

Experiment – 7: MongoDB

Name of Student	Ronak Katariya
Class Roll No	D15A / 23
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

The screenshot shows the Compass MongoDB interface. On the left, there's a sidebar with 'My Queries' and a 'CONNECTIONS' section listing several MongoDB clusters. The main area shows the 'admin' database with its collections: 'admin', 'config', 'inventory', and 'local'. A modal window titled 'Create Database' is open in the center. It has fields for 'Database Name' (set to 'StudentDB') and 'Collection Name' (set to 'student'). There are also checkboxes for 'Time-Series' and 'Additional preferences'. At the bottom right of the modal are 'Cancel' and 'Create Database' buttons.

Inserted details of one student.

The screenshot shows the Compass MongoDB interface. The left sidebar shows connections and the 'admin' database. The main area shows the 'StudentDB' database with its collection 'students'. A modal window titled 'Insert Document' is open, indicating it's for the 'students' collection. The text area contains the following JSON document:

```
1  /**
2  * Paste one or more documents here
3  */
4  [
5      {
6          "Name": "John Doe",
7          "RollNo": 101,
8          "ClassName": "IT-3A"
9      }
]
```

At the bottom right of the modal are 'Cancel' and 'Insert' buttons.

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays 'cluster0.in3hikv.mongodb.net' with its databases: 'StudentDB' (selected), 'admin', 'config', 'inventory', and 'local'. Under 'StudentDB', the 'students' collection is selected. The main panel shows the 'Documents' tab with one document listed. The document details are as follows:

```
_id: ObjectId('67ec19ff8601ee907e84a94e')
Name : "John Doe"
RollNo : 101
ClassName : "IT-3A"
```

Below the document list are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The bottom right corner shows pagination: '25' and '1-1 of 1'.

Inserted multiple student details at once.

The screenshot shows the Compass MongoDB interface with the 'Insert Document' dialog box open. The dialog is titled 'Insert Document' and specifies 'To collection StudentDB.students'. The text area contains the following JSON documents:

```
1  /**
2   * Paste one or more documents here
3   */
4  [
5    {
6      "Name": "Alice Smith", "RollNo": 102, "ClassName": "IT-3B"
7    },
8    {
9      "Name": "Bob Johnson", "RollNo": 103, "ClassName": "IT-3B"
10   },
11   {
12     "Name": "Charlie Brown", "RollNo": 104, "ClassName": "IT-3B"
13   },
14   {
15     "Name": "David Miller", "RollNo": 105, "ClassName": "IT-3B"
16   }
]
```

At the bottom of the dialog are 'Cancel' and 'Insert' buttons.

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays three connections: cluster0.In3hikv.mongodb.net, cluster0.dmymy.mongodb.net, and cluster0.hlwoyba.mongodb.net. Under the first connection, the StudentDB database is selected, and its 'students' collection is shown. The 'Documents' tab is active, displaying five documents. Each document contains the following fields:

- `_id: ObjectId('67ec19ff8601ee907e84a94e')`
- `Name: "John Doe"`
- `RollNo: 101`
- `ClassName: "IT-3A"`

The second document is highlighted, showing the same structure. The other three documents are partially visible below it.

Display student for a particular class

This screenshot shows the same Compass MongoDB interface as the previous one, but with a query applied. The search bar at the top contains the query `{"ClassName": "IT-3A"}`. The results show only three documents that match this criteria:

- `_id: ObjectId('67ec19ff8601ee907e84a94e')`
- `Name: "John Doe"`
- `RollNo: 101`
- `ClassName: "IT-3A"`

The second document is highlighted. The other two are partially visible below it.

Display students of specific roll no in a class

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays three connections: cluster0.In3hikv.mongodb.net, cluster0.dmynny.mongodb.net, and cluster0.hlwoyba.mongodb.net. Under the first connection, the 'StudentDB' database is selected, and its 'students' collection is shown. The 'Documents' tab is active, displaying a single document with the following data:

```
{ "RollNo": 102, "ClassName": "IT-3A" }
```

Below the document, the preview pane shows the same document with the RollNo field highlighted in red.

Change the roll no of a student

The screenshot shows the Compass MongoDB interface with an open update dialog for the 'students' collection in the 'StudentDB'. The dialog title is 'Update 1 document'. The 'Filter' section contains the query: { Name: 'Alice Smith' }. The 'Update' section contains the command: `1 [{ "$set": { "RollNo": 110 } }]`. The 'Preview' section shows the resulting document after the update:

```
_id: ObjectId('67ec1a498601ee907e84a950')
Name: "Alice Smith"
RollNo: 110
ClassName: "IT-3A"
```

At the bottom right of the dialog are 'Cancel' and 'Update 1 document' buttons.

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays 'cluster0.In3hikv.mongodb.net' with its databases: 'StudentDB' (selected), 'admin', 'config', 'inventory', and 'local'. Under 'StudentDB', the 'students' collection is selected. The main pane shows a list of documents with one document highlighted: '_id: ObjectId('67ec1a498601ee907e84a950') Name : 'Alice Smith' RollNo : 102 ClassName : 'IT-3A''. Below this, there are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. A status bar at the bottom indicates '1 document has been updated.'

Delete entries of particular student

The screenshot shows the Compass MongoDB interface with a modal dialog titled 'Delete 1 document'. The dialog is centered over the 'students' collection in the 'StudentDB'. It contains the message 'Delete 1 document' and 'StudentDB.students'. Below this is a 'Filter' section with the query '{ Name: 'Bob Johnson' }'. A preview section shows a single document: '_id: ObjectId('67ec1a498601ee907e84a951') Name : 'Bob Johnson' RollNo : 103 ClassName : 'IT-3B''. At the bottom of the dialog are 'Cancel' and 'Delete 1 document' buttons. A status bar at the bottom of the main window indicates '1 document has been updated.'

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 🚀'))

// 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' });
  }
});
```

```
// 1 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' });
  } catch (error) {
    res.status(500).json({ message: 'Server Error' });
  }
});

// 2 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' });
  }
});

// 3 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}`));

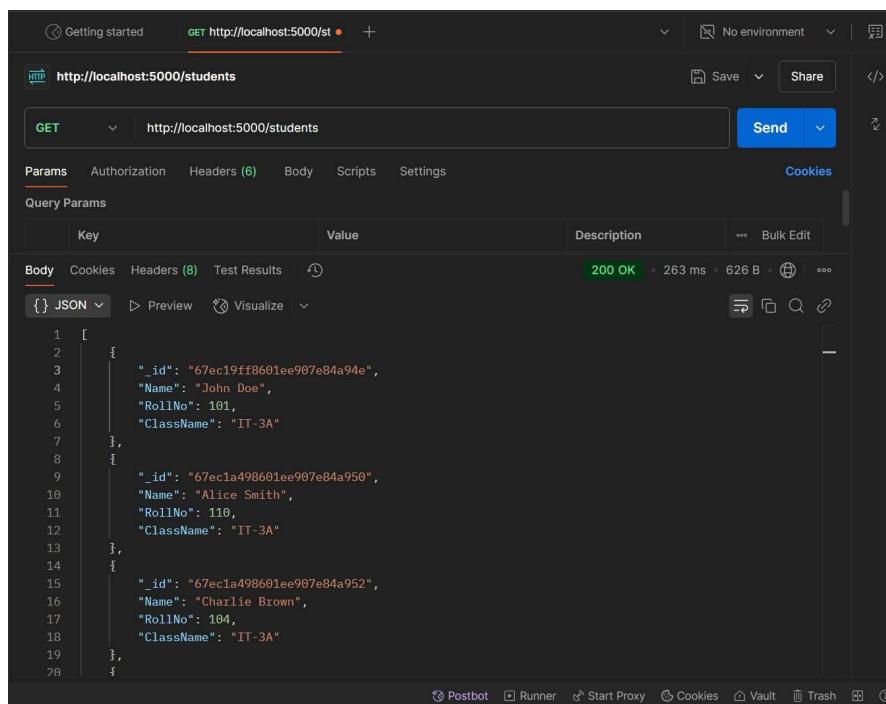
```

Retrieve a list of all students.

The screenshot shows the MongoDB Compass interface. On the left, the connection tree displays 'cluster0.in3hikv.mongodb.net' with 'StudentDB' selected. Under 'StudentDB', the 'students' collection is expanded, showing four documents. The documents are listed as follows:

- Document 1:** _id: ObjectId('67ec19ff860lee907e84a94e'), Name: "John Doe", RollNo: 101, ClassName: "IT-3A"
- Document 2:** _id: ObjectId('67ec1a49860lee907e84a950'), Name: "Alice Smith", RollNo: 110, ClassName: "IT-3A"
- Document 3:** _id: ObjectId('67ec1a49860lee907e84a952'), Name: "Charlie Brown", RollNo: 104, ClassName: "IT-3A"
- Document 4:** _id: ObjectId('67ec1a49860lee907e84a953'), Name: "David Miller", RollNo: 105, ClassName: "IT-3B"

The top navigation bar includes tabs for 'Welcome', 'cluster0.in3hikv.mongodb.net', 'students', 'products', 'local', 'admin', and '+'. The 'students' tab is active. Below the tabs are buttons for 'Open MongoDB shell', 'Generate query', 'Explain', 'Reset', 'Find', 'Options', 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The document list has a '25' dropdown, a '1-4 of 4' indicator, and navigation arrows.



HTTP <http://localhost:5000/students>

GET <http://localhost:5000/students>

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit

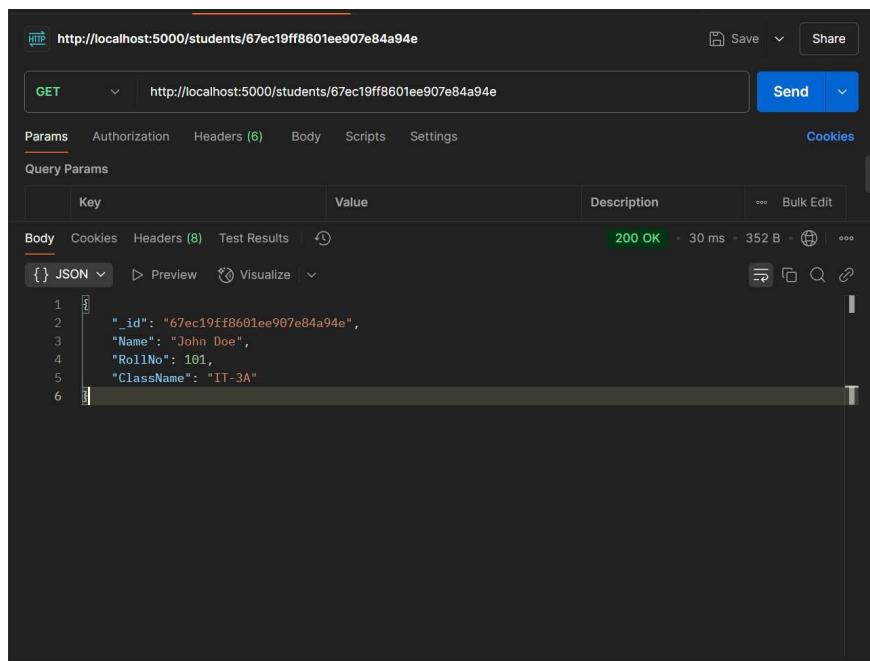
Body Cookies Headers (8) Test Results

200 OK • 263 ms • 626 B • ↗

```
1 [  
2 {  
3   "_id": "67ec19ff8601ee907e84a94e",  
4   "Name": "John Doe",  
5   "RollNo": 101,  
6   "ClassName": "IT-3A"  
7 },  
8 {  
9   "_id": "67ec1a498601ee907e84a950",  
10  "Name": "Alice Smith",  
11  "RollNo": 110,  
12  "ClassName": "IT-3A"  
13 },  
14 {  
15  "_id": "67ec1a498601ee907e84a952",  
16  "Name": "Charlie Brown",  
17  "RollNo": 104,  
18  "ClassName": "IT-3A"  
19 },  
20 ]
```

Postbot Runner Start Proxy Cookies Vault Trash

Retrieve details of an individual student by ID.



HTTP <http://localhost:5000/students/67ec19ff8601ee907e84a94e>

GET <http://localhost:5000/students/67ec19ff8601ee907e84a94e>

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

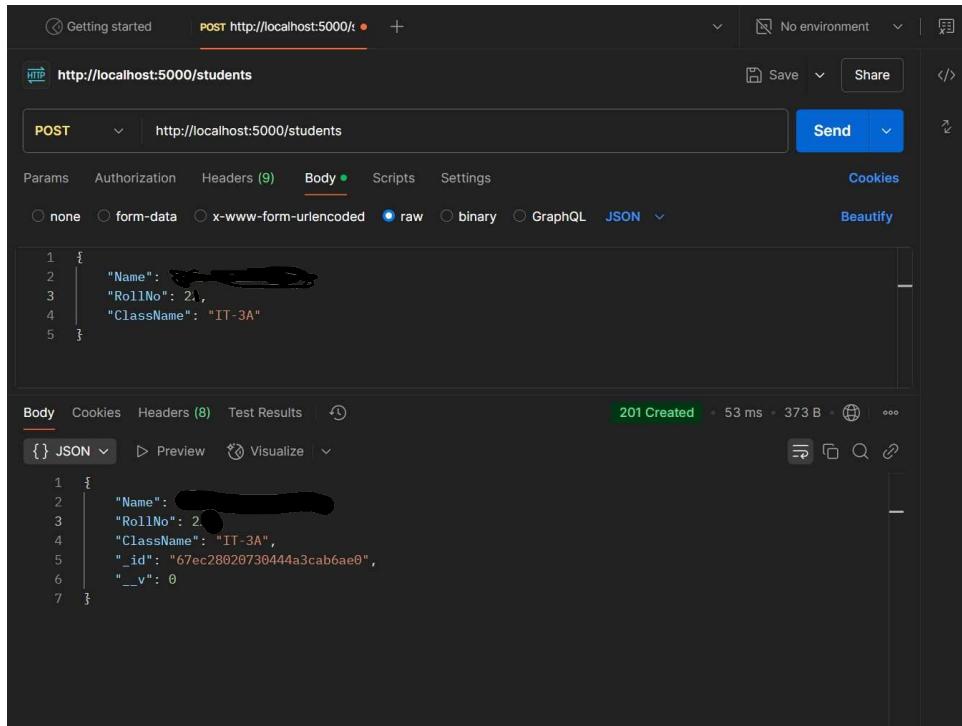
Key	Value	Description	...	Bulk Edit

Body Cookies Headers (8) Test Results

200 OK • 30 ms • 352 B • ↗

```
1 {  
2   "_id": "67ec19ff8601ee907e84a94e",  
3   "Name": "John Doe",  
4   "RollNo": 101,  
5   "ClassName": "IT-3A"  
6 }
```

Add a new student to the database.

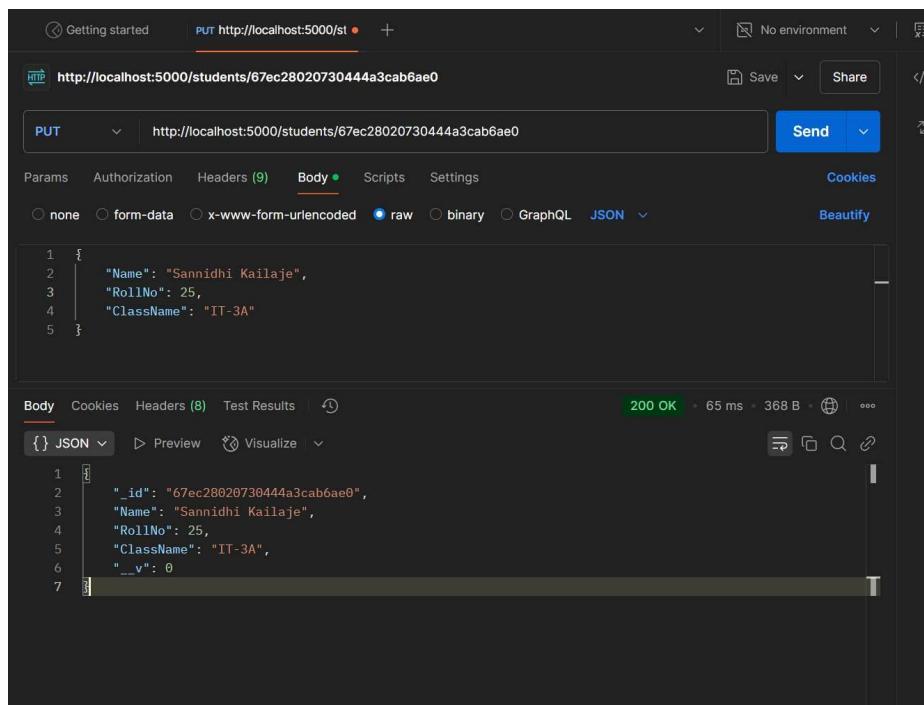


The screenshot shows the Postman interface with a POST request to `http://localhost:5000/students`. The request body contains the following JSON:

```
1 {
2     "Name": "Sannidhi Kailaje",
3     "RollNo": 25,
4     "ClassName": "IT-3A"
5 }
```

The response status is `201 Created` with a response time of 53 ms and a response size of 373 B. The response body is identical to the request body.

Update details of an existing student by ID.



The screenshot shows the Postman interface with a PUT request to `http://localhost:5000/students/67ec28020730444a3cab6ae0`. The request body contains the following JSON:

```
1 {
2     "Name": "Sannidhi Kailaje",
3     "RollNo": 25,
4     "ClassName": "IT-3A"
5 }
```

The response status is `200 OK` with a response time of 65 ms and a response size of 368 B. The response body is identical to the request body.

Delete a student from the database by ID.

The screenshot shows the Postman application interface. At the top, there's a header bar with 'Getting started' and 'http://localhost:5000/st'. Below it, the main URL is 'http://localhost:5000/students/67ec19ff8601ee907e84a94e'. The method is set to 'DELETE'. On the right side of the header, there are 'Save' and 'Share' buttons. Underneath the URL, the status is 'No environment'. The main content area has tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is selected, showing options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', 'GraphQL', and 'JSON'. 'raw' is selected. The 'Body' section contains the JSON response: { "message": "Student Deleted" }. Above the body, there are buttons for 'Send' (highlighted in blue), 'Cookies', 'Beautify', and other test results. The status bar at the bottom indicates '200 OK', '30 ms', '296 B', and a globe icon.