Experiment – 7: MongoDB

Name of Student	**Sannidhi kailaje**
Class Roll No	**D15A / 22**
D.O.P.	
D.O.S.	
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

1. Insert one student details
2. Insert at once multiple student details
3. Display student for a particular class
4. Display students of specific roll no in a class
5. Change the roll no of a student
6. Delete entries of particular student

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

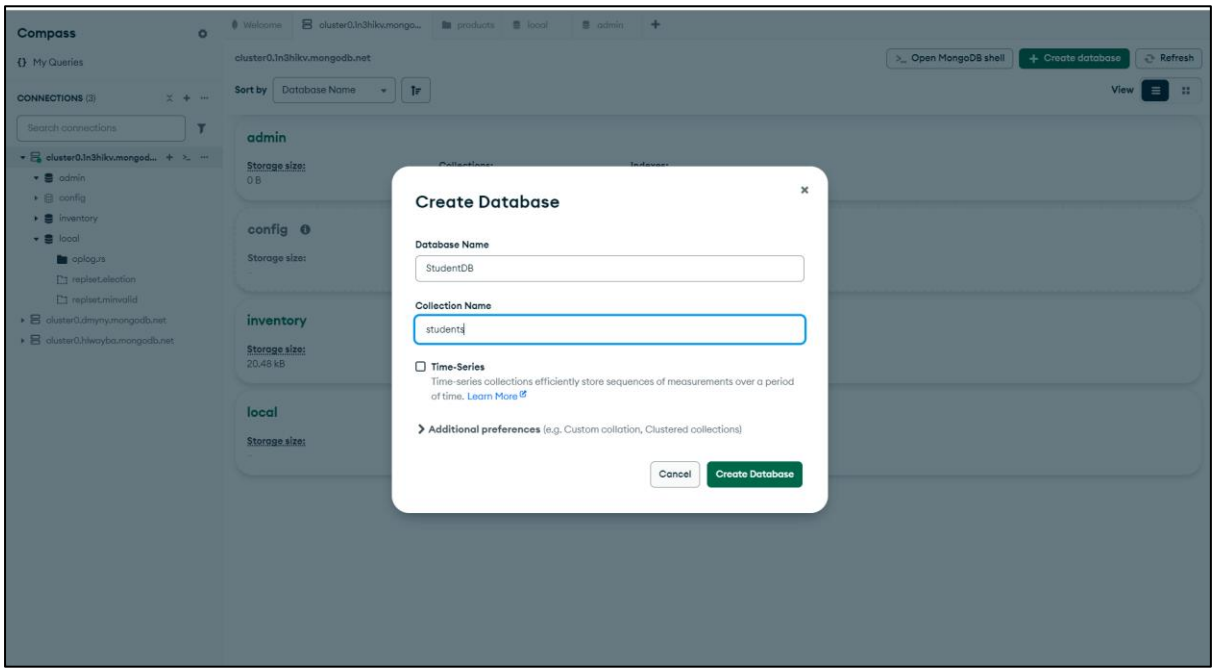
The endpoints should support:

- 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.

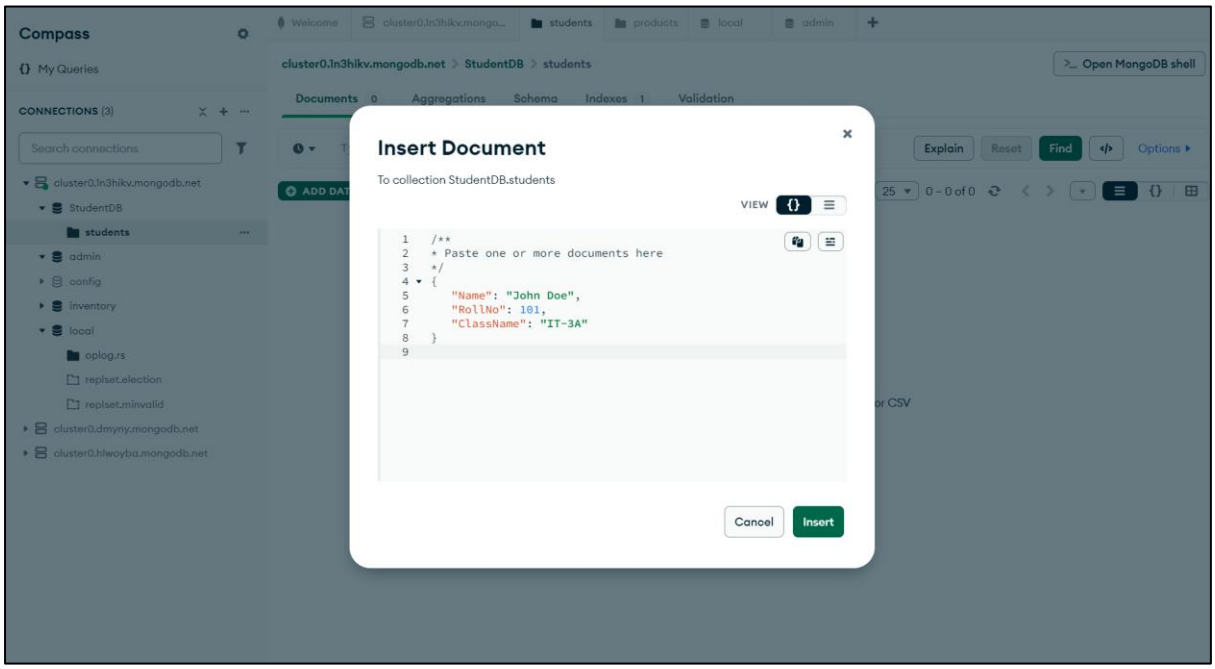
Connect the server to MongoDB using Mongoose, and store student data with attributes: name, age, and grade.

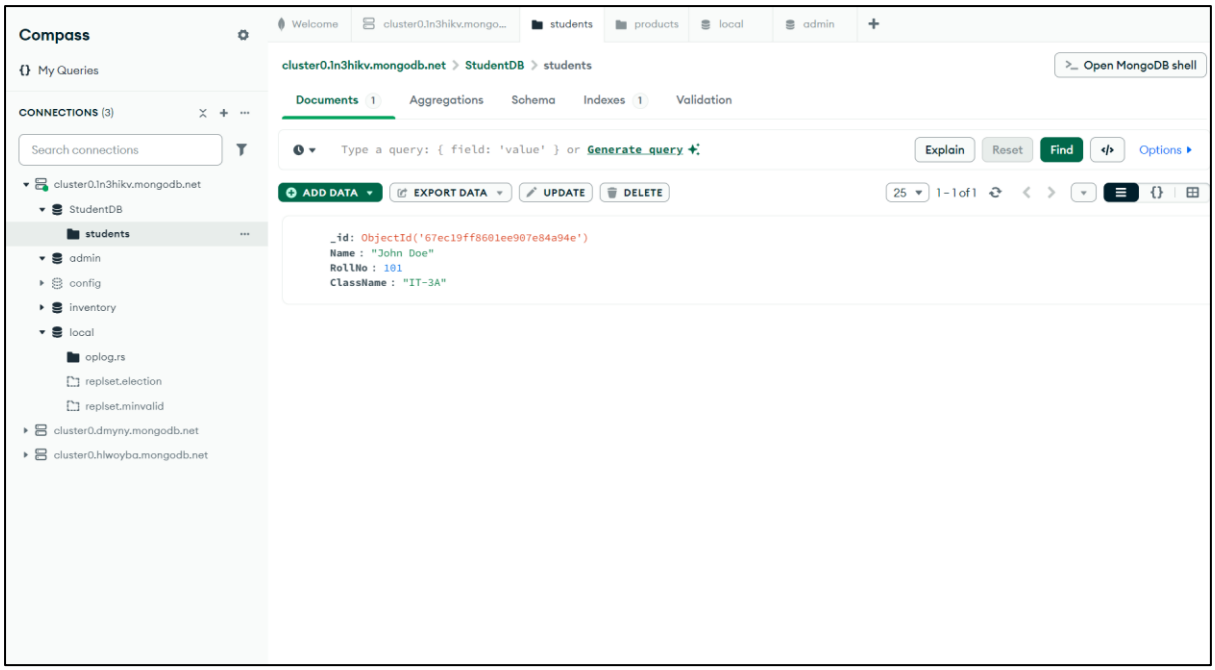
Output:

Create a new database to storage student details of IT dept

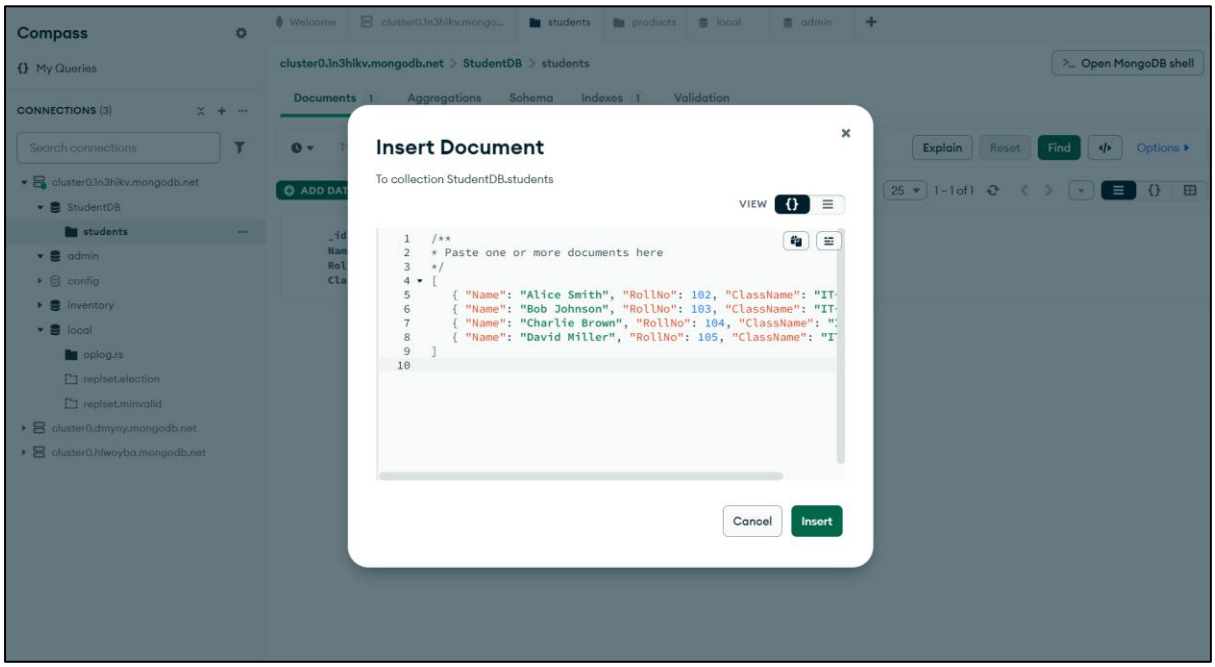


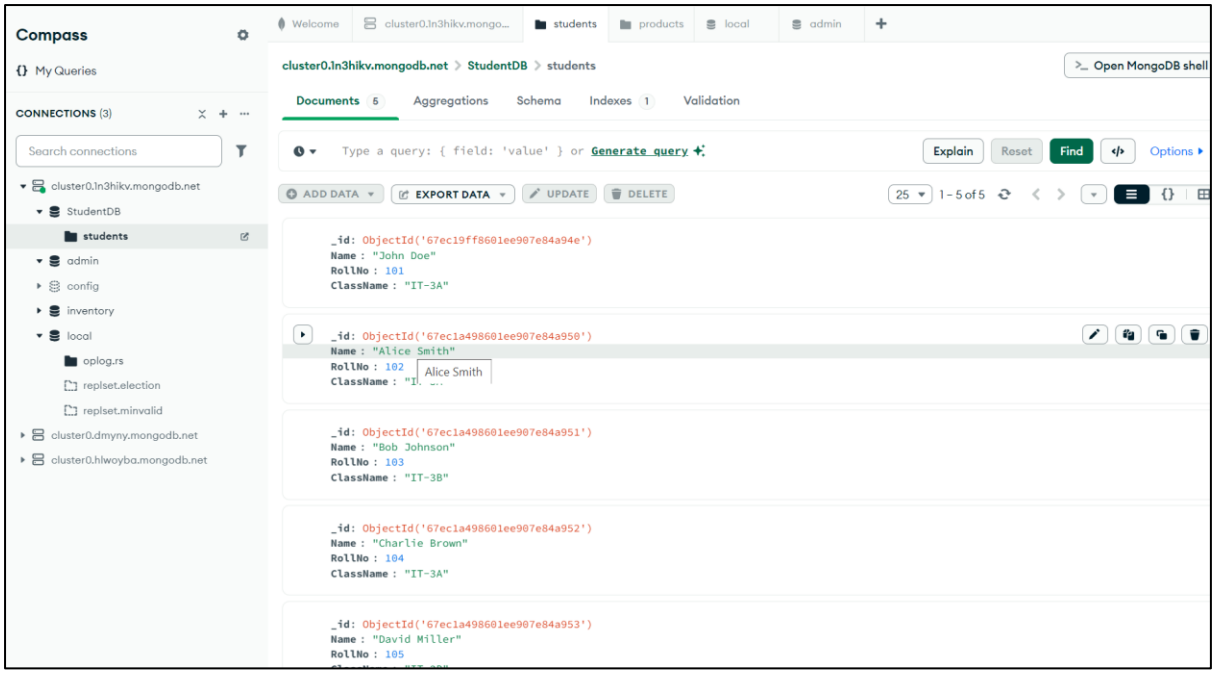
Inserted details of one student.



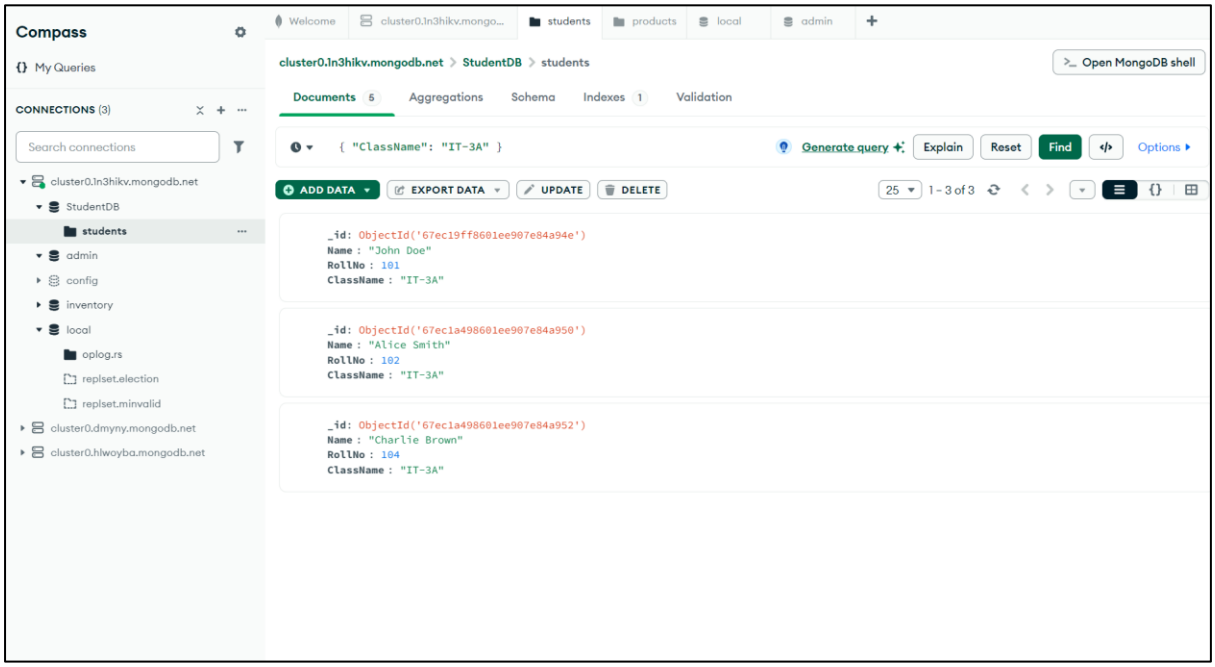


Inserted multiple student details at once.

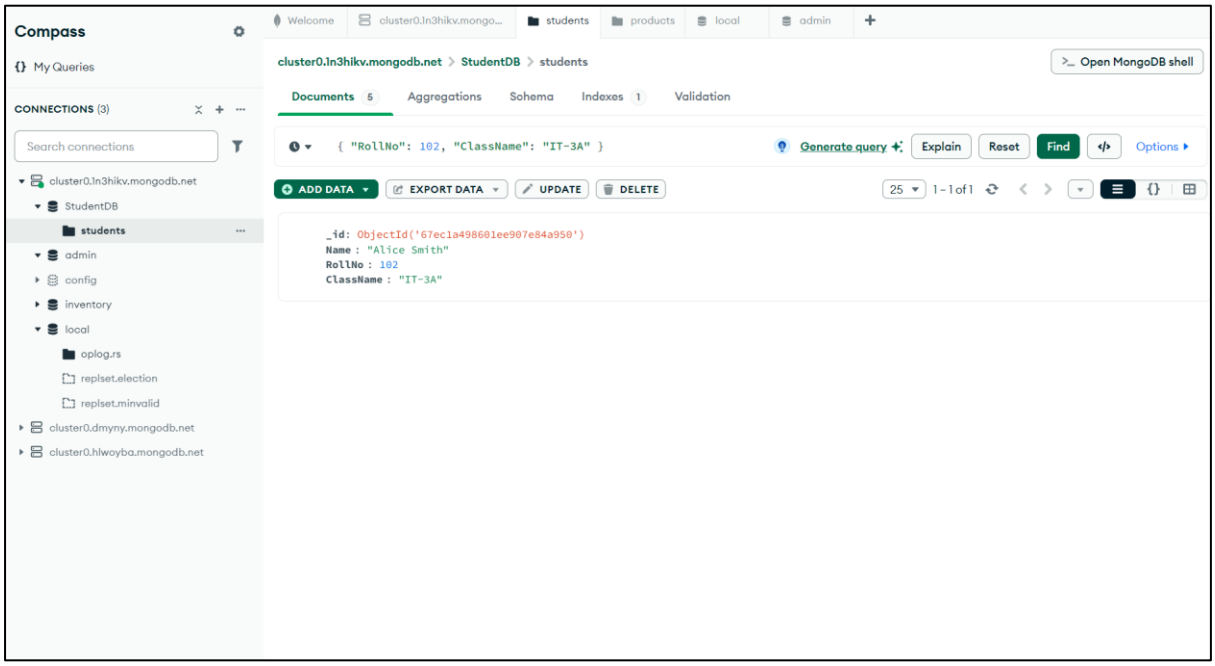




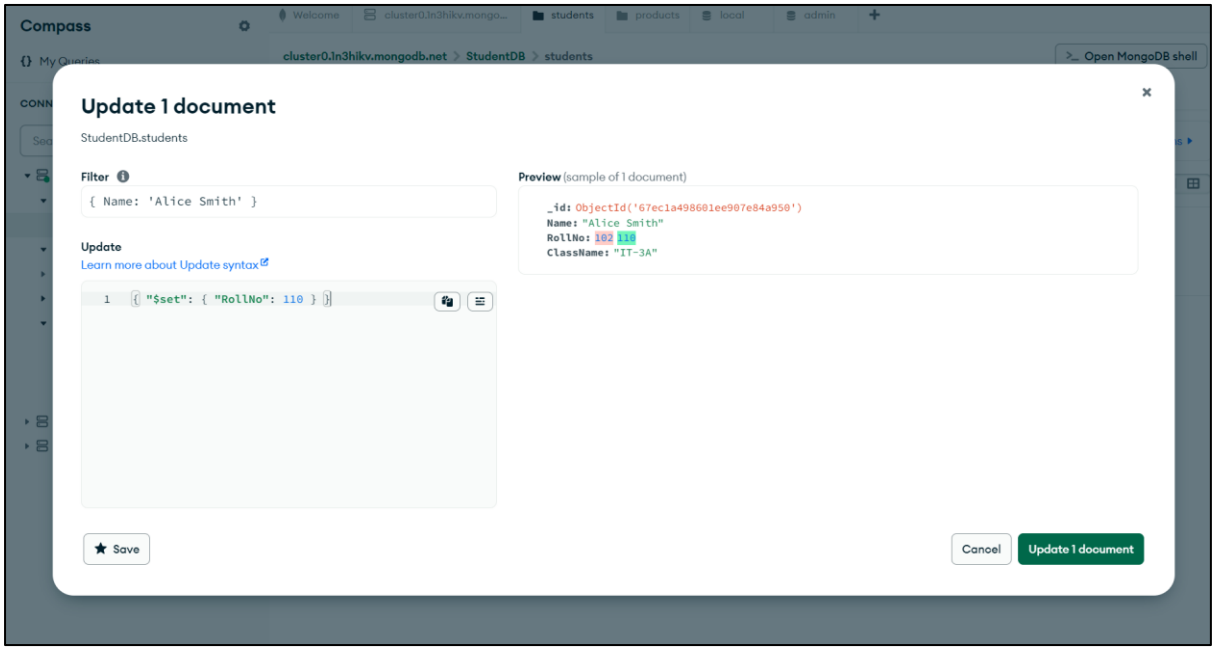
Display student for a particular class

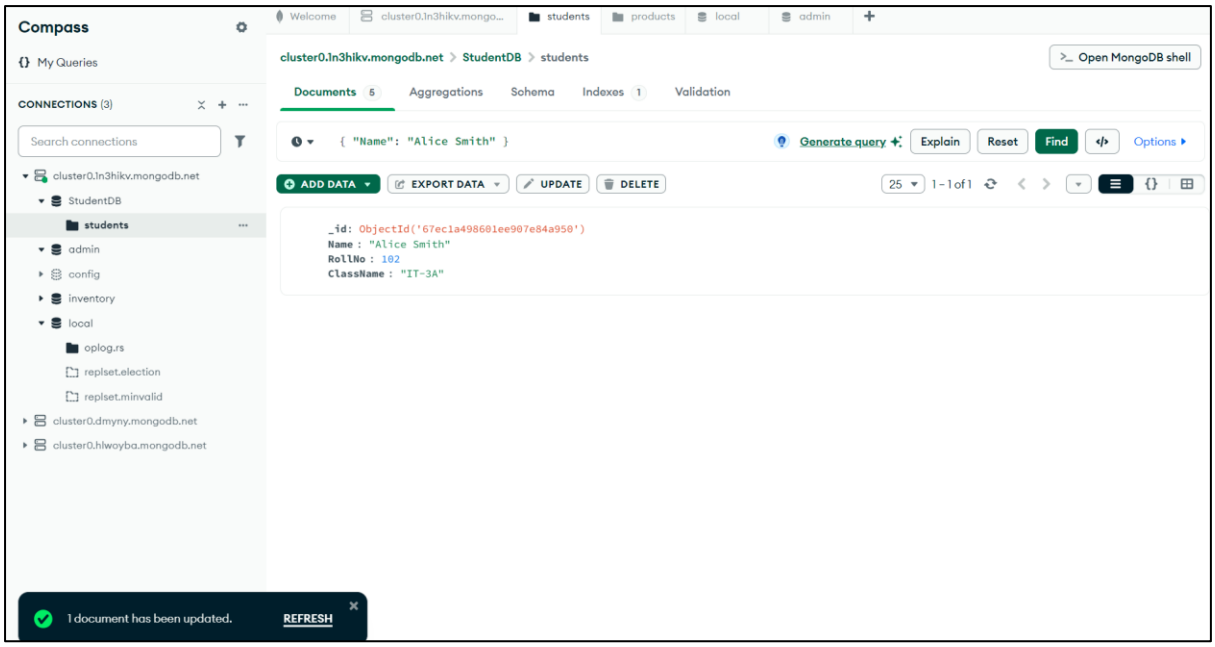


Display students of specific roll no in a class

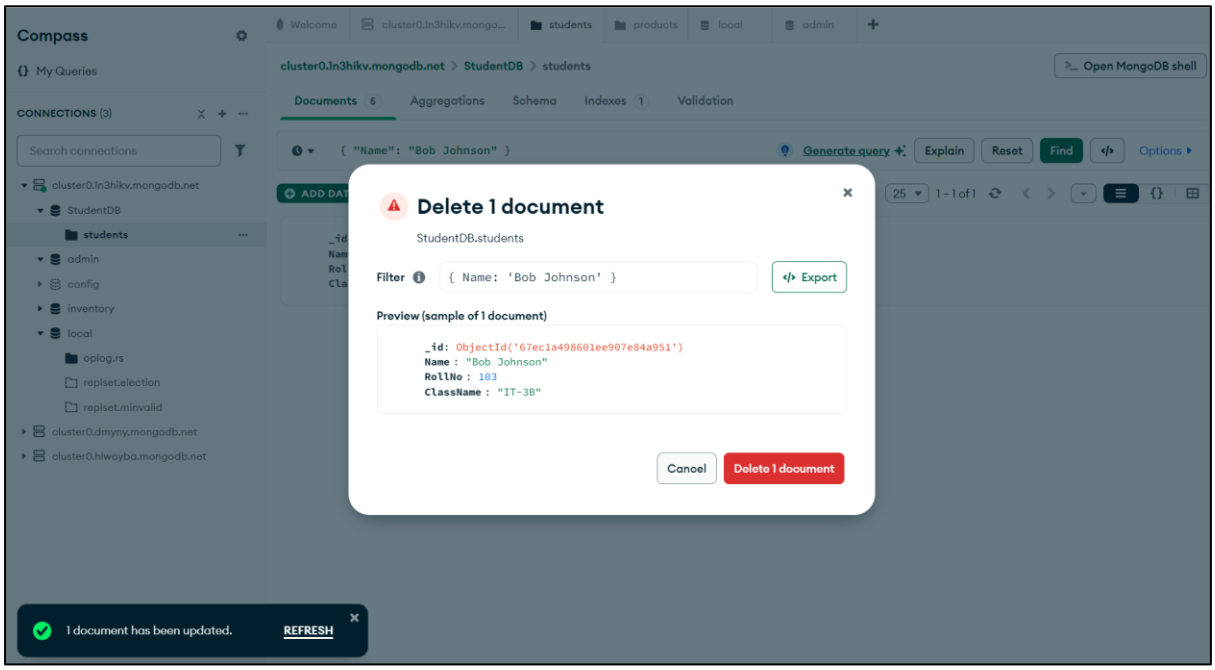


Change the roll no of a student





Delete entries of particular student



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

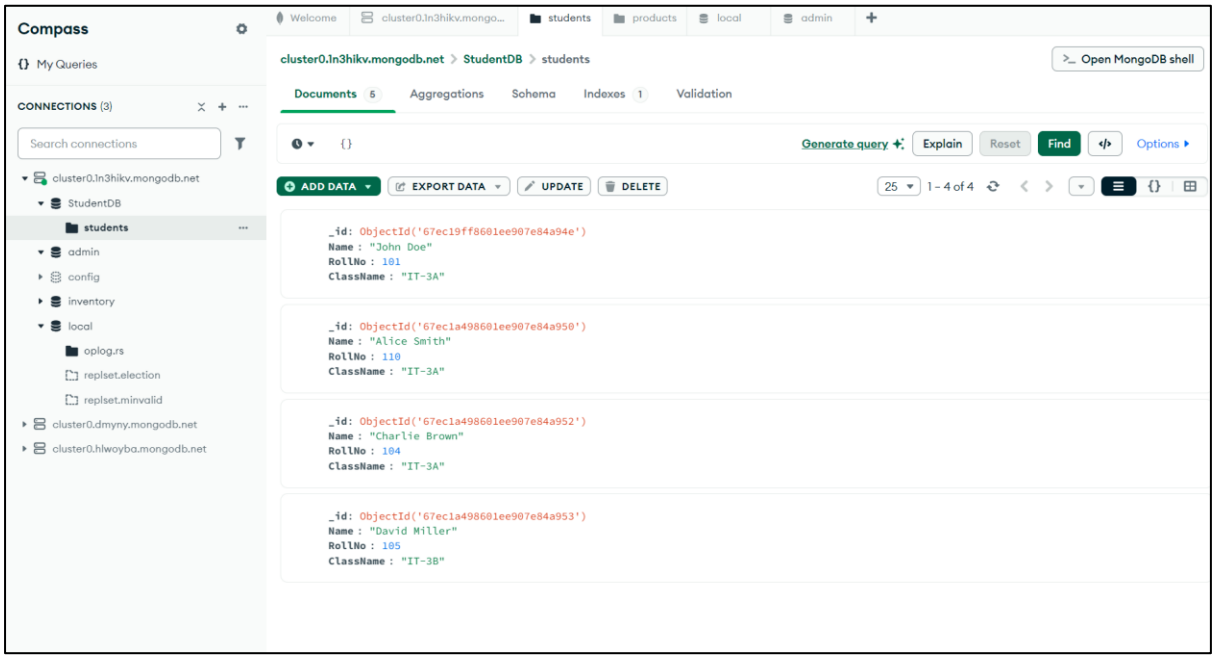
Server.js

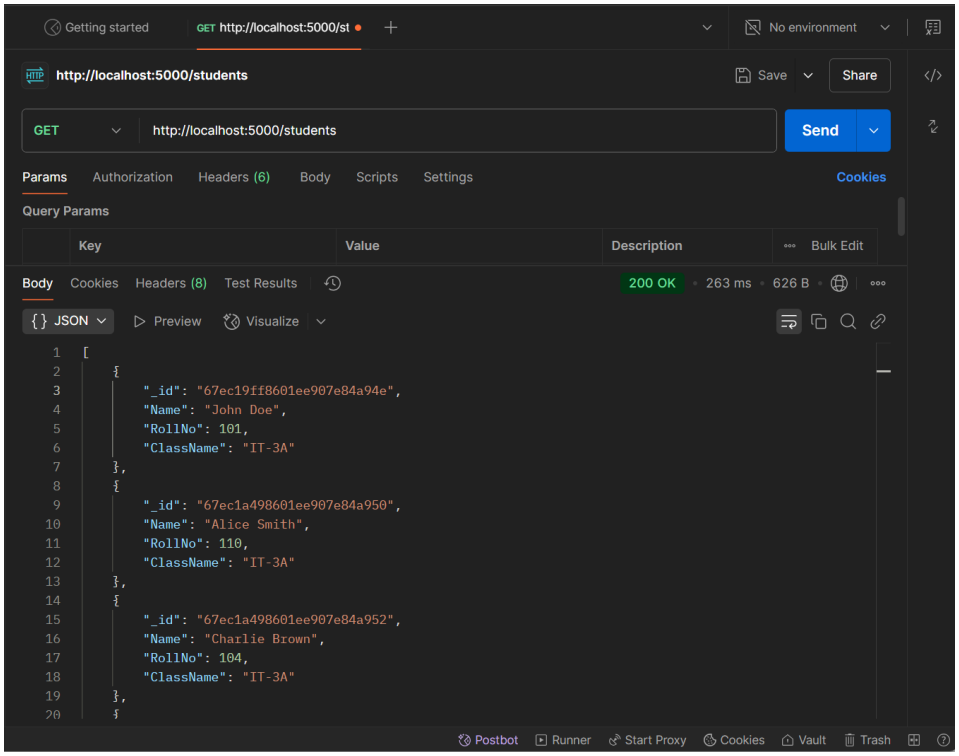
```
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

// Initialize Express App
const app = express();
app.use(cors());
app.use(express.json());

// Connect to MongoDB Atlas
mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('MongoDB Connected'))
  .catch(err => console.log('Error: ', err));

// Define Student Schema (For Student DB)
const studentSchema = new mongoose.Schema({
    Name: { type: String, required: true },
    RollNo: { type: Number, required: true, unique: true },
    ClassName: { type: String, required: true }
});
const Student = mongoose.model('Student', studentSchema);

//  Routes
app.get('/', (req, res) => res.send('Student API is Running 🚀'));

// 1️⃣ Retrieve all students
app.get('/students', async (req, res) => {
    try {
        const students = await Student.find();
        res.json(students);
    } catch (error) {
        res.status(500).json({ message: 'Server Error' });
    }
});
```

```
// 2️⃣ Retrieve a student by ID
app.get('/students/: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: 'Server Error' });
    }
});

// 3️⃣ Add a new student
app.post('/students', async (req, res) => {
    try {
        const { Name, RollNo, ClassName } = req.body;

        // Validation: Ensure all fields are present
        if (!Name || !RollNo || !ClassName) {
            return res.status(400).json({ message: 'Invalid Data. All fields are
required.' });
        }

        const newStudent = new Student({ Name, RollNo, ClassName });
        await newStudent.save();
        res.status(201).json(newStudent);
    } catch (error) {
        if (error.code === 11000) {
            return res.status(400).json({ message: 'Roll Number already exists.' });
        }
        res.status(400).json({ message: 'Invalid Data' });
    }
});

// 4️⃣ Update a student by ID
app.put('/students/:id', async (req, res) => {
    try {
        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);
    } catch (error) {
        res.status(500).json({ message: 'Server Error' });
    }
});
```

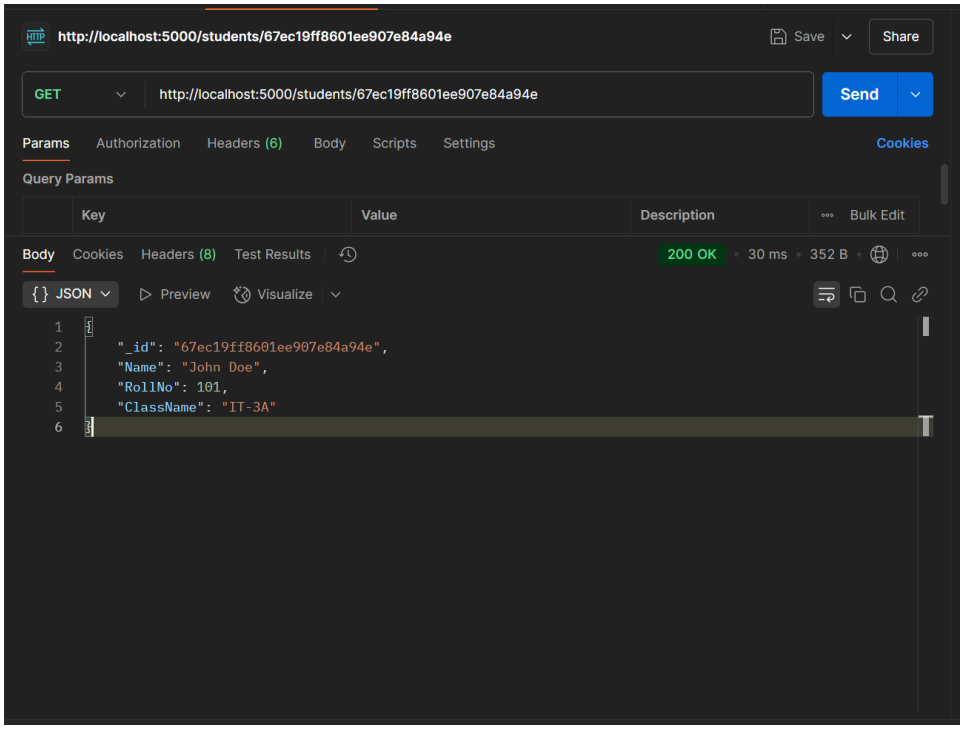
```
// 5. Delete a student by ID
app.delete('/students/:id', async (req, res) => {
    try {
        const deletedStudent = await Student.findByIdAndDelete(req.params.id);
        if (!deletedStudent) return res.status(404).json({ message: 'Student Not
Found' });
        res.json({ message: 'Student Deleted' });
    } catch (error) {
        res.status(500).json({ message: 'Server Error' });
    }
});

// Start Server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Retrieve a list of all students.

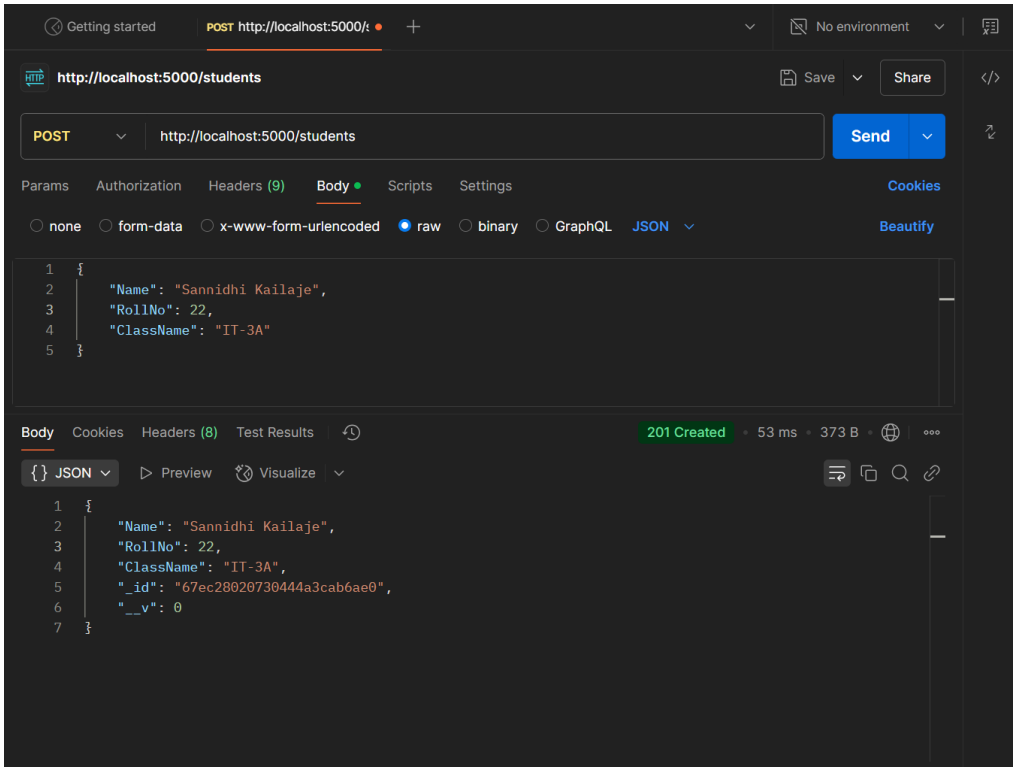




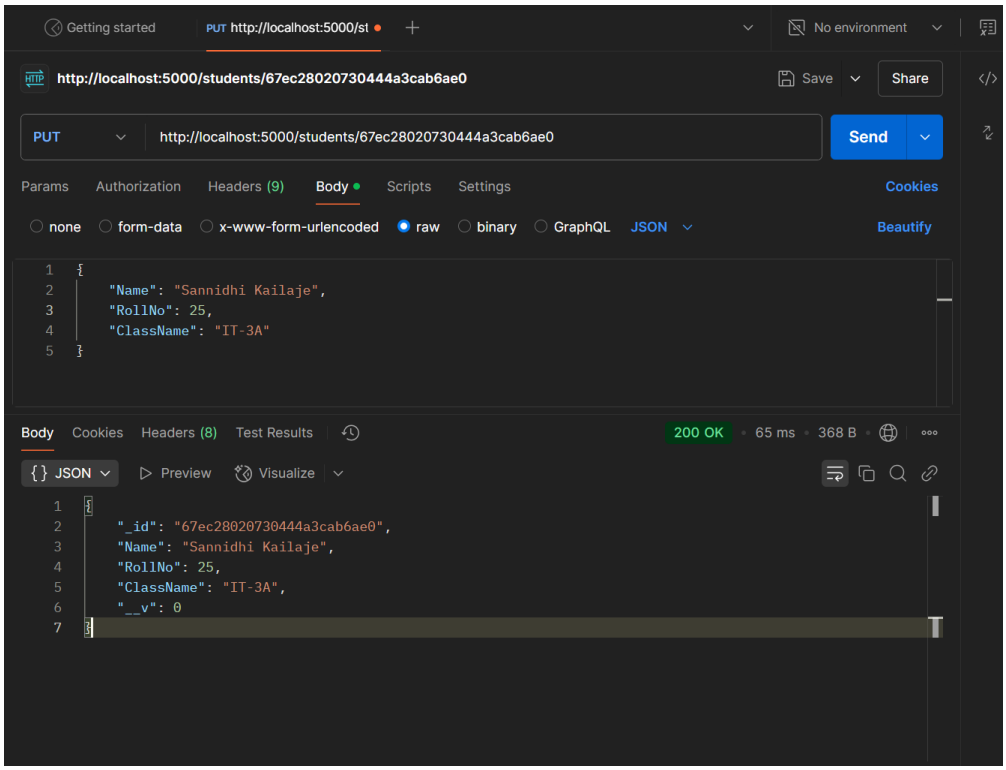
Retrieve details of an individual student by ID.



Add a new student to the database.



Update details of an existing student by ID.



Delete a student from the database by ID.