	

- 1) Aim: To study CRUD operations in MongoDB
- 2) Problem Statement:
 - A) Create a new database to storage student details of IT dept(Name, Roll no, class name) and perform the following on the database
 - a) Insert one student details
 - b) Insert at once multiple student details
 - c) Display student for a particular class
 - d) Display students of specific roll no in a class
 - e) Change the roll no of a student
 - f) Delete entries of particular student
 - B) Create a set of RESTful endpoints using Node.js, Express, and Mongoose for handling student data operations.

The endpoints should support:

- Retrieve a list of all students.
- Retrieve details of an individual student by ID.
- Add a new student to the database.
- Update details of an existing student by ID.
- Delete a student from the database by ID.

Connect the server to MongoDB using Mongoose, and store student data with attributes: name, age, and grade.

MongoDB and RESTful API Integration Introduction

MongoDB is a popular NoSQL database that stores data in a flexible, document-oriented format called BSON (Binary JSON). Its schema-less nature makes it ideal for modern applications that deal with unstructured or semi-structured data. When paired with RESTful APIs, MongoDB provides a powerful mechanism for seamless communication between web applications and databases, leveraging standard HTTP methods for CRUD (Create, Read, Update, Delete) operations. This integration is particularly beneficial for web and mobile applications due to its efficiency and scalability.

RESTful Architecture with MongoDB

REST (Representational State Transfer) is an architectural style that enables stateless communication between clients and servers over HTTP. RESTful APIs use standard methods like GET, POST, PUT, PATCH, and DELETE to interact with resources. When integrated with MongoDB:

- Data is exchanged in JSON format, which aligns well with MongoDB's document model.
- REST endpoints provide a structured way to manipulate and retrieve data stored in MongoDB collections.

CRUD Operations in MongoDB Using RESTful Endpoints Create Operation (POST Method)

The Create operation inserts new documents into a MongoDB collection. In a RESTful API:

- The POST method is used to send data to the server.
- Example endpoint:

bash

POST /api/products

This endpoint accepts JSON payloads containing product details and inserts them into the database.

Read Operation (GET Method)

The *Read* operation retrieves data from MongoDB collections. The **GET** method is used for fetching documents:

Fetch all documents:

bash

GET /api/products

• Fetch a specific document by ID:

bash

GET /api/products/:id

Update Operation (PUT/PATCH Methods)

The *Update* operation modifies existing documents in the collection:

PUT replaces the entire document.

bash

PUT /api/products/:id

PATCH updates specific fields of a document.

bash

PATCH /api/products/:id

Delete Operation (DELETE Method)

The *Delete* operation removes documents from the collection using the **DELETE** method:

Delete a specific document by ID:

bash

DELETE /api/products/:id

Advantages of Using MongoDB with RESTful APIs

- 1. **Flexibility:** MongoDB's schema-less design allows dynamic changes to data structure without migration.
- 2. **Scalability:** MongoDB supports horizontal scaling through sharding, making it suitable for large-scale applications.
- 3. **Ease of Integration:** JSON-based REST APIs align seamlessly with MongoDB's BSON format.
- 4. **Efficiency:** RESTful APIs simplify CRUD operations while maintaining statelessness.

Implementation Example

Using Node.js and Express.js as the backend framework:

- 1. Set up a connection to MongoDB using the MongoClient.
- 2. Define RESTful endpoints for CRUD operations.
- 3. Example code snippet for creating a product:

```
javascript
```

```
app.post("/api/products", (req, res) =>
    { collection.insertOne(req.body, (err, result) => {
        if (err) return res.status(500).send(err);
        res.send(result);
    });
});
```

4. Test endpoints using tools like Postman or curl.

Output:

A)

```
show dbs
 admin
            40.00 KiB
 config
            72.00 KiB
 inventory 72.00 KiB
            72.00 KiB
 local
 use IT_Dept
 switched to db IT_Dept
 db.students.insertOne({
    name: "John Doe",
    roll_no: 101,
    class_name: "IT-A"
 })
< €
   acknowledged: true,
   insertedId: ObjectId('67db92f42caef4eeddce6aaa')
```

```
> db.students.insertMany([
     { name: "Alice", roll_no: 102, class_name: "IT-A" },
     { name: "Bob", roll_no: 103, class_name: "IT-B" },
     { name: "Charlie", roll_no: 104, class_name: "IT-A" }
 1)
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('67db92fa2caef4eeddce6aab'),
      '1': ObjectId('67db92fa2caef4eeddce6aac'),
      '2': ObjectId('67db92fa2caef4eeddce6aad')
    }
 1
> db.students.find().pretty()
> db.students.find().pretty()
< {
   _id: ObjectId('67db92f42caef4eeddce6aaa'),
   name: 'John Doe',
   roll_no: 101,
   class_name: 'IT-A'
 }
   _id: ObjectId('67db92fa2caef4eeddce6aab'),
   name: 'Alice',
   roll_no: 102,
   class_name: 'IT-A'
 }
   _id: ObjectId('67db92fa2caef4eeddce6aac'),
   name: 'Bob',
   roll_no: 103,
   class_name: 'IT-B'
 }
   _id: ObjectId('67db92fa2caef4eeddce6aad'),
   name: 'Charlie',
   roll_no: 104,
   class_name: 'IT-A'
```

```
> db.students.find({ class_name: "IT-A" })
< {
   _id: ObjectId('67db92f42caef4eeddce6aaa'),
   name: 'John Doe',
   roll_no: 101,
   class_name: 'IT-A'
 }
   _id: ObjectId('67db92fa2caef4eeddce6aab'),
   name: 'Alice',
   roll_no: 102,
   class_name: 'IT-A'
 }
   _id: ObjectId('67db92fa2caef4eeddce6aad'),
   name: 'Charlie',
   roll_no: 104,
   class_name: 'IT-A'
> db.students.find({ roll_no: 102, class_name: "IT-A" })
< {
   _id: ObjectId('67db92fa2caef4eeddce6aab'),
   name: 'Alice',
   roll_no: 102,
```

```
> db.students.find({ roll_no: 102, class_name: "IT-A" })

< {
    _id: ObjectId('67db92fa2caef4eeddce6aab'),
    name: 'Alice',
    roll_no: 102,
    class_name: 'IT-A'
}

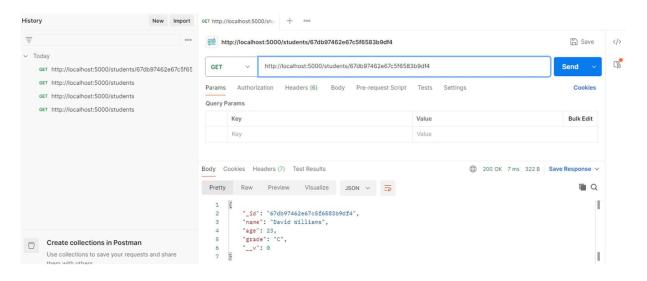
> db.students.updateOne(
    { name: "John Doe" }, # Find condition
    { $set: { roll_no: 110 } } # Update action
)

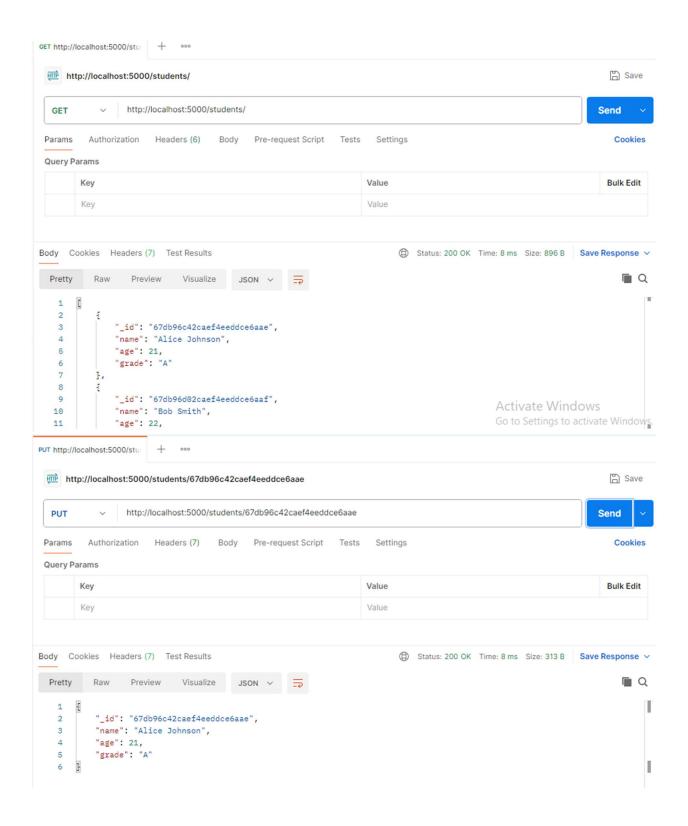
> db.students.updateMany(
```

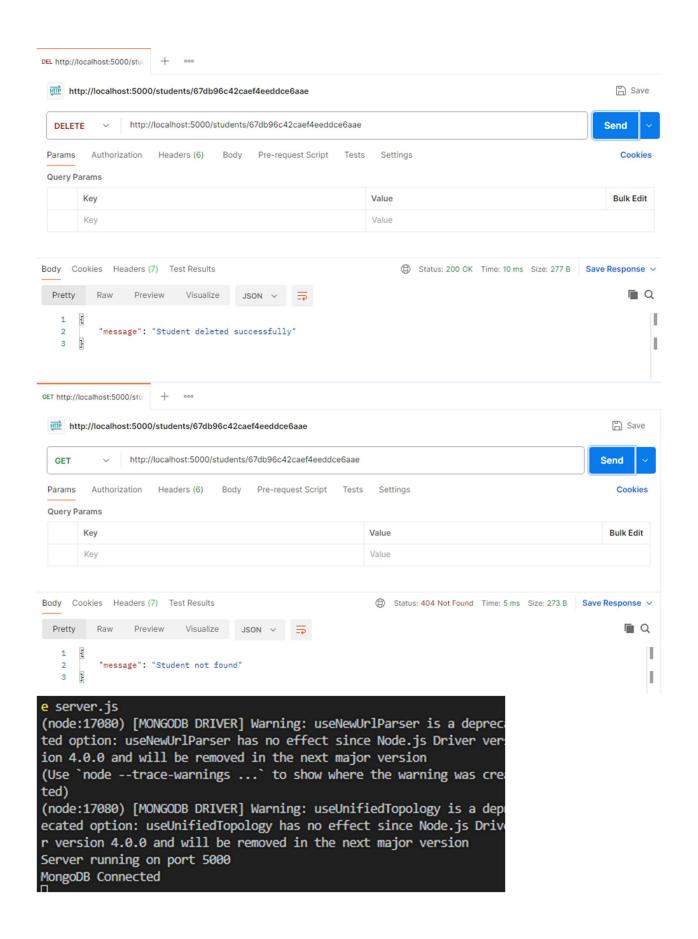
```
> db.students.updateMany(
    { class_name: "IT-A" },
    { $set: { class_name: "IT-C" } }
< {
   acknowledged: true,
   insertedId: null,
   matchedCount: 3,
   modifiedCount: 3,
   upsertedCount: 0
> db.students.deleteOne({ name: "John Doe" })
< €
   acknowledged: true,
   deletedCount: 1
> db.students.deleteMany({ class_name: "IT-A" })
< {
   acknowledged: true,
   deletedCount: 0
```

```
> show collections
< students
> db.students.drop()
< true
IT_Dept>
```

B)







Add student

```
const mongoose = require("mongoose");
const Student = require("./models/Student"); // Import the model
  .connect("mongodb://127.0.0.1:27017/studentDB", { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error(err));
  { name: "Alice Johnson", age: 21, grade: "A" },
  { name: "Bob Smith", age: 22, grade: "B" },
  { name: "Charlie Brown", age: 20, grade: "A" },
  { name: "David Williams", age: 23, grade: "C" }
Student.insertMany(students)
  .then(() \Rightarrow
    { console.log("Students
    Added!");
    mongoose.connection.close();
  .catch(err => console.error(err));
 PS C:\Users\Student\web\student-api> node addStudents.js
  (node:17960) [MONGODB DRIVER] Warning: useNewUrlParser is a deprec
 ted option: useNewUrlParser has no effect since Node.js Driver ver
  ion 4.0.0 and will be removed in the next major version
  (Use `node --trace-warnings ... ` to show where the warning was cre
 ted)
  (node:17960) [MONGODB DRIVER] Warning: useUnifiedTopology is a dep
 ecated option: useUnifiedTopology has no effect since Node.js Driv
 r version 4.0.0 and will be removed in the next major version
                                           Activate Window
 MongoDB Connected
 Students Added!
                                           Go to Settings to activ
```

```
const mongoose = require("mongoose");

const StudentSchema = new
  mongoose.Schema({ name: { type: String,
    required: true }, age: { type: Number,
    required: true }, grade: { type: String,
    required: true }
});

module.exports = mongoose.model("Student", StudentSchema);
```

Routes

```
const express = require("express");
const router = express.Router();
const Student = require("../models/Student");
router.get("/", async (req, res) => {
   const students = await Student.find();
    res.json(students);
    res.status(500).json({ message: err.message });
});
// {\mathbb I} 2. Get student by ID {\sf router.get("/:id",}
async (req, res) \Rightarrow {
    const student = await Student.findById(req.params.id);
    if (!student) return res.status(404).json({ message: "Student not
found" });
    res.json(student);
  } catch (err) {
    res.status(500).json({ message: err.message });
});
router.post("/", async (req, res) => {
```

```
const { name, age, grade } = req.body;
  const student = new Student({ name, age, grade });
   const newStudent = await student.save();
    res.status(201).json(newStudent);
  } catch (err) {
    res.status(400).json({ message: err.message });
});
// {
m I} 4. Update student by ID {
m router.put}("/:{
m id}",
async (req, res) => {
    const updatedStudent = await Student.findByIdAndUpdate(req.params.id,
req.body, { new: true });
    if (!updatedStudent) return res.status(404).json({ message: "Student
not found" });
    res.json(updatedStudent);
    res.status(400).json({ message: err.message });
router.delete("/:id", async (req, res) => {
    const deletedStudent = await Student.findByIdAndDelete(req.params.id);
    if (!deletedStudent) return res.status(404).json({ message: "Student
not found" });
    res.json({ message: "Student deleted successfully" });
  } catch (err) {
    res.status(500).json({ message: err.message });
});
```

Server

```
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const studentRoutes = require("./routes/studentRoutes");

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(bodyParser.json());
app.use("/students", studentRoutes);

// MongoDB Connection
mongoose
   .connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true })
   .then(() => console.log("MongoDB Connected"))
   .catch(err => console.error(err));

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Conclusion

Integrating MongoDB with RESTful APIs provides developers with an efficient and scalable solution for managing application data. By leveraging HTTP methods and JSON-based communication, developers can build robust web applications capable of handling diverse data structures and high traffic loads.