

## **EXPERIMENT NO. 7 - MongoDB**

<b>Name of Student</b>	<b>Sneha Patra</b>
<b>Class Roll No</b>	<b>D15A_40</b>
<b>D.O.P.</b>	<b><u>13/03/2025</u></b>
<b>D.O.S.</b>	<b><u>20/03/2025</u></b>
<b>Sign and Grade</b>	

**Aim:** To study CRUD operations in MongoDB

### **Problem Statement:**

1. Create a new database to storage student details of IT dept( Name, Roll no, class name) and perform the following on the database
  - Insert one student details
  - Insert at once multiple student details
  - Display student for a particular class
  - Display students of specific roll no in a class
  - Change the roll no of a student
  - Delete entries of particular student
2. 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.

**GitHub Link:** [https://github.com/Sneha0321/WebX\\_Exp\\_7](https://github.com/Sneha0321/WebX_Exp_7)

### **Code:**

[student.js](#)

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

const studentSchema = new mongoose.Schema({
  name: String,
  age : Number,
  grade: String,
});

const Student = mongoose.model('Student', studentSchema);
module.exports = Student;
```

## server.js

```
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const studentRoutes = require('./routes/studentRoutes');

const app = express();
app.use(express.json());
app.use('/students', studentRoutes);

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('MongoDB connected'))
  .catch((err) => console.error(err));

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

## studentRoutes.js

```
const express = require('express');
const Student = require('./models/student');
const router = express.Router();

// Get all students
router.get('/', async (req, res) => {
  try {
    const students = await Student.find();
    res.json(students);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// Get student by ID
router.get('/:id', async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json(student);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// Get student by rollNo
router.get('/age/:age', async (req, res) => {
  try {
    const student = await Student.findOne({ age: req.params.age });
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
  }
});
```

```

    }
    res.json(student);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

```

// Create a new student

```

router.post('/', async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.status(201).json(student);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});

```

// Update a student by rollNo

```

router.put('/age/:age', async (req, res) => {
  try {
    const student = await Student.findOneAndUpdate(
      { age: req.params.age },
      req.body,
      { new: true }
    );
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json(student);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

```

// Delete a student by rollNo

```

router.delete('/age/:age', async (req, res) => {
  try {
    const student = await Student.findOneAndDelete({ age: req.params.age });
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json({ message: 'Student deleted' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

```

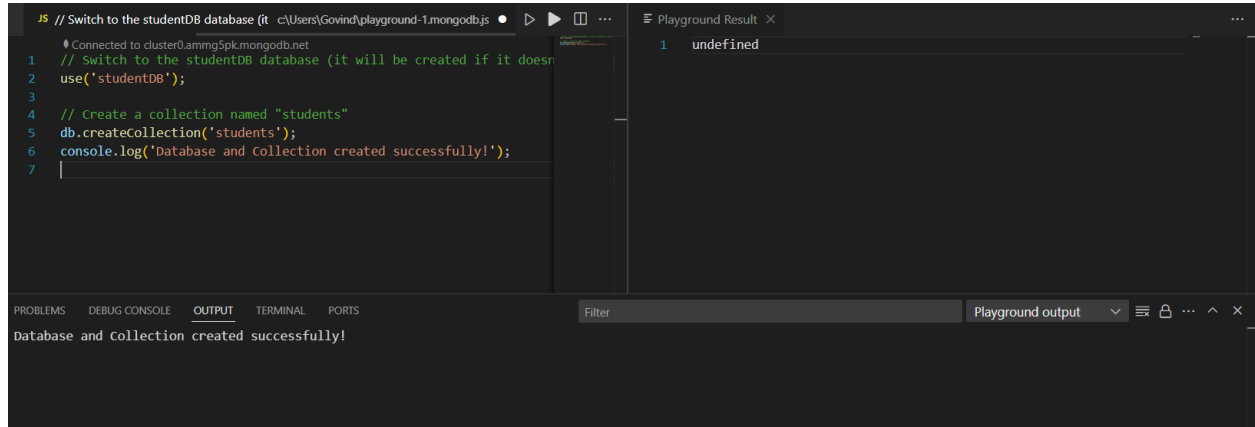
```

module.exports = router;

```

## Output:

1. Create a new database to storage student details of IT dept( Name, Roll no, class name) and perform the following on the database



The screenshot displays the MongoDB Playground interface. The left pane shows the following JavaScript code:

```
JS // Switch to the studentDB database (it c:\Users\Govind\playground-1\mongodb.js
1 // Switch to the studentDB database (it will be created if it doesn't exist)
2 use('studentDB');
3
4 // Create a collection named "students"
5 db.createcollection('students');
6 console.log('Database and Collection created successfully!');
7
```

The right pane, titled "Playground Result", shows the output of the execution:

```
1 undefined
```

The bottom status bar indicates the execution was successful, displaying the message: "Database and Collection created successfully!".

```
JS // Switch to the studentDB database (it c:\Users\Govind\playground-1.mongodb.js ● ▶ □ ...

5 db.createCollection('students');
6 console.log('Database and Collection created successfully!');
7 db.getCollection('students').insertOne({
8   name: 'John Doe',
9   rollNo: 101,
10  className: 'IT-1'
11 });
12 console.log('One student inserted');
13
```

Database and Collection created successfully!  
One student inserted

```
JS // Switch to the studentDB database (it c:\Users\Govind\playground-1.mongodb.js ● ▶ □ ...

12 console.log('One student inserted');
13 db.getCollection('students').insertMany([
14   { name: 'Alice', rollNo: 102, className: 'IT-1' },
15   { name: 'Bob', rollNo: 103, className: 'IT-2' },
16   { name: 'Charlie', rollNo: 104, className: 'IT-1' }
17 ]);
18 console.log('Multiple students inserted');
19
```

Database and Collection created successfully!  
One student inserted  
Multiple students inserted

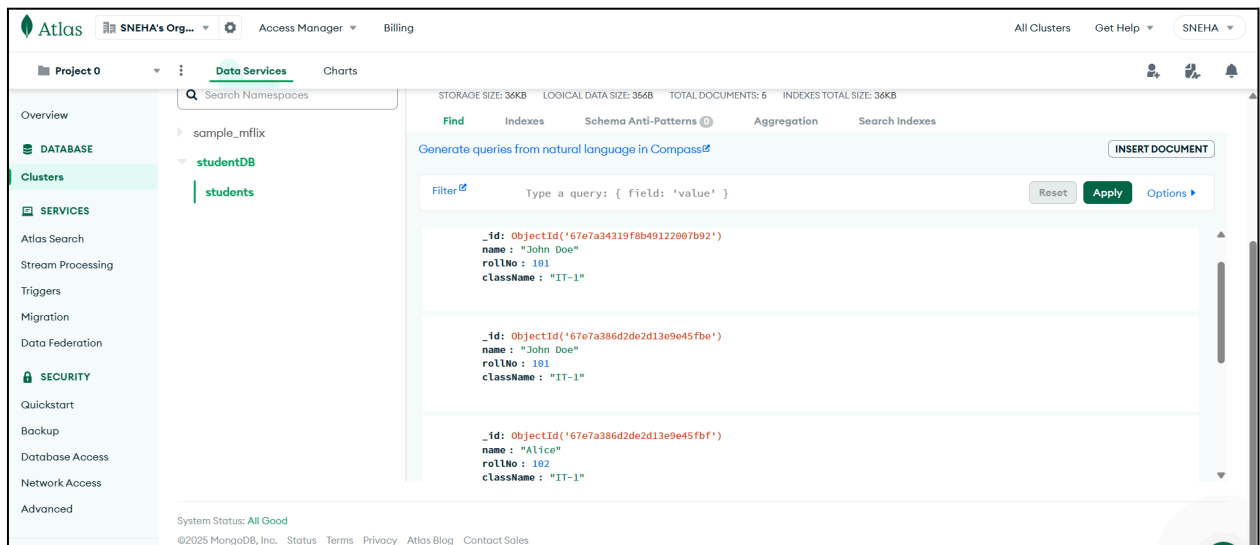
Playground Result X

1	undefined
---	-----------

- **Insert One Student**

```
{  
  
  "name": "John Doe",  
  
  "rollNo": 101,  
  
  "className": "IT-1"
```

```
}
```

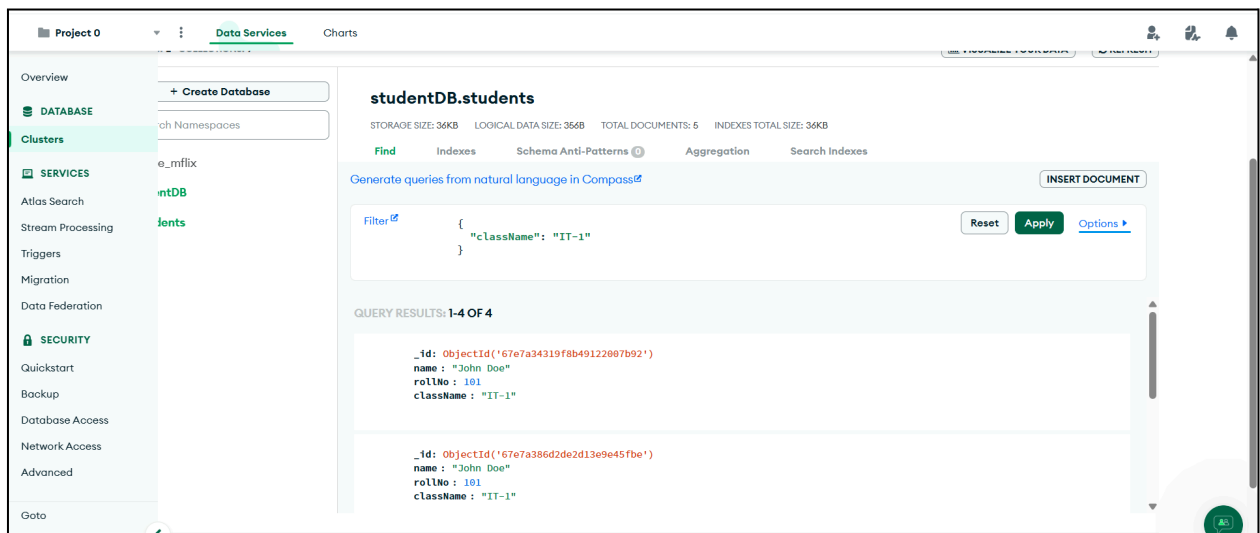


- Display Students for a Particular Class

```
{
```

```
"className": "IT-1"
```

```
}
```



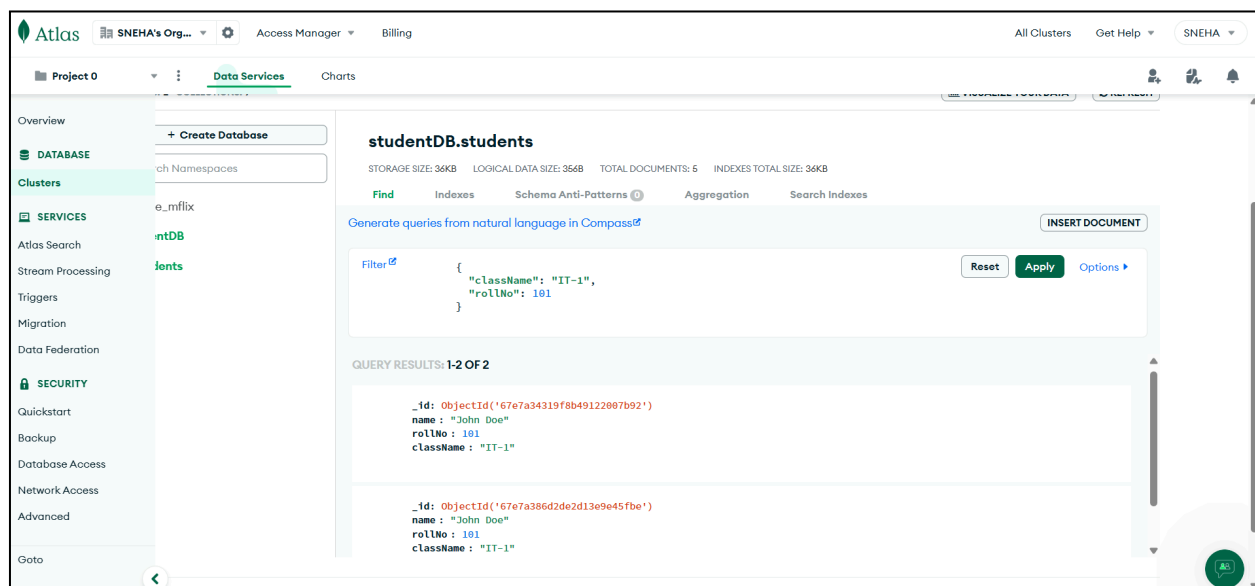
- Display Students of a Specific Roll Number in a Class

```
{
```

```
"className": "IT-1",
```

```
"rollNo": 101
```

```
}
```



- **Change the Roll Number of a Student**

Go to the Update tab.

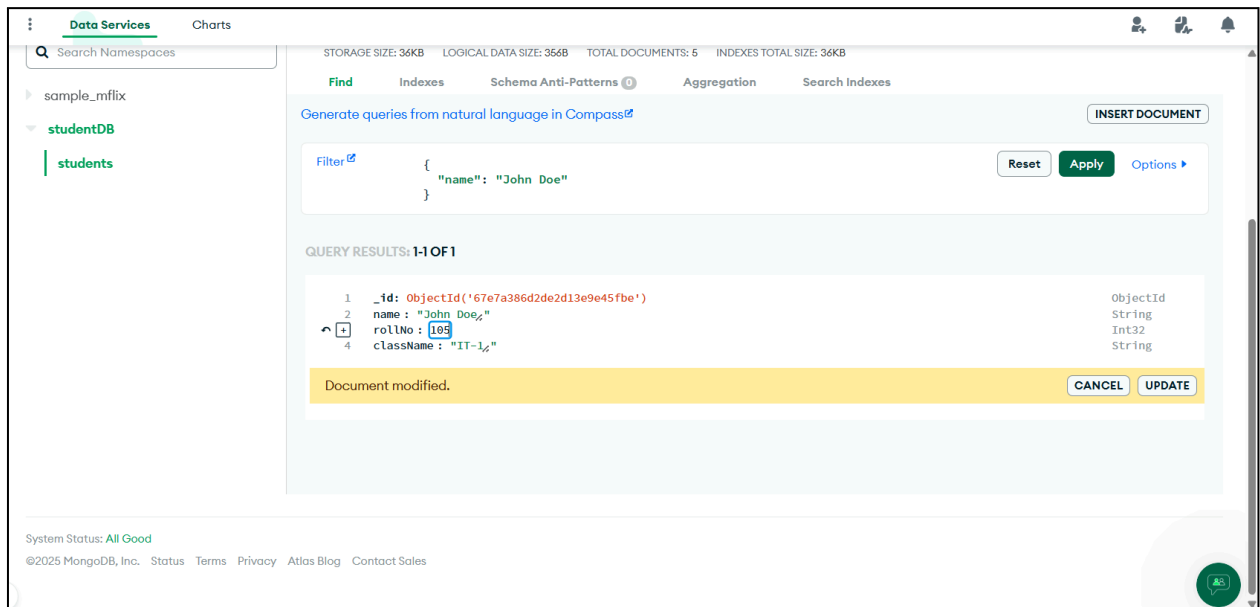
Use this filter to find the student:

```
{  
  
  "name": "John Doe"  
}
```

Use this update to change the roll number

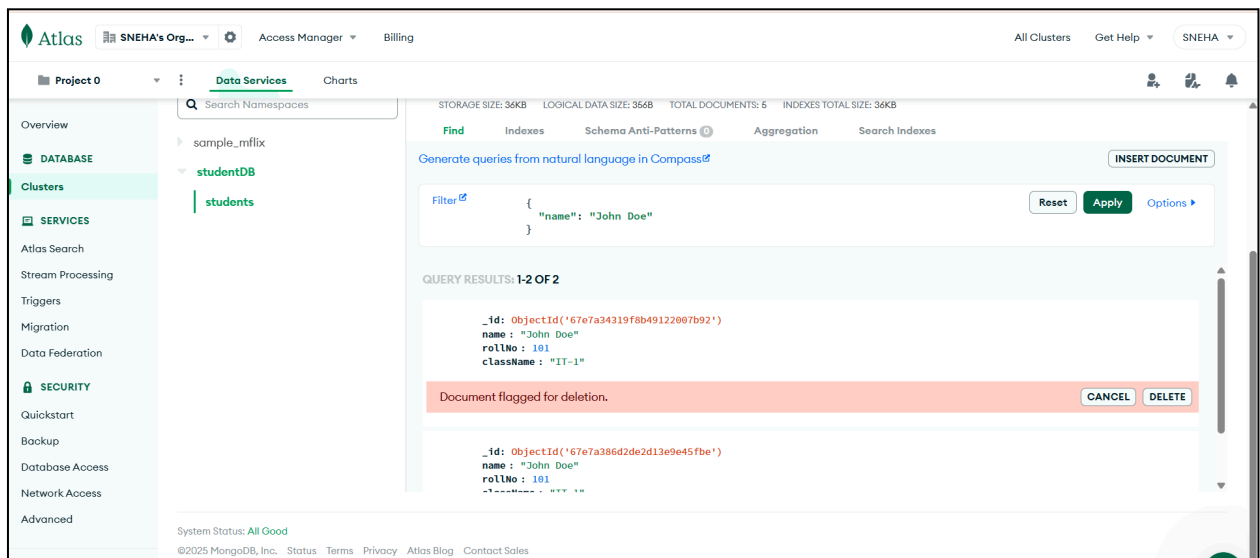
```
{  
  
  "$set": {  
  
    "rollNo": 105  
  
  }  
}
```

- **Click Update.**



## Delete Entries of a Particular Student

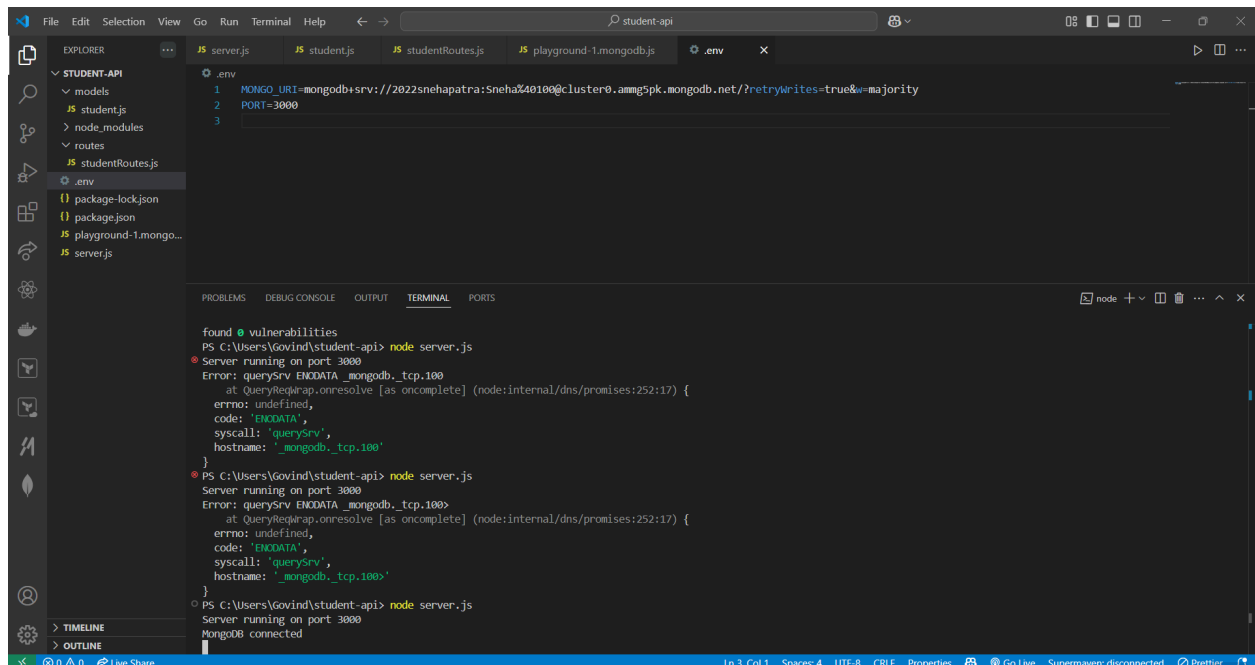
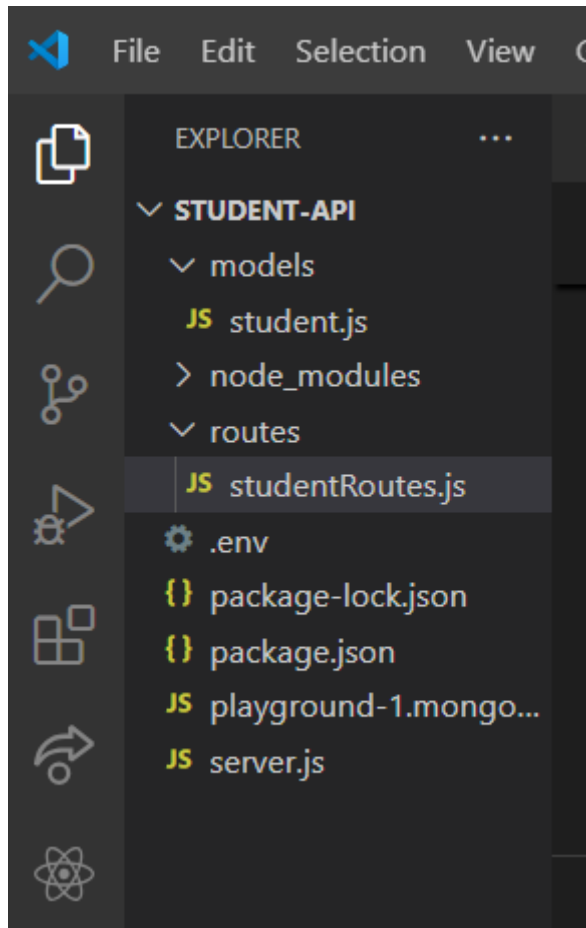
- Go to the Delete tab.
- Use this filter to delete a student:



2. Create a set of RESTful endpoints using Node.js, Express, and Mongoose for handling student data operations.

## Folder Structure:





## 1. Retrieve all students

GET **/students**

## 2. Retrieve details of a specific student by ID

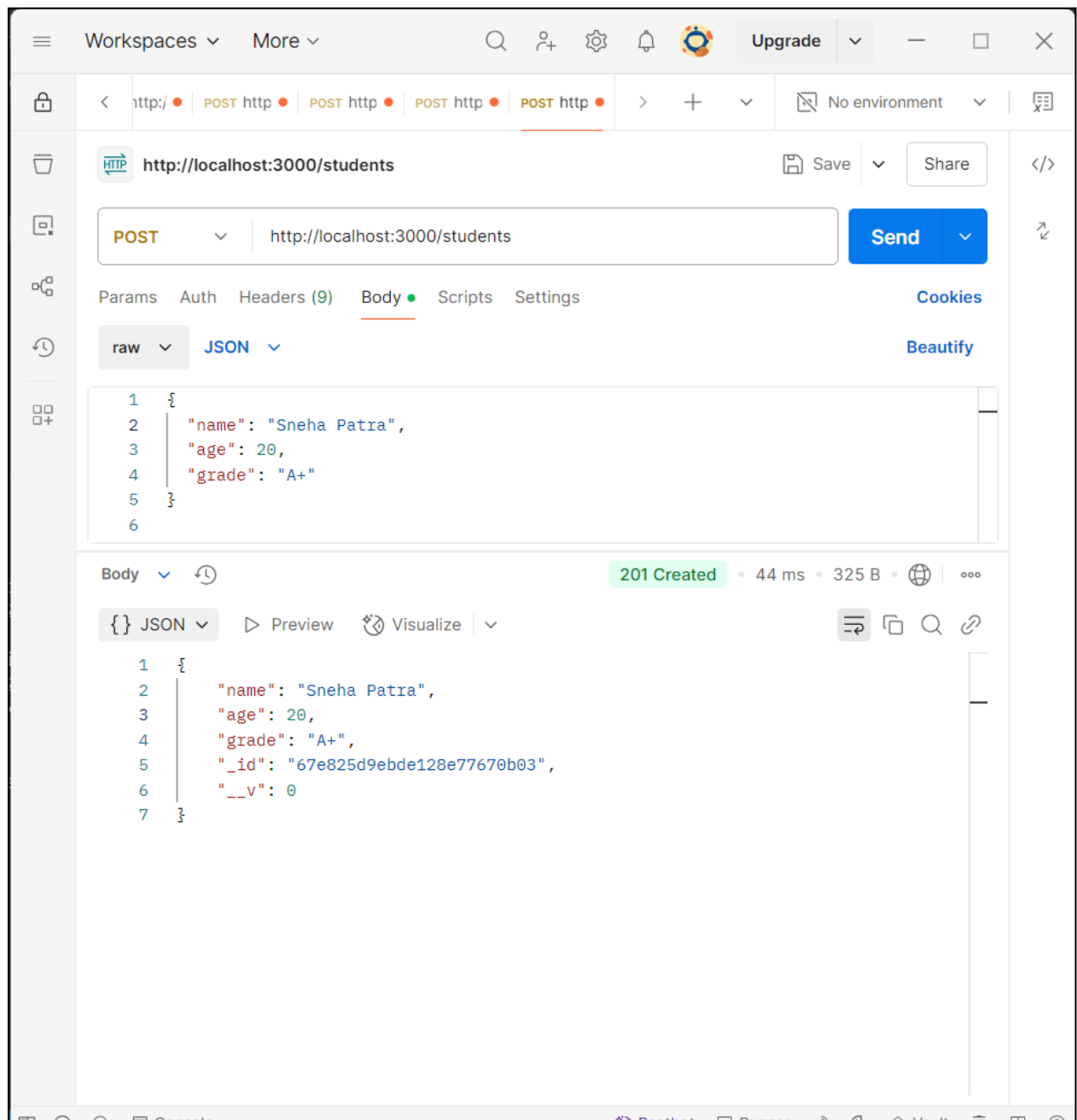
GET **/students/:id**

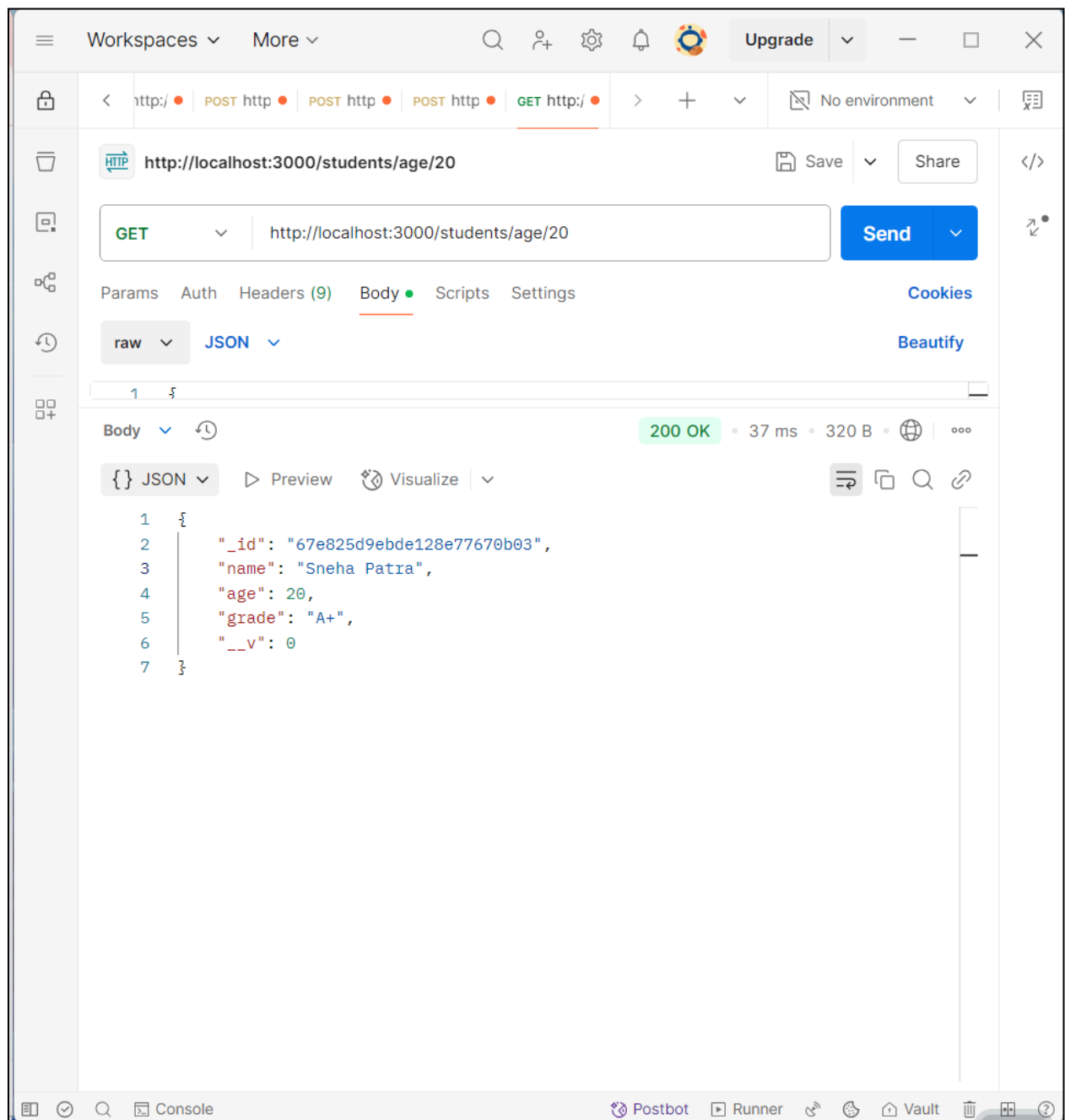
### 3. Add a new student

**POST /students**

**Request Body:**

```
{  
  
  "name": "Sneha Patra",  
  
  "age": 20,  
  
  "grade": "A+"  
}
```





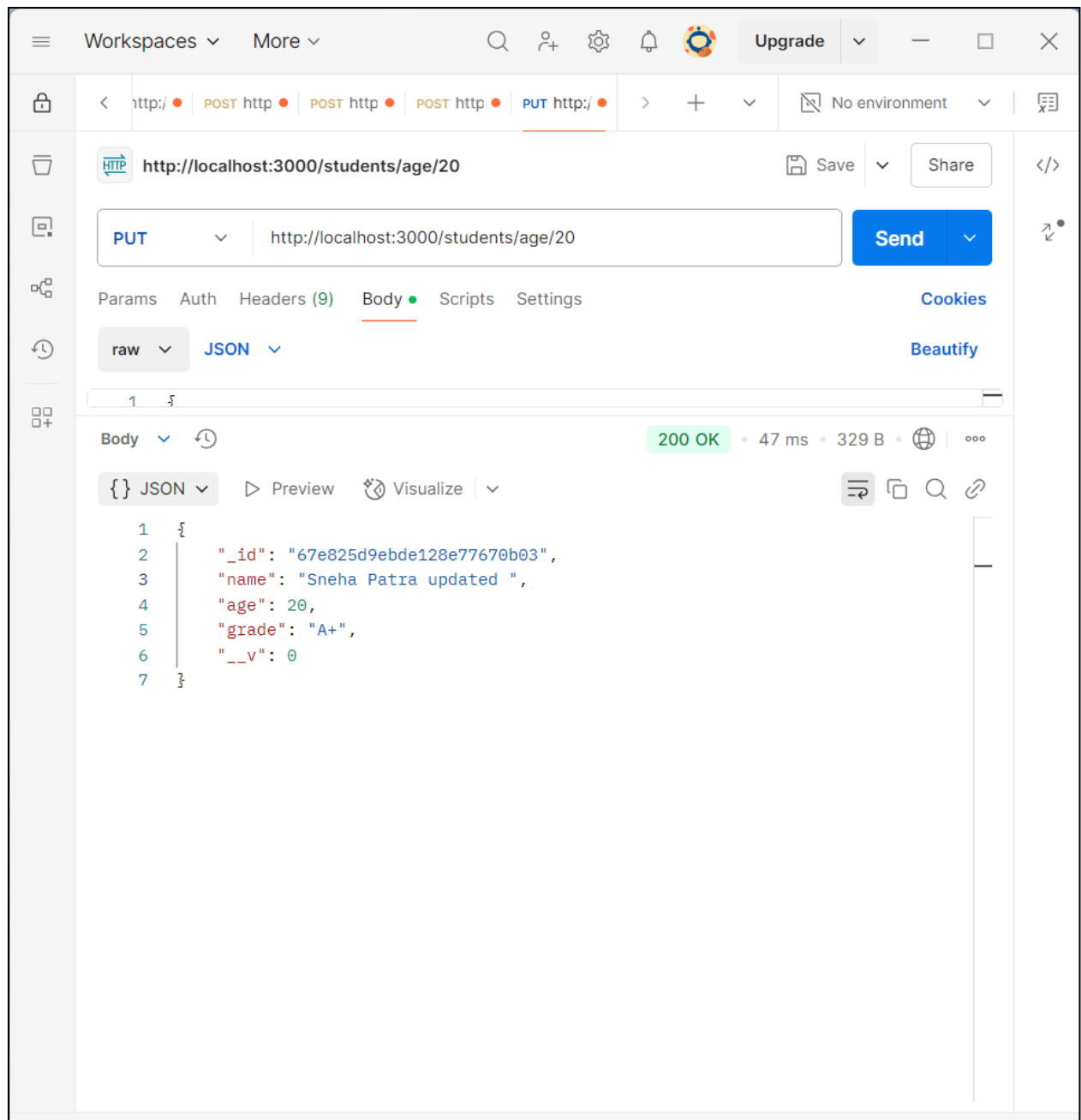
#### 4. Update student details by ID

**PUT /students/:id**

**Request Body:**

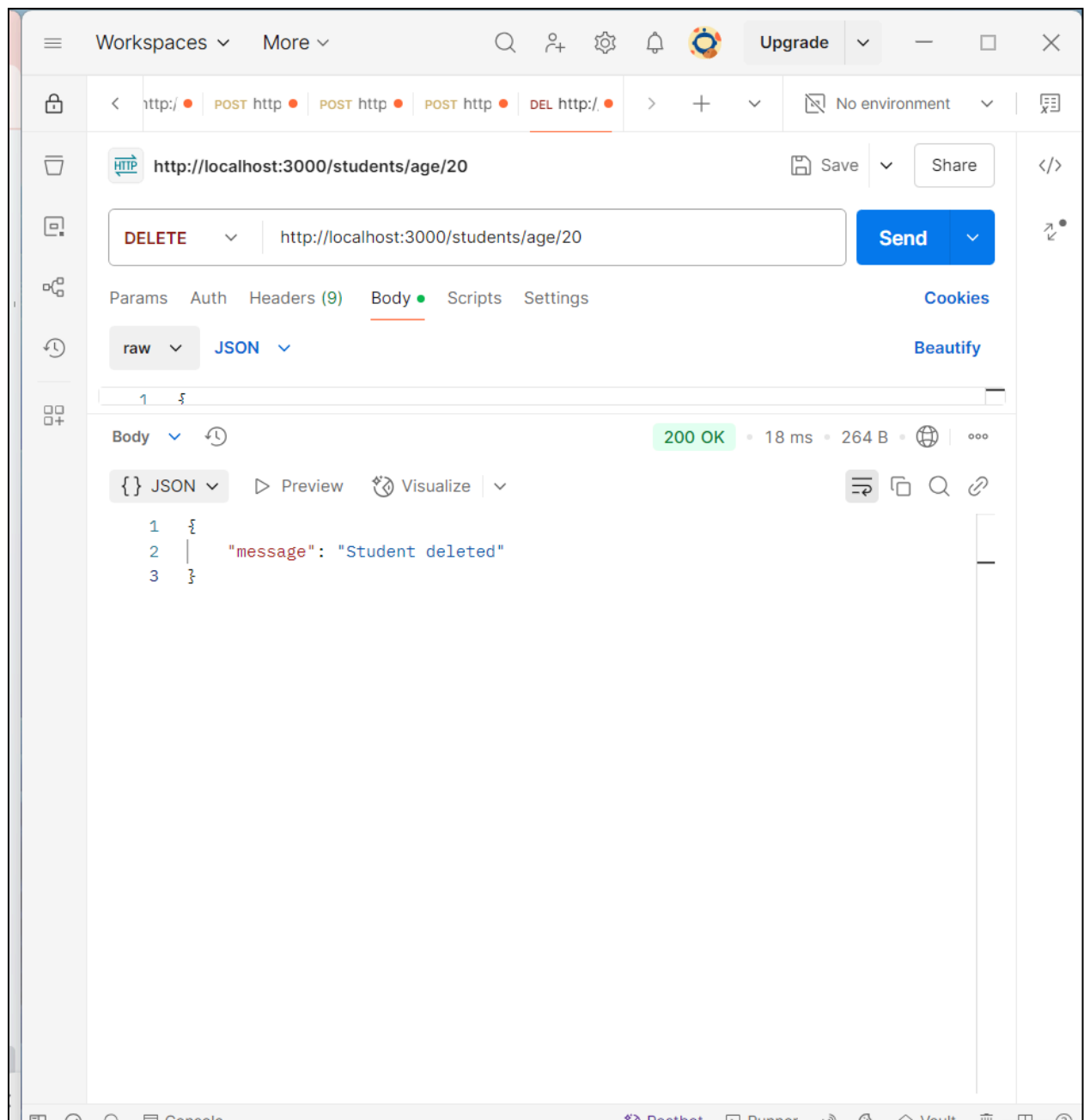
```
{  
  
  "name": "Sneha Patra updated ",  
  
  "age": 20 ,  
  
  "grade": "A+"  
}
```

}



## 5. Delete a student by ID

**DELETE** `/students/:id`



## Conclusion:

In this experiment, we successfully implemented CRUD operations in MongoDB to manage student data. We also developed RESTful APIs using Node.js, Express, and Mongoose to perform these operations efficiently. This experiment demonstrated the practical application of MongoDB in managing databases and how RESTful APIs interact with the database for seamless data retrieval and modification.