EXPERIMENT NO. 7 - MongoDB

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

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

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

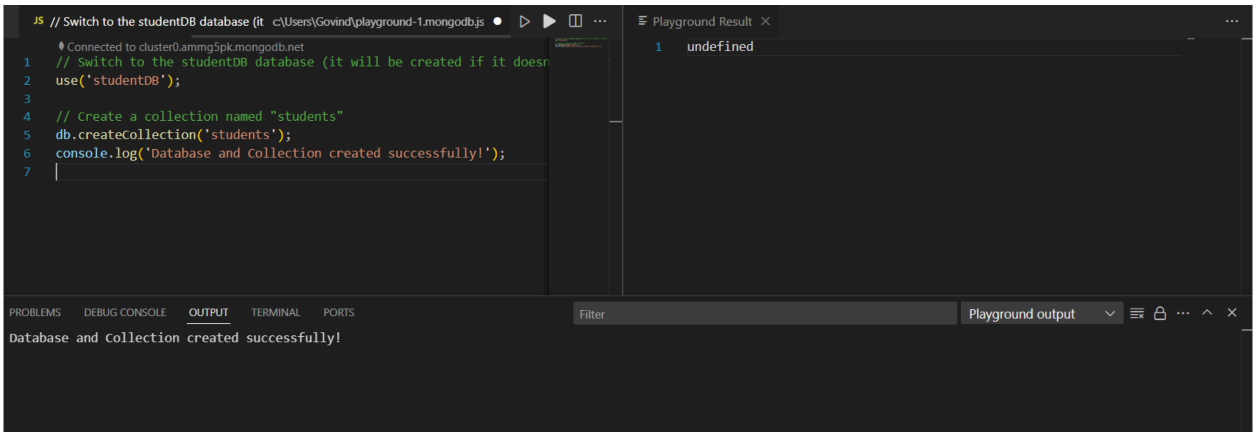
```
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const studentRoutes = require('./routes/studentRoutes');

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:

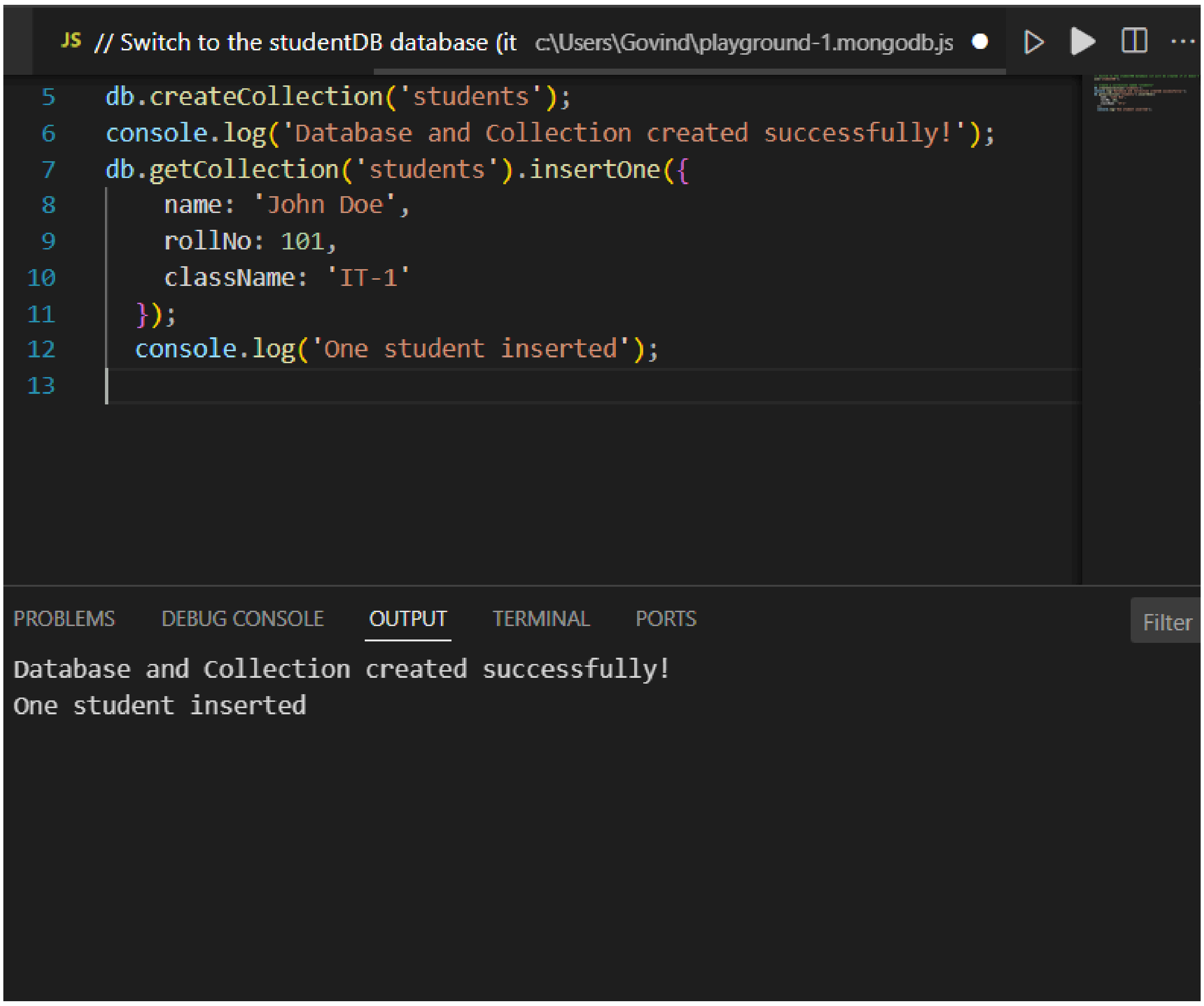
1. Create a new database to storage student details of IT dept(Name, Roll no, class name) and perform the following on the database


// Switch to the studentDB database (it c:\Users\Govind\playground-1.mongodb.js
Connected to cluster0.ammg5pk.mongodb.net
Playground Result
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
PORTS
Filter
Playground output

```javascript
// Switch to the studentDB database (it will be created if it doesn
use('studentDB');

// Create a collection named "students"
db.createCollection('students');
console.log('Database and Collection created successfully!');
```

```
undefined
```

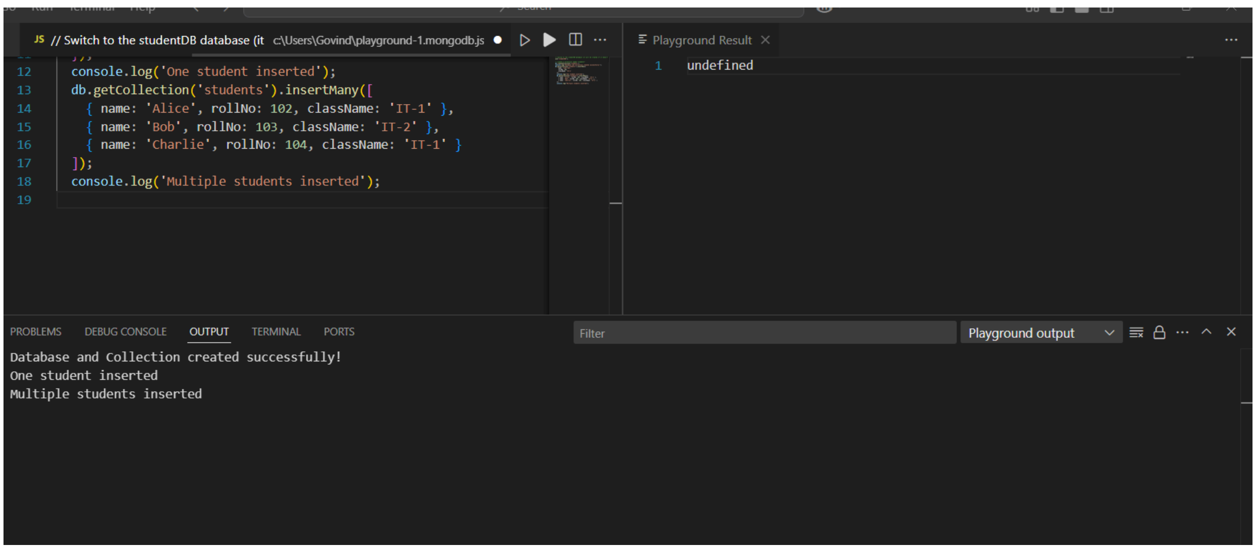
```
Database and Collection created successfully!
```



JS // Switch to the studentDB database (it c:\Users\Govind\playground-1.mongodb.js
PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
Filter

```
db.createCollection('students');
console.log('Database and Collection created successfully!');
db.getCollection('students').insertOne({
    name: 'John Doe',
    rollNo: 101,
    className: 'IT-1'
  });
  console.log('One student inserted');
```

```
Database and Collection created successfully!
One student inserted
```



JS // Switch to the studentDB database (it c:\Users\Govind\playground-1.mongodb.js
Playground Result
PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
Filter
Playground output

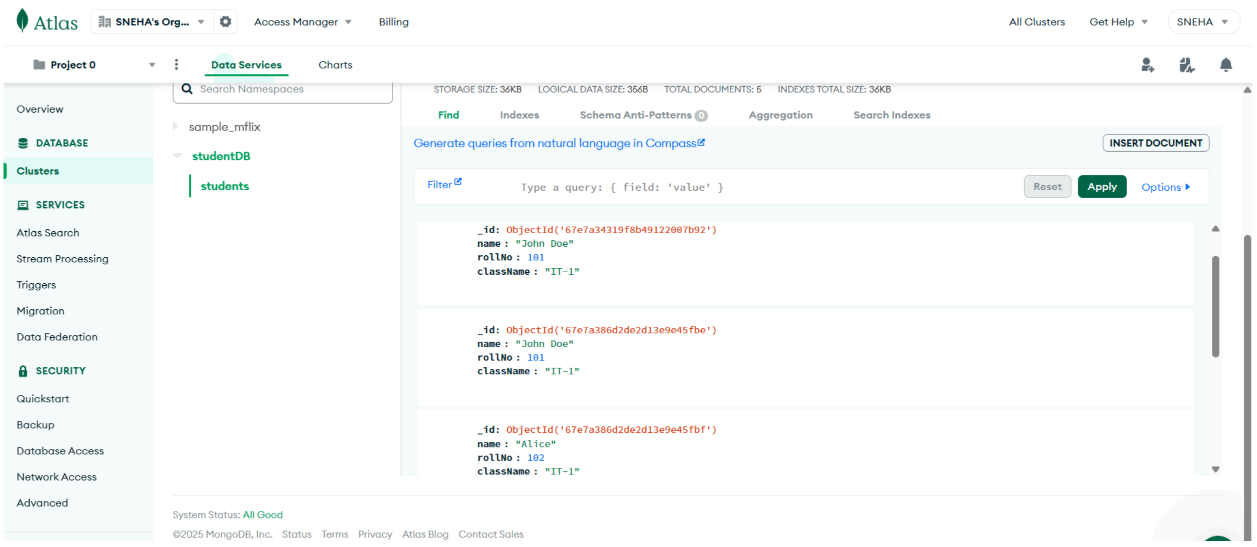
```
console.log('One student inserted');
db.getCollection('students').insertMany([
  { name: 'Alice', rollNo: 102, className: 'IT-1' },
  { name: 'Bob', rollNo: 103, className: 'IT-2' },
  { name: 'Charlie', rollNo: 104, className: 'IT-1' }
]);
console.log('Multiple students inserted');
```

```
undefined
```

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

- **Insert One Student**

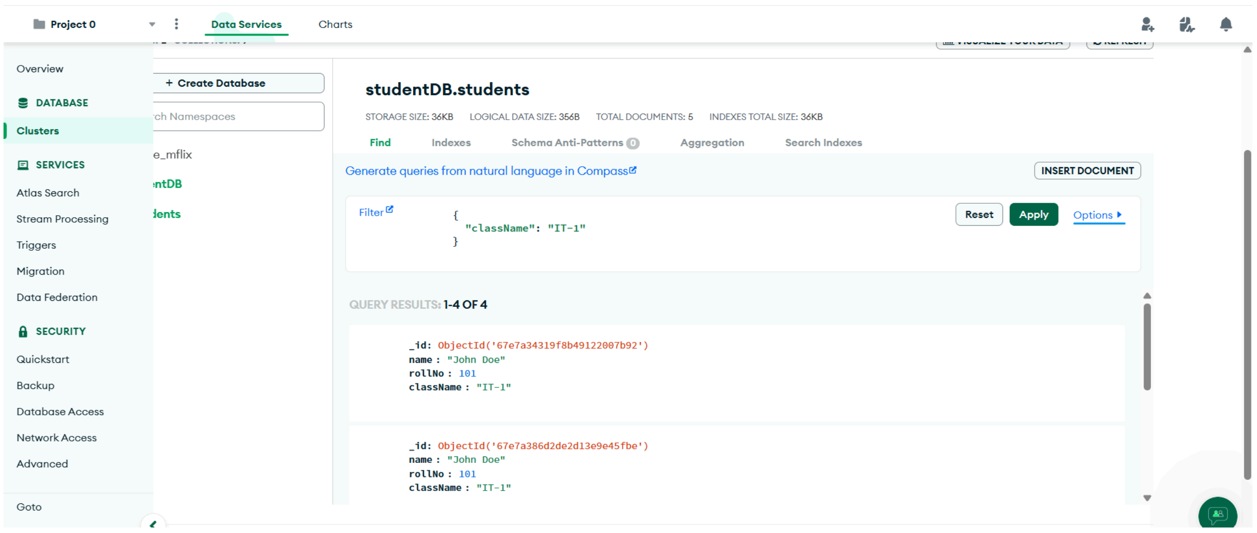
```
{
  "name": "John Doe",
  "rollNo": 101,
  "className": "IT-1"
}
```



```
_id: ObjectId('67e7a34319f8b49122007b92')
name : "John Doe"
rollNo : 101
className : "IT-1"

_id: ObjectId('67e7a386d2de2d13e9e45fbe')
name : "John Doe"
rollNo : 101
className : "IT-1"

_id: ObjectId('67e7a386d2de2d13e9e45fbf')
name : "Alice"
rollNo : 102
className : "IT-1"
```

- **Display Students for a Particular Class**

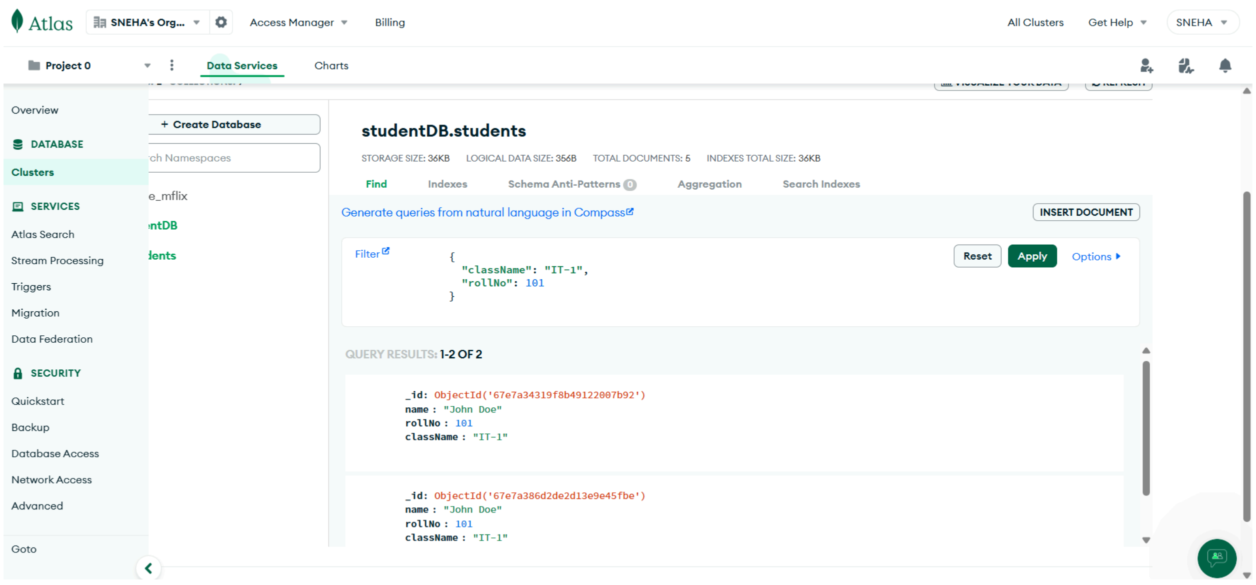
```
{
  "className": "IT-1"
}
```



```
{
  "className": "IT-1"
}

_id: ObjectId('67e7a34319f8b49122007b92')
name : "John Doe"
rollNo : 101
className : "IT-1"

_id: ObjectId('67e7a386d2de2d13e9e45fbe')
name : "John Doe"
rollNo : 101
className : "IT-1"
```

- **Display Students of a Specific Roll Number in a Class**

```
{
  "className": "IT-1",
  "rollNo": 101
}
```



studentDB.students

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 356B TOTAL DOCUMENTS: 5 INDEXES TOTAL SIZE: 36KB

```
{
  "className": "IT-1",
  "rollNo": 101
}
```

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('67e7a34319f8b49122007b92')
name : "John Doe"
rollNo : 101
className : "IT-1"

_id: ObjectId('67e7a386d2de2d13e9e45fbe')
name : "John Doe"
rollNo : 101
className : "IT-1"
```

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

  Go to the Update tab.

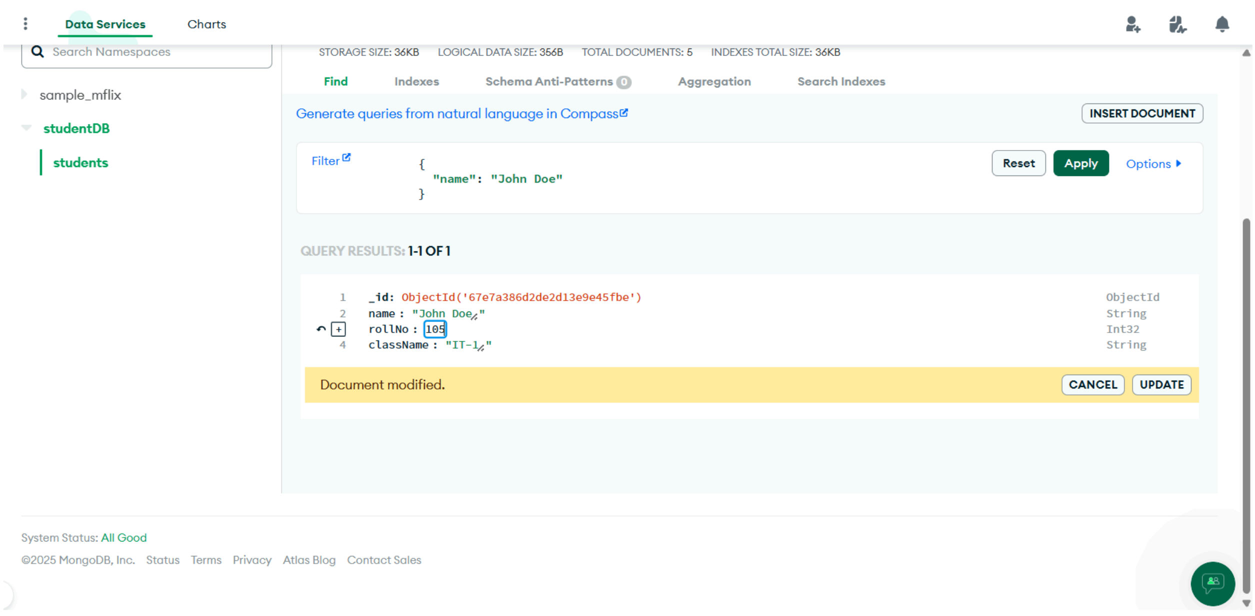
  Use this filter to find the student:

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

  Use this update to change the roll number

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

- **Click Update.**

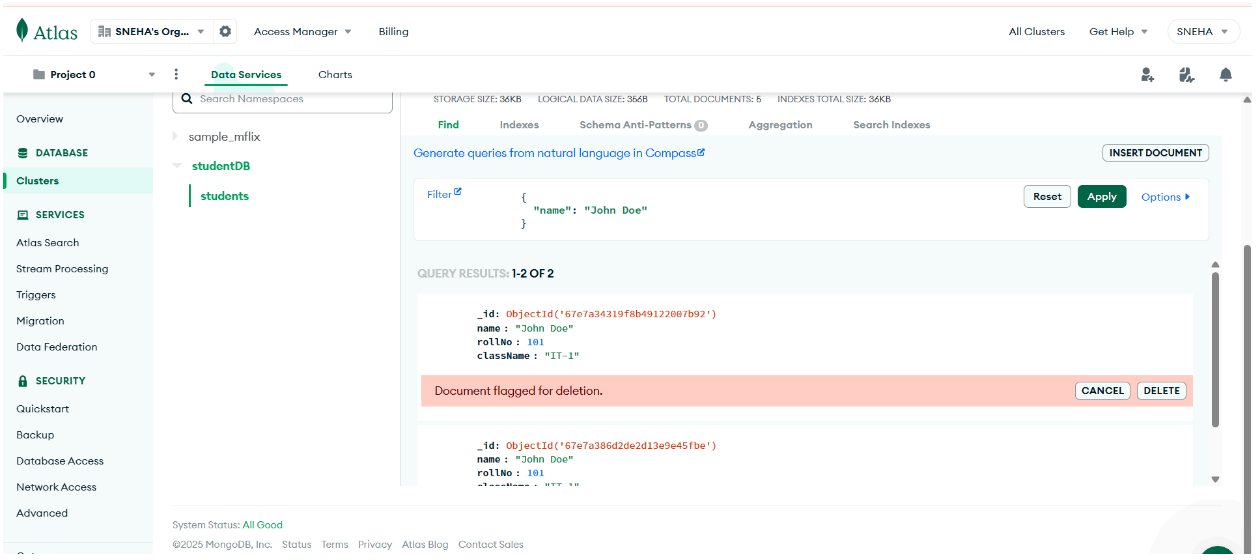


```
{
  "name": "John Doe"
}
```

```
_id: ObjectId('67e7a386d2de2d13e9e45fbe')      ObjectId
name : "John Doe"                              String
rollNo : 105                                   Int32
className : "IT-1"                             String
```

Delete Entries of a Particular Student

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

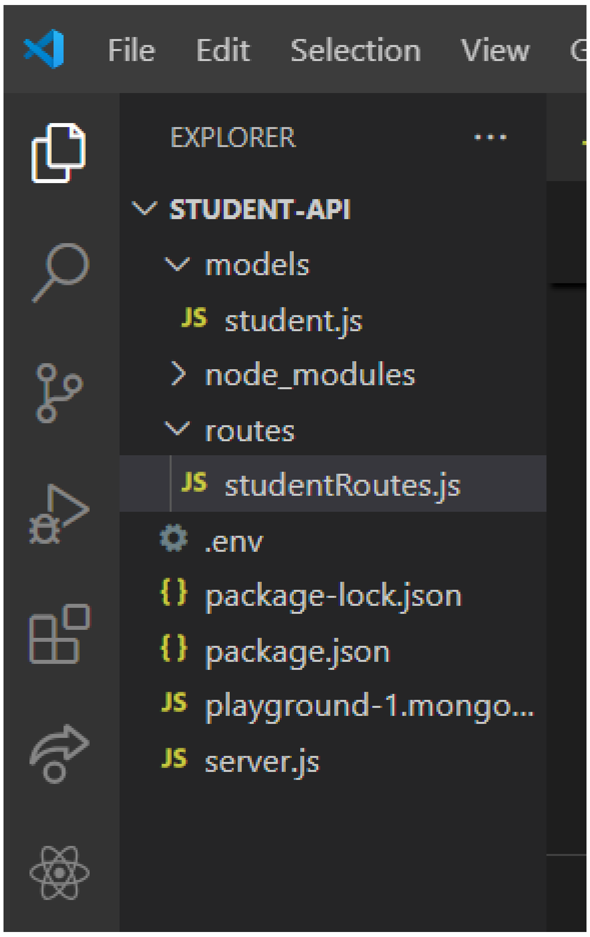


```
{
  "name": "John Doe"
}
```

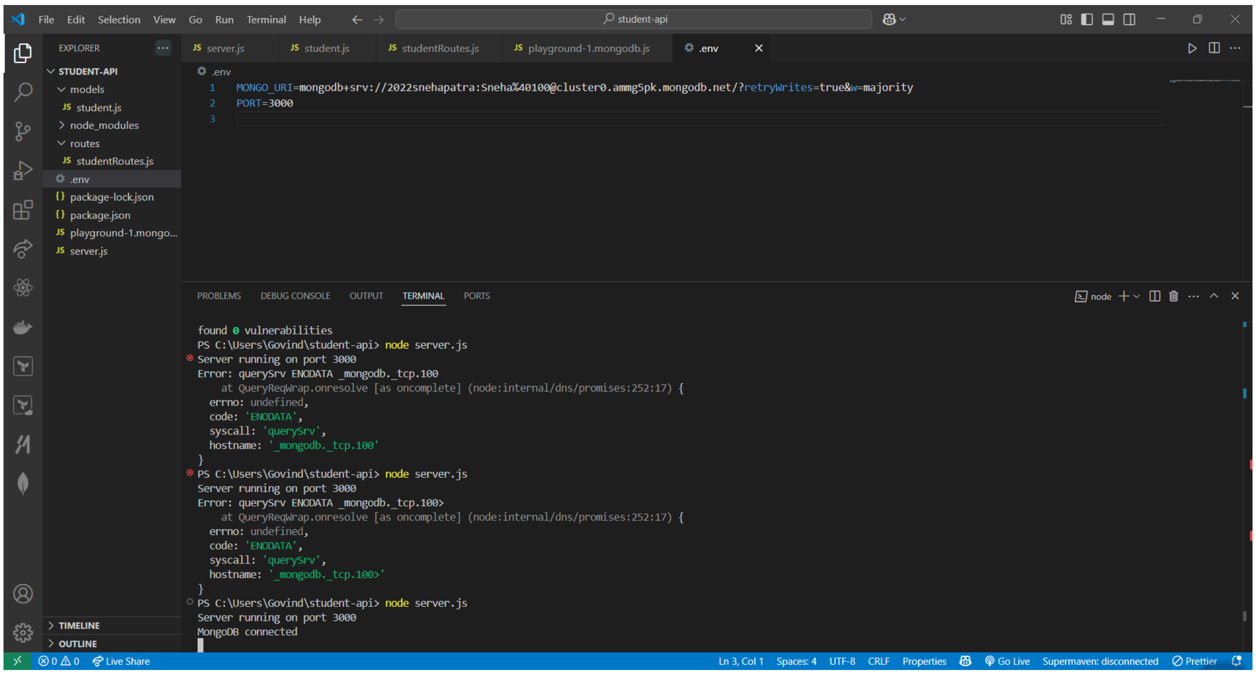
```
_id: ObjectId('67e7a34319f8b49122007b92')
name : "John Doe"
rollNo : 101
className : "IT-1"

_id: ObjectId('67e7a386d2de2d13e9e45fbe')
name : "John Doe"
rollNo : 101
```

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

Folder Structure:



```
STUDENT-API
  models
    student.js
  node_modules
  routes
    studentRoutes.js
  .env
  package-lock.json
  package.json
  playground-1.mongo...
  server.js
```



```
MONGO_URI=mongodb+srv://2022snehapatra:Sneha%40100@cluster0.ammg5pk.mongodb.net/?retryWrites=true&w=majority
PORT=3000
```

```
found 0 vulnerabilities
PS C:\Users\Govind\student-api> node server.js
Server running on port 3000
Error: querySrv ENODATA _mongodb._tcp.100
    at QueryReqWrap.onresolve [as oncomplete] (node:internal/dns/promises:252:17) {
  errno: undefined,
  code: 'ENODATA',
  syscall: 'querySrv',
  hostname: '_mongodb._tcp.100'
}
PS C:\Users\Govind\student-api> node server.js
Server running on port 3000
Error: querySrv ENODATA _mongodb._tcp.100>
    at QueryReqWrap.onresolve [as oncomplete] (node:internal/dns/promises:252:17) {
  errno: undefined,
  code: 'ENODATA',
  syscall: 'querySrv',
  hostname: '_mongodb._tcp.100>'
}
PS C:\Users\Govind\student-api> node server.js
Server running on port 3000
MongoDB connected
```

1. Retrieve all students

GET `/students`

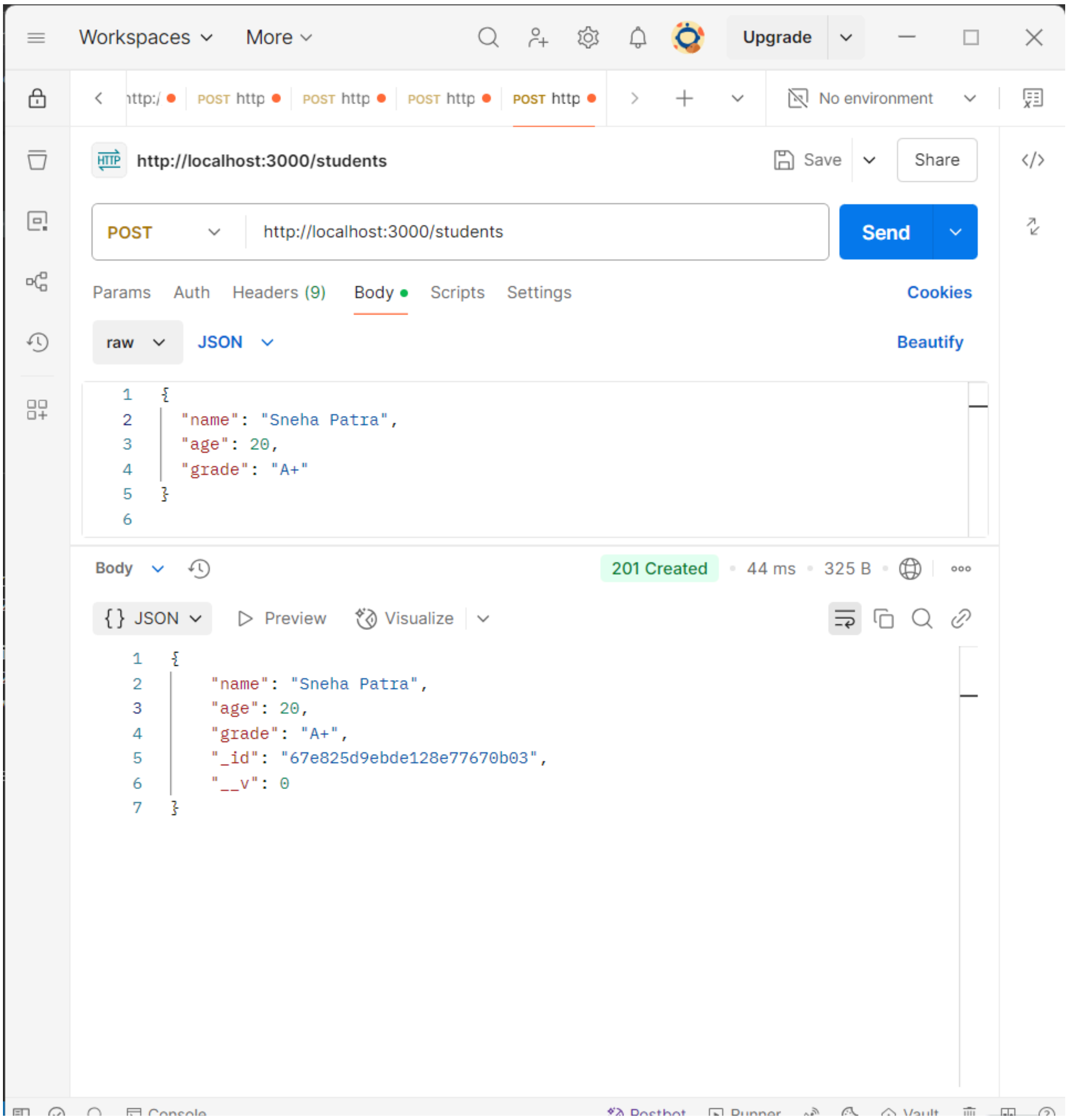
2. Retrieve details of a specific student by ID

GET `/students/:id`

3. Add a new student

POST `/students`
Request Body:

```
{

  "name": "Sneha Patra",

  "age": 20,

  "grade": "A+"

}
```

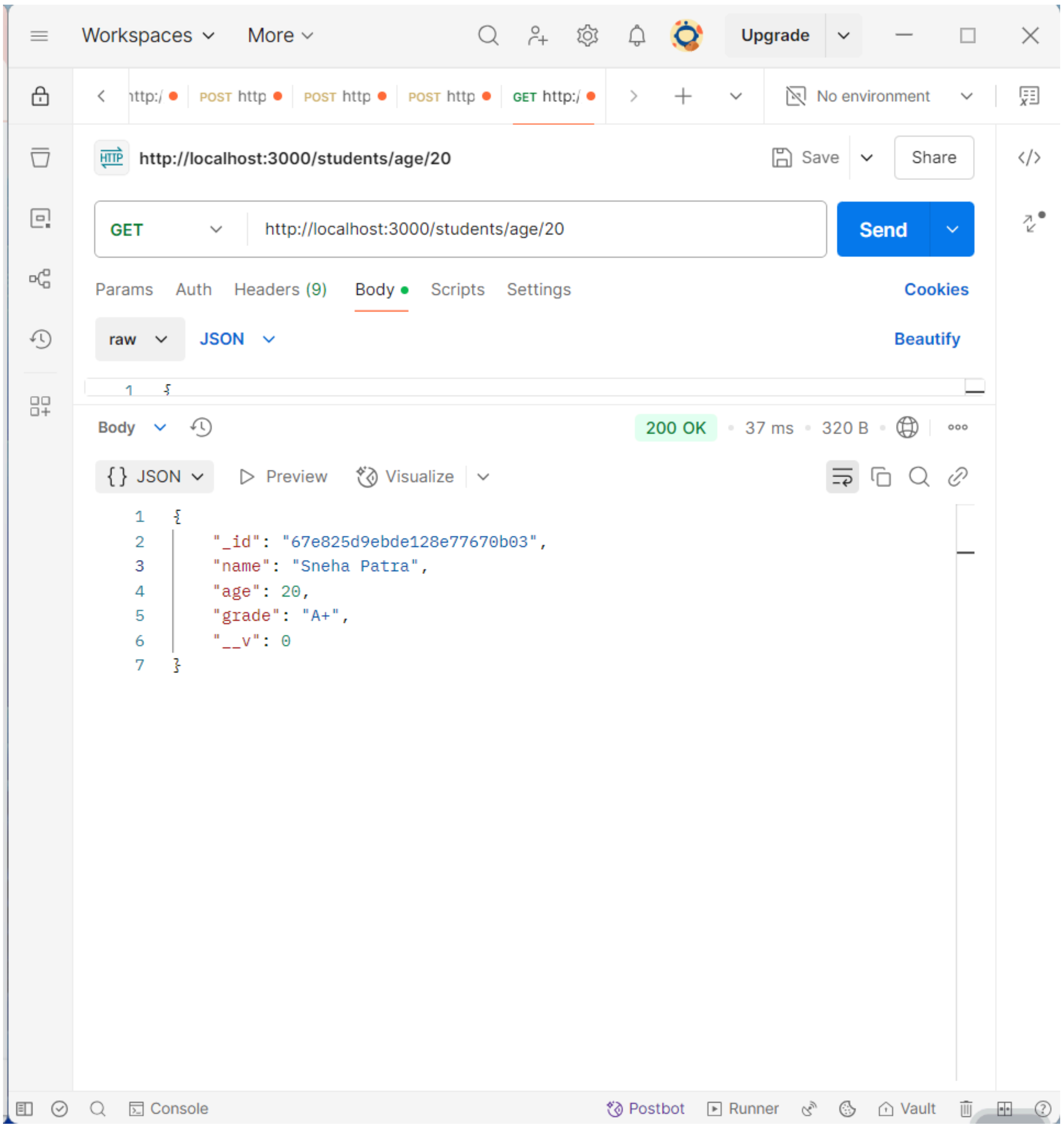


POST http://localhost:3000/students

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

201 Created • 44 ms • 325 B

```
{
    "name": "Sneha Patra",
    "age": 20,
    "grade": "A+",
    "_id": "67e825d9ebde128e77670b03",
    "__v": 0
}
```



http://localhost:3000/students/age/20

GET http://localhost:3000/students/age/20 Send

Body 200 OK 37 ms 320 B

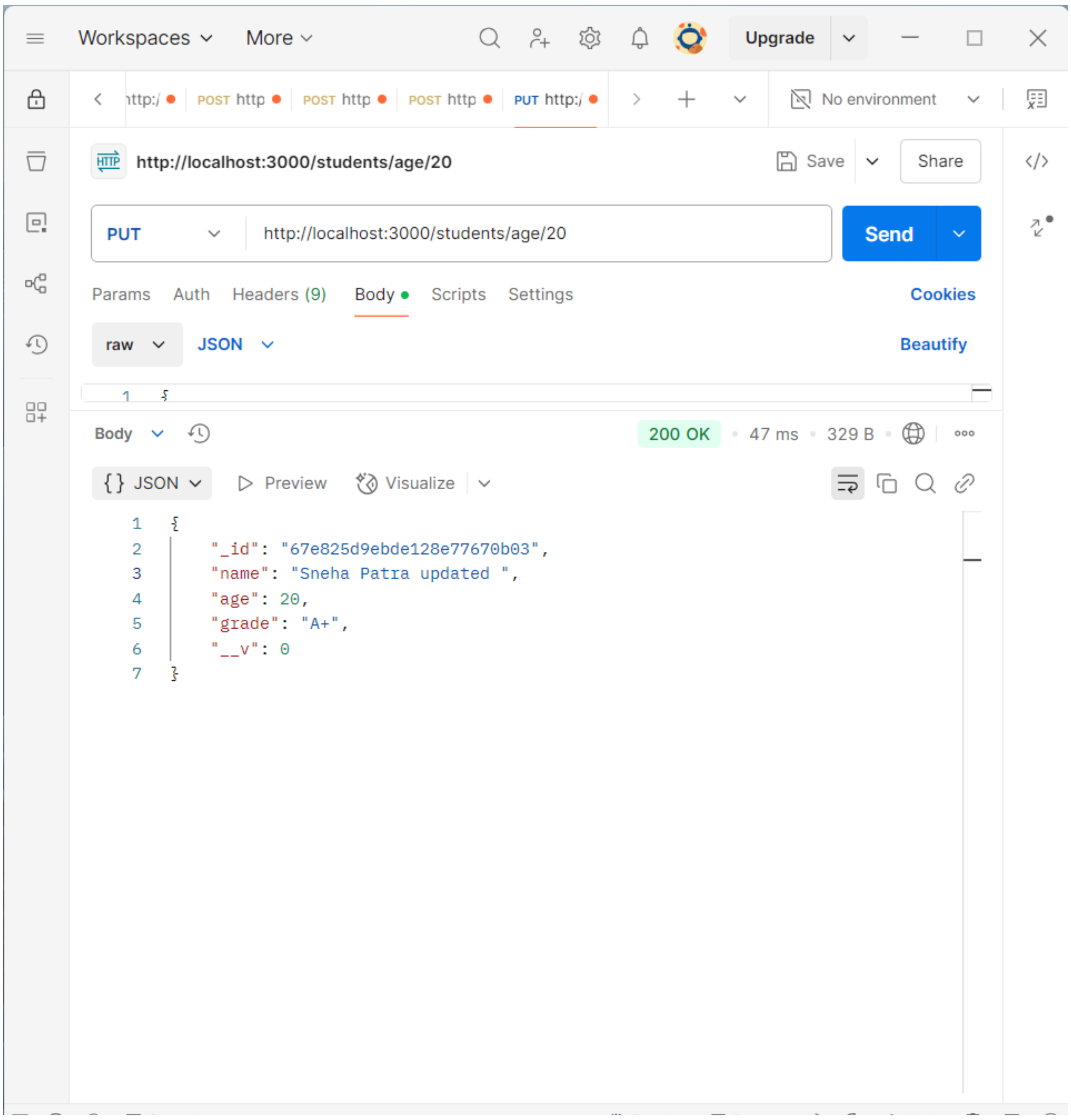
```
{
    "_id": "67e825d9ebde128e77670b03",
    "name": "Sneha Patra",
    "age": 20,
    "grade": "A+",
    "__v": 0
}
```

4. Update student details by ID

PUT `/students/:id`

Request Body:

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



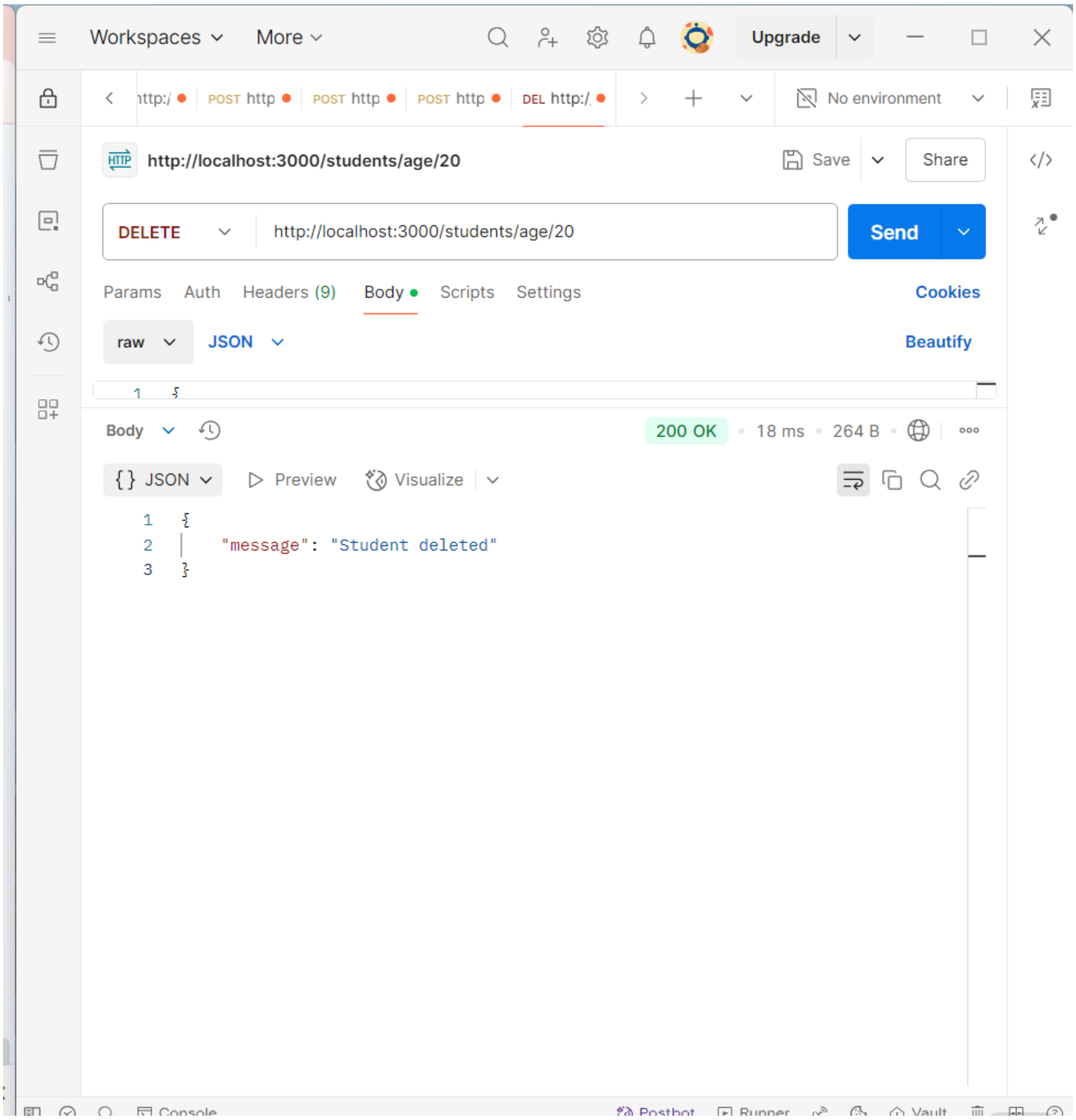
PUT http://localhost:3000/students/age/20

200 OK • 47 ms • 329 B

```
{
    "_id": "67e825d9ebde128e77670b03",
    "name": "Sneha Patra updated ",
    "age": 20,
    "grade": "A+",
    "__v": 0
}
```

5. Delete a student by ID

DELETE `/students/:id`



http://localhost:3000/students/age/20

DELETE http://localhost:3000/students/age/20

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

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.