EXPERIMENT NO. 7

Name of Student	**Shreyash Ganesh Dhekane**
Class Roll No	**16**
D.O.P.	**25-03-2025**
D.O.S.	**01-04-2025**
Sign and Grade	

AIM:

To study CRUD operations in MongoDB

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.

THEORY:

Introduction

MongoDB is a widely used NoSQL database that stores data in a flexible, document-based format (BSON). When integrated with RESTful APIs, it enables seamless communication between web applications and databases using standard HTTP methods. This approach allows developers to perform **CRUD (Create, Read, Update, Delete) operations** efficiently, making it ideal for modern web and mobile applications.

RESTful Architecture with MongoDB

REST (Representational State Transfer) is an architectural style that allows applications to interact over HTTP using a **stateless, client-server** model. When combined with MongoDB, REST APIs enable **easy data manipulation and retrieval** in JSON format.

CRUD Operations in MongoDB using RESTful Endpoints

1) Create Operation (POST Method)

   The **Create operation** is used to insert a new document into a MongoDB collection. In a RESTful API, the **POST** method is used to send data to the server, which then saves it in the database.

   POST /api/products

2) Read Operation (GET Method)

   The **Read operation** is used to retrieve data from MongoDB. The **GET** method in a RESTful API allows fetching either all documents or a specific document using an ID.

   Fetch all products:
   GET /api/products

   Fetch products by id:
   GET /api/products/:id

3) Update Operation (PUT/PATCH Method)

   The **Update operation** is used to modify existing documents in a MongoDB collection. In a RESTful API, we use:
   - **PUT** → Replaces the entire document.
   - **PATCH** → Updates only specific fields.

   Update entire product (PUT method)
   PUT /api/products/:id

   Update specific fields of a product (PATCH method)
   PATCH /api/products/:id

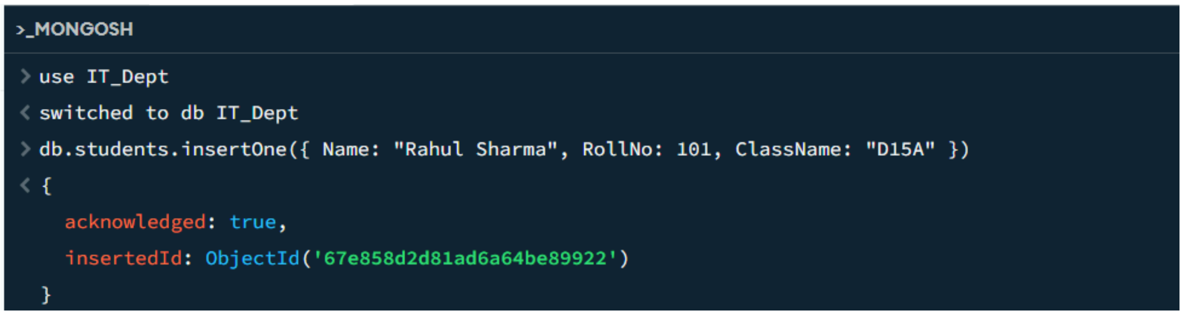
4) Delete Operation (DELETE Method)

   The **Delete operation** is used to remove a document from a MongoDB collection. In a RESTful API, the **DELETE method** is used to delete records based on a unique identifier.

DELETE /api/products/:id

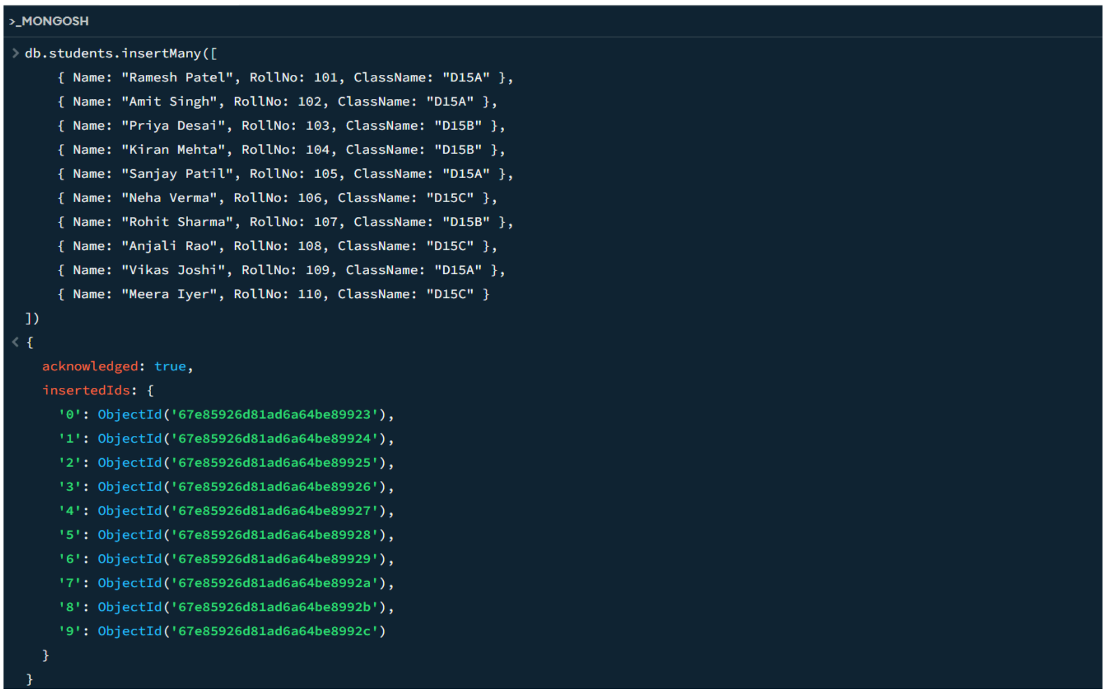
OUTPUT: -

- Create a database named IT_Dept and create a collection named "students" with the fields and Insert one student details



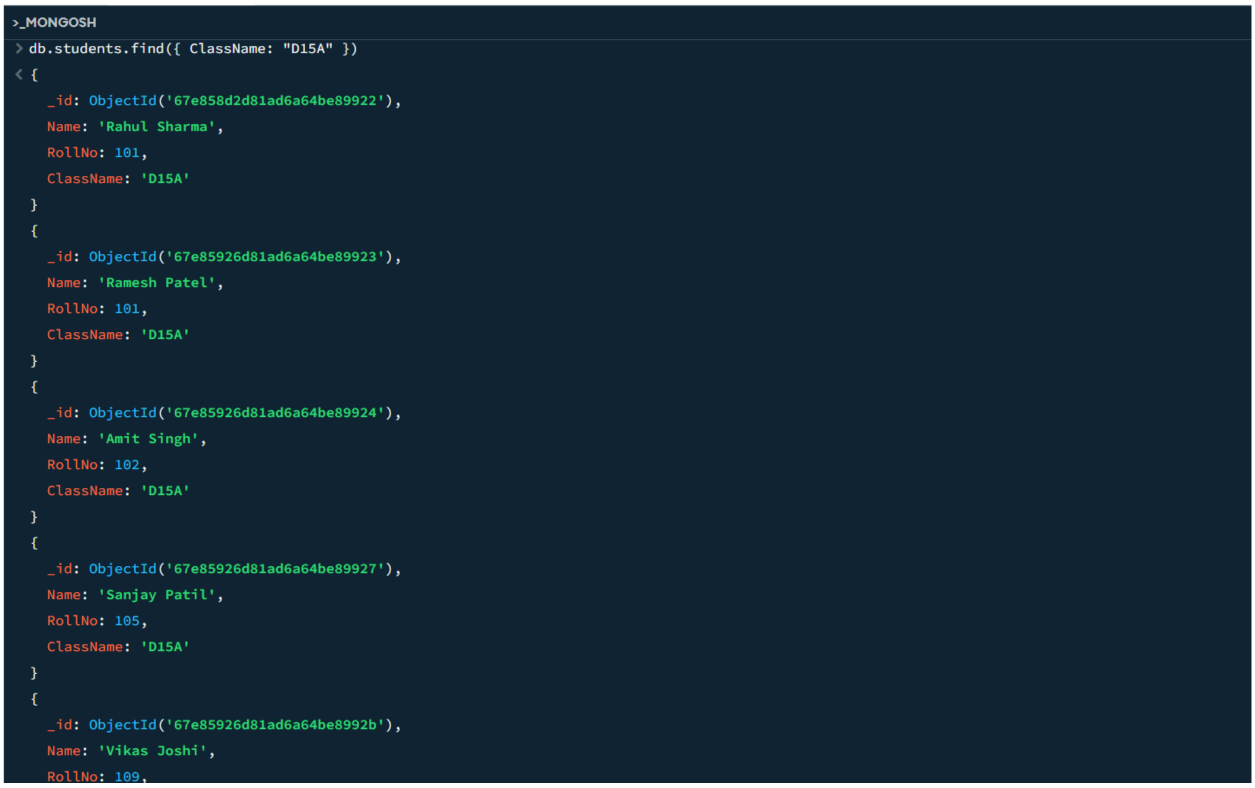
```
>_MONGOSH
> use IT_Dept
< switched to db IT_Dept
> db.students.insertOne({ Name: "Rahul Sharma", RollNo: 101, ClassName: "D15A" })
< {
    acknowledged: true,
    insertedId: ObjectId('67e858d2d81ad6a64be89922')
  }
```

- Insert at once multiple student details



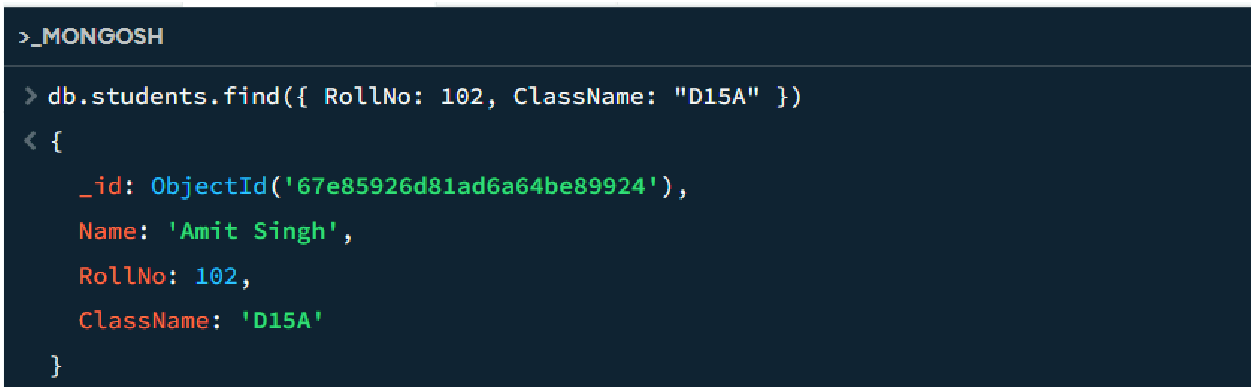
```
>_MONGOSH
> db.students.insertMany([
      { Name: "Ramesh Patel", RollNo: 101, ClassName: "D15A" },
      { Name: "Amit Singh", RollNo: 102, ClassName: "D15A" },
      { Name: "Priya Desai", RollNo: 103, ClassName: "D15B" },
      { Name: "Kiran Mehta", RollNo: 104, ClassName: "D15B" },
      { Name: "Sanjay Patil", RollNo: 105, ClassName: "D15A" },
      { Name: "Neha Verma", RollNo: 106, ClassName: "D15C" },
      { Name: "Rohit Sharma", RollNo: 107, ClassName: "D15B" },
      { Name: "Anjali Rao", RollNo: 108, ClassName: "D15C" },
      { Name: "Vikas Joshi", RollNo: 109, ClassName: "D15A" },
      { Name: "Meera Iyer", RollNo: 110, ClassName: "D15C" }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('67e85926d81ad6a64be89923'),
      '1': ObjectId('67e85926d81ad6a64be89924'),
      '2': ObjectId('67e85926d81ad6a64be89925'),
      '3': ObjectId('67e85926d81ad6a64be89926'),
      '4': ObjectId('67e85926d81ad6a64be89927'),
      '5': ObjectId('67e85926d81ad6a64be89928'),
      '6': ObjectId('67e85926d81ad6a64be89929'),
      '7': ObjectId('67e85926d81ad6a64be8992a'),
      '8': ObjectId('67e85926d81ad6a64be8992b'),
      '9': ObjectId('67e85926d81ad6a64be8992c')
    }
  }
```

- Display student for a particular class



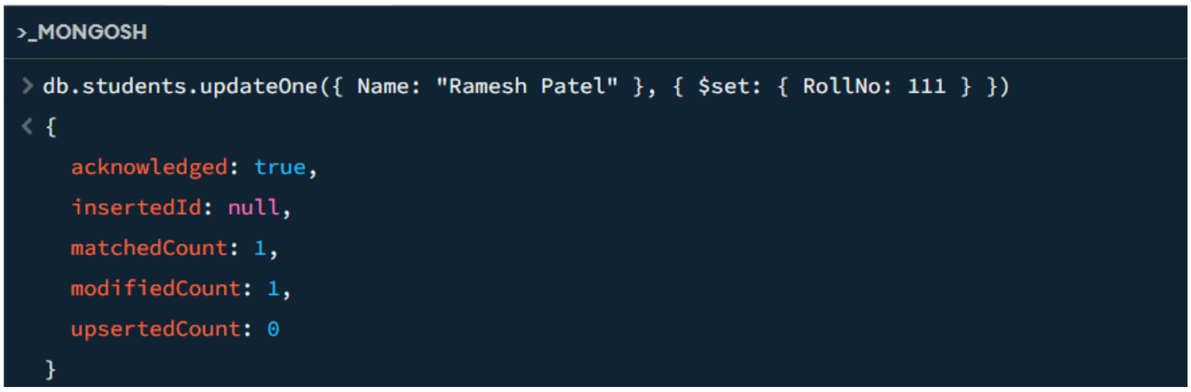
```
>_MONGOSH
> db.students.find({ ClassName: "D15A" })
< {
    _id: ObjectId('67e858d2d81ad6a64be89922'),
    Name: 'Rahul Sharma',
    RollNo: 101,
    ClassName: 'D15A'
  }
  {
    _id: ObjectId('67e85926d81ad6a64be89923'),
    Name: 'Ramesh Patel',
    RollNo: 101,
    ClassName: 'D15A'
  }
  {
    _id: ObjectId('67e85926d81ad6a64be89924'),
    Name: 'Amit Singh',
    RollNo: 102,
    ClassName: 'D15A'
  }
  {
    _id: ObjectId('67e85926d81ad6a64be89927'),
    Name: 'Sanjay Patil',
    RollNo: 105,
    ClassName: 'D15A'
  }
  {
    _id: ObjectId('67e85926d81ad6a64be8992b'),
    Name: 'Vikas Joshi',
    RollNo: 109,
```

- Display students of specific roll no in a class



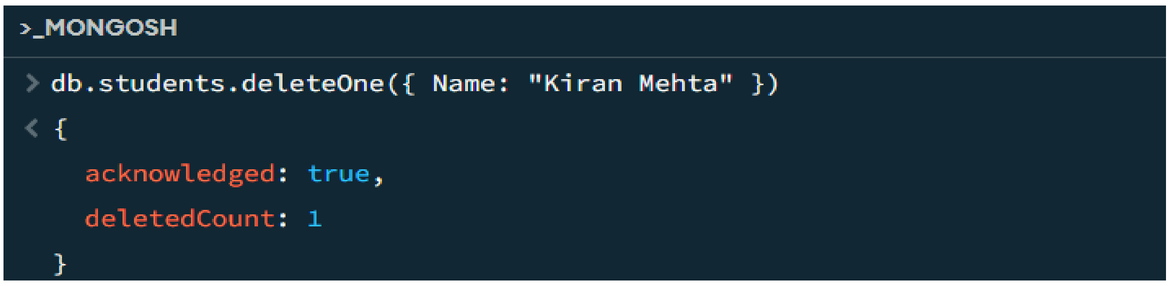
```
>_MONGOSH
> db.students.find({ RollNo: 102, ClassName: "D15A" })
< {
    _id: ObjectId('67e85926d81ad6a64be89924'),
    Name: 'Amit Singh',
    RollNo: 102,
    ClassName: 'D15A'
  }
```

- Change the roll no of a student



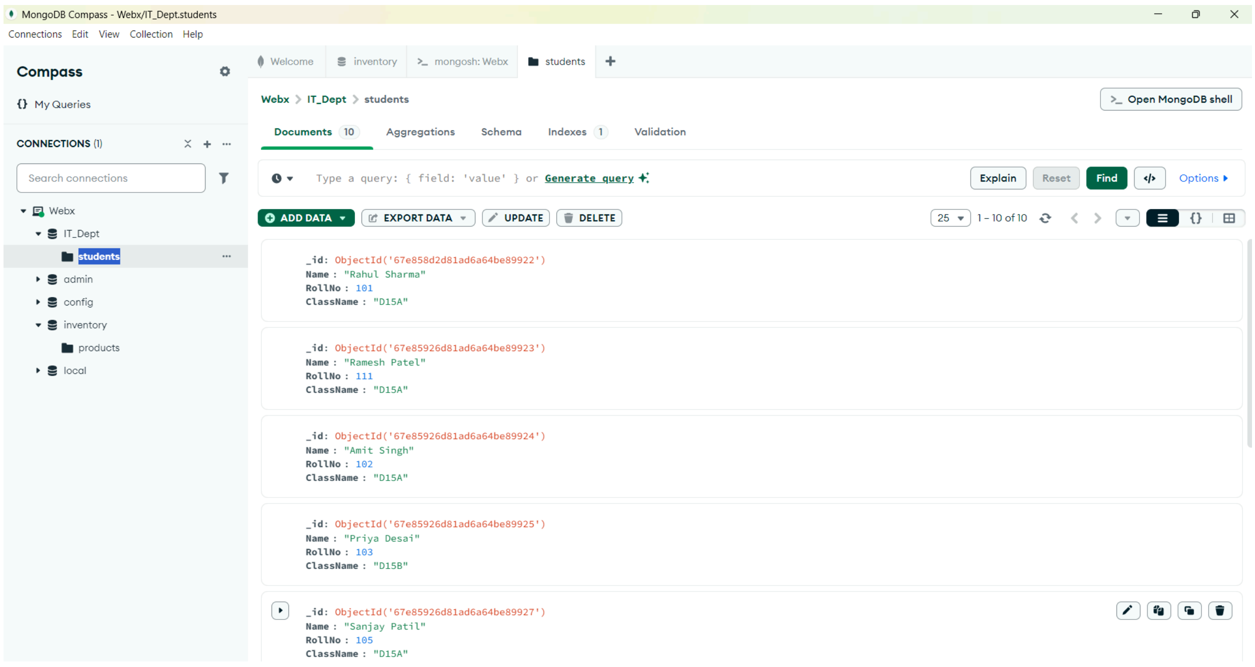
```
>_MONGOSH
> db.students.updateOne({ Name: "Ramesh Patel" }, { $set: { RollNo: 111 } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

- Delete entries of particular student



```
>_MONGOSH
> db.students.deleteOne({ Name: "Kiran Mehta" })
< {
    acknowledged: true,
    deletedCount: 1
  }
```

- Final data in the database



```
_id: ObjectId('67e858d2d81ad6a64be89922')
Name : "Rahul Sharma"
RollNo : 101
ClassName : "D15A"

_id: ObjectId('67e85926d81ad6a64be89923')
Name : "Ramesh Patel"
RollNo : 111
ClassName : "D15A"

_id: ObjectId('67e85926d81ad6a64be89924')
Name : "Amit Singh"
RollNo : 102
ClassName : "D15A"

_id: ObjectId('67e85926d81ad6a64be89925')
Name : "Priya Desai"
RollNo : 103
ClassName : "D15B"

_id: ObjectId('67e85926d81ad6a64be89927')
Name : "Sanjay Patil"
RollNo : 105
ClassName : "D15A"
```

- 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.

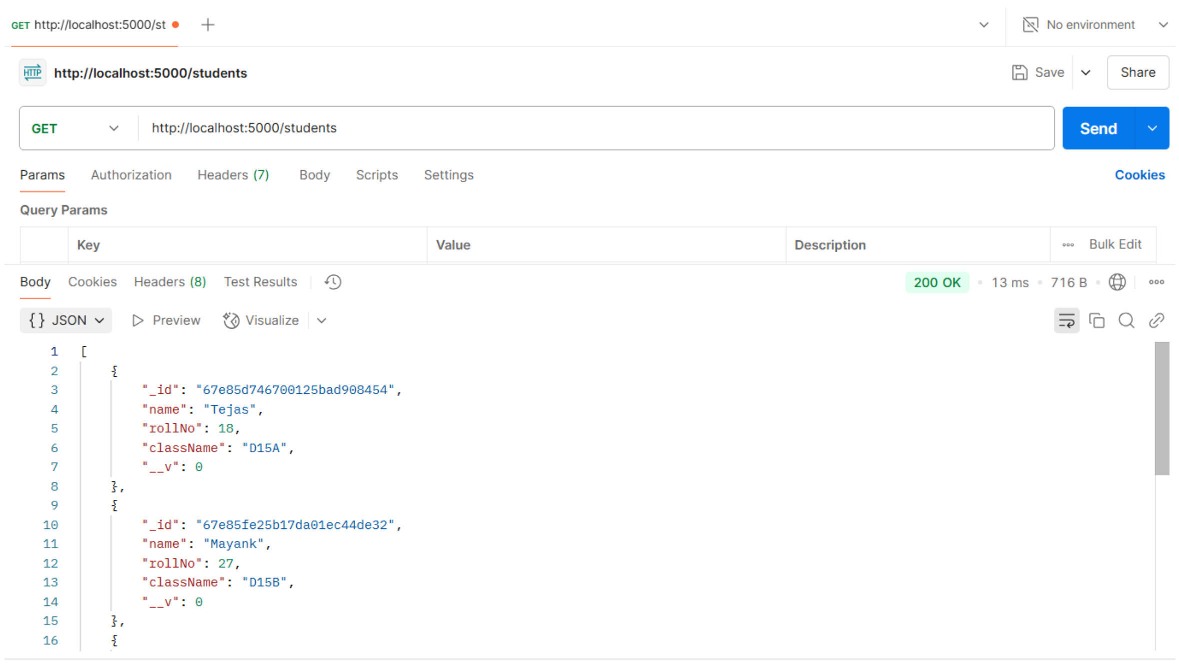
Server.js

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

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

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/studentDB", {
   useNewUrlParser: true,
   useUnifiedTopology: true
}).then(() => console.log("MongoDB Connected"))
  .catch(err => console.log(err));

// Define Student Schema
const studentSchema = new mongoose.Schema({
   name: String,
   rollNo: Number,
   className: String
});

const Student = mongoose.model("Student", studentSchema);

app.get("/students", async (req, res) => {
   const students = await Student.find();
   res.json(students);
});

app.get("/students/:id", async (req, res) => {
   const student = await Student.findById(req.params.id);
   res.json(student);
});

// Add a New Student
app.post("/students", async (req, res) => {
   const { name, rollNo, className } = req.body;
   const newStudent = new Student({ name, rollNo, className });
   await newStudent.save();
   res.json(newStudent);
});

app.put("/students/:id", async (req, res) => {
   const { name, rollNo, className } = req.body;

const updatedStudent = await Student.findByIdAndUpdate(
      req.params.id,
      { name, rollNo, className },
      { new: true }
   );
   res.json(updatedStudent);
});

app.delete("/students/:id", async (req, res) => {
   await Student.findByIdAndDelete(req.params.id);
   res.json({ message: "Student deleted" });
});
```

```
// Start Server
app.listen(5000, () => console.log("Server running on port 5000"));
```

OUTPUT : -

- Retrieve a list of all students.



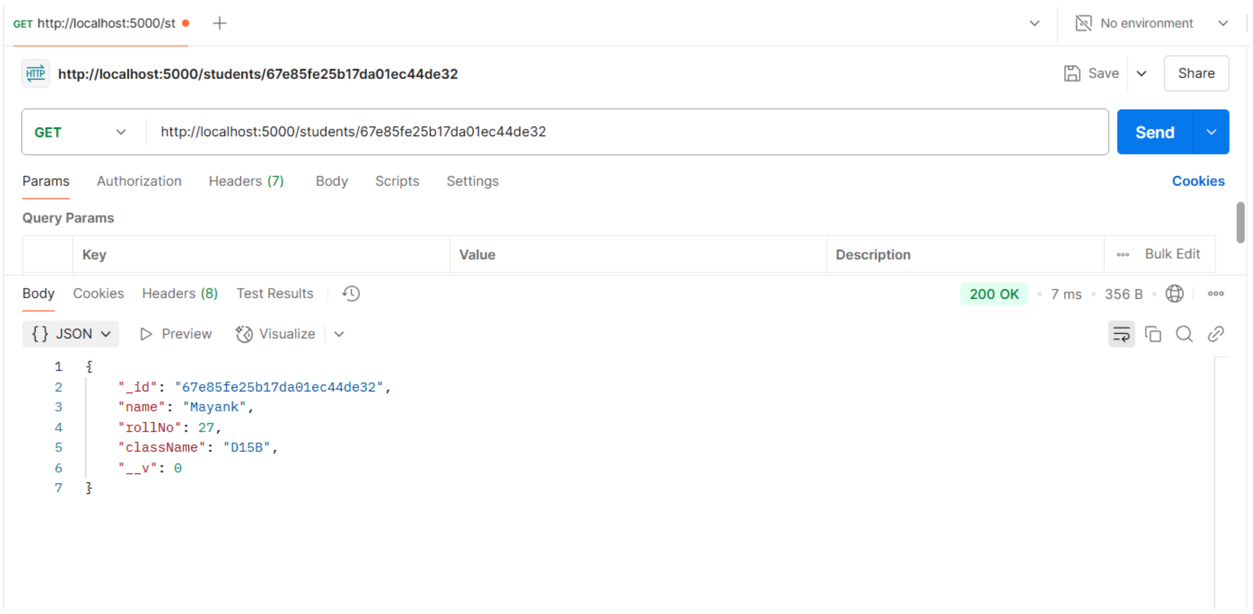
http://localhost:5000/students

GET http://localhost:5000/students

200 OK • 13 ms • 716 B

```
[
    {
        "_id": "67e85d746700125bad908454",
        "name": "Tejas",
        "rollNo": 18,
        "className": "D15A",
        "__v": 0
    },
    {
        "_id": "67e85fe25b17da01ec44de32",
        "name": "Mayank",
        "rollNo": 27,
        "className": "D15B",
        "__v": 0
    },
    {
```

- Retrieve details of an individual student by ID.



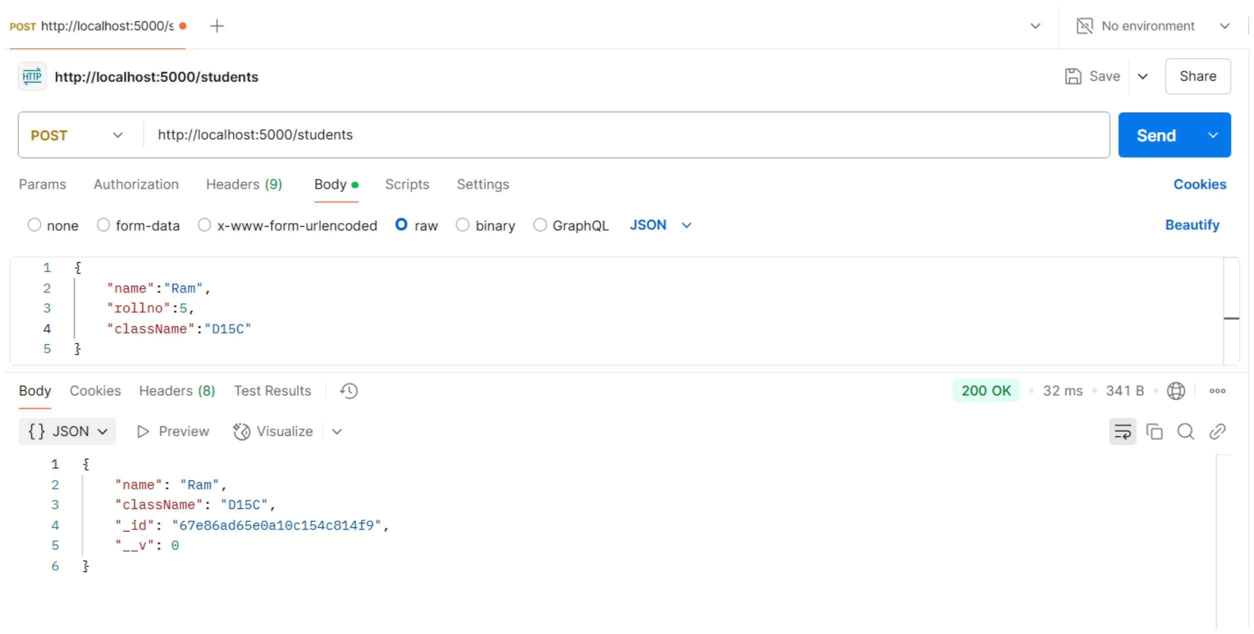
http://localhost:5000/students/67e85fe25b17da01ec44de32

GET http://localhost:5000/students/67e85fe25b17da01ec44de32

200 OK • 7 ms • 356 B

```
{
    "_id": "67e85fe25b17da01ec44de32",
    "name": "Mayank",
    "rollNo": 27,
    "className": "D15B",
    "__v": 0
}
```

- Add a new student to the database.



POST http://localhost:5000/students

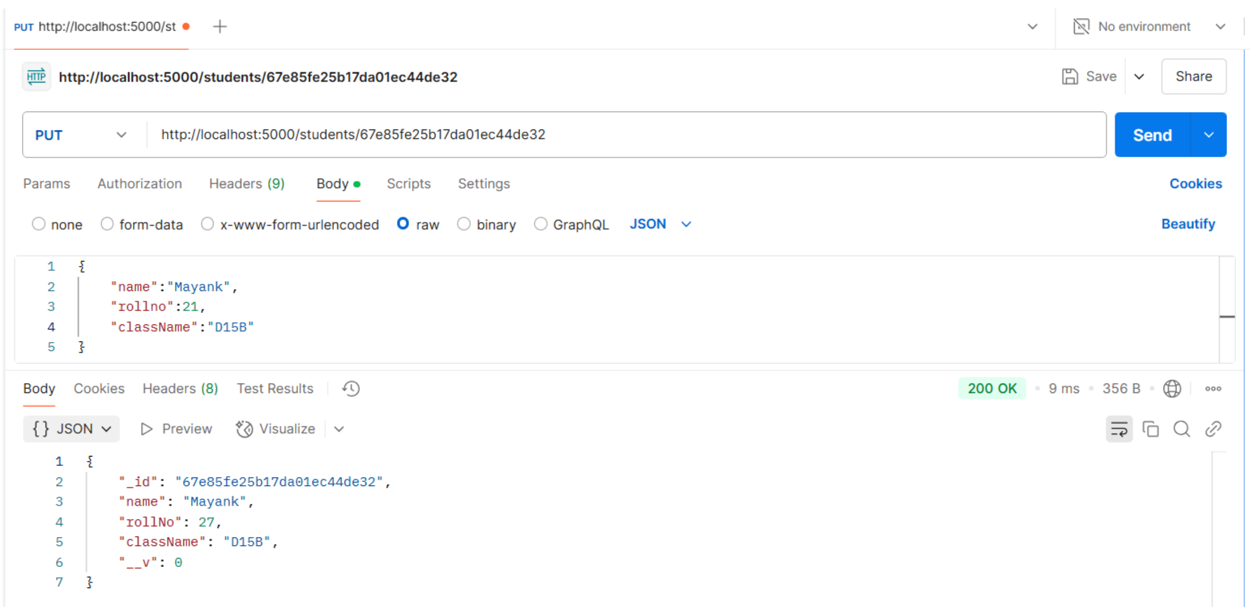
Body (raw, JSON):

```
{
    "name":"Ram",
    "rollno":5,
    "className":"D15C"
}
```

Response: 200 OK · 32 ms · 341 B

```
{
    "name": "Ram",
    "className": "D15C",
    "_id": "67e86ad65e0a10c154c814f9",
    "__v": 0
}
```

- Update details of an existing student by ID



PUT http://localhost:5000/students/67e85fe25b17da01ec44de32

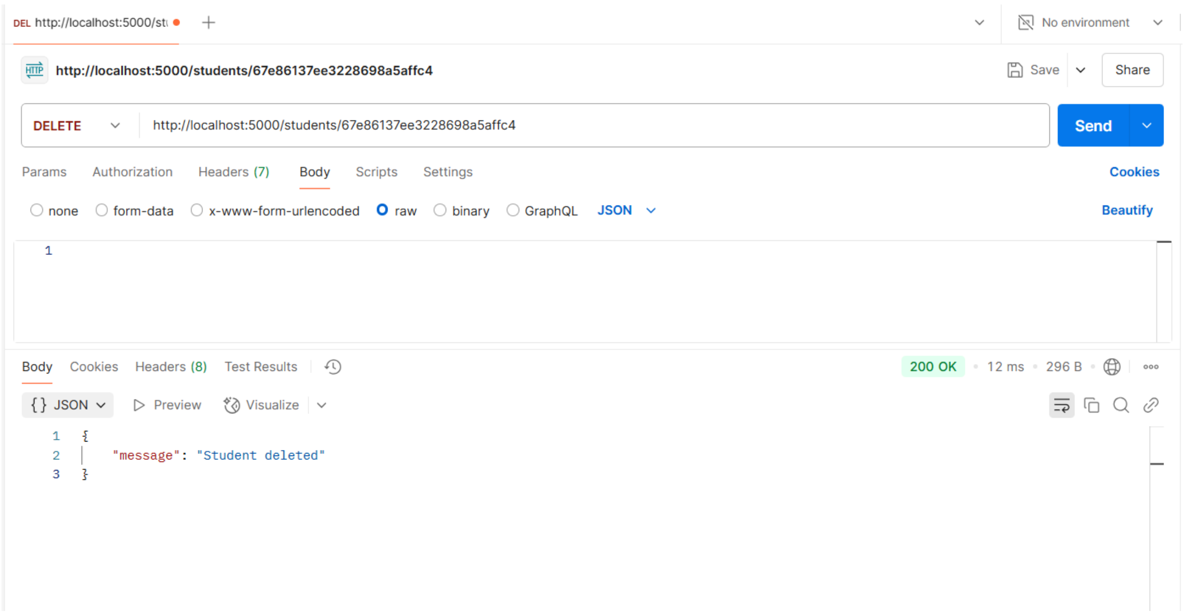
Body (raw, JSON):

```
{
    "name":"Mayank",
    "rollno":21,
    "className":"D15B"
}
```

Response: 200 OK · 9 ms · 356 B

```
{
    "_id": "67e85fe25b17da01ec44de32",
    "name": "Mayank",
    "rollNo": 27,
    "className": "D15B",
    "__v": 0
}
```

- Delete a student from the database by ID.


DEL http://localhost:5000/st

No environment

http://localhost:5000/students/67e86137ee3228698a5affc4

DELETE http://localhost:5000/students/67e86137ee3228698a5affc4

200 OK • 12 ms • 296 B

```
{
    "message": "Student deleted"
}
```

Conclusion : -

This practical demonstrates CRUD operations in MongoDB and their implementation using Node.js, Express, and Mongoose. It covers creating a student database, performing data insertion, retrieval, updating, and deletion. The RESTful API enables seamless interaction with MongoDB, supporting operations like adding, retrieving, updating, and deleting student records. This exercise highlights efficient data management and server-side processing in modern applications