

## Experiment – 7: MongoDB

Name of Student	Prajwal Pandey
Class Roll No	D15A / 32
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 titled 'CONNECTIONS (3)' listing databases: admin, config, inventory, local, and two external connections. The main area shows the 'admin' database with its storage size at 0 B. A modal window titled 'Create Database' is open in the center. It contains 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 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 the same connection list as before. The main area is focused on the 'students' collection within the 'StudentDB' database. A modal window titled 'Insert Document' is open, prompting the user to 'To collection StudentDB.students'. Below this, a code editor displays a single document being inserted: 

```
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 of the modal 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.hlwayba.mongodb.net. Under the first connection, the StudentDB database is selected, and its 'students' collection is shown. The 'Documents' tab is active, showing one document. 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'. To the right, there are navigation and search controls.

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 main area contains a code editor with the following JSON array:

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

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

The screenshot shows the Compass MongoDB interface. The left sidebar displays connections: cluster0.in3hikv.mongodb.net (selected), cluster0.dmnyy.mongodb.net, and cluster0.hlwoyba.mongodb.net. Under the selected connection, the 'StudentDB' database is expanded, showing the 'students' collection (5 documents), 'admin', 'config', 'inventory', and 'local' databases. The 'students' collection page has tabs for 'Documents' (5), 'Aggregations', 'Schema', 'Indexes' (1), and 'Validation'. A search bar at the top says 'Type a query: { field: 'value' } or Generate query'. Below it are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The main area lists five student documents:

- `_id: ObjectId('67ec19ff8601ee907e84a94e')`  
Name : "John Doe"  
RollNo : 101  
ClassName : "IT-3A"
- `_id: ObjectId('67ec1a498601ee907e84a950')`  
Name : "Alice Smith"  
RollNo : 102  
ClassName : "IT-3A"
- `_id: ObjectId('67ec1a498601ee907e84a951')`  
Name : "Bob Johnson"  
RollNo : 103  
ClassName : "IT-3B"
- `_id: ObjectId('67ec1a498601ee907e84a952')`  
Name : "Charlie Brown"  
RollNo : 104  
ClassName : "IT-3A"
- `_id: ObjectId('67ec1a498601ee907e84a953')`  
Name : "David Miller"  
RollNo : 105

## Display student for a particular class

This screenshot is similar to the first one but shows a query being run. The search bar at the top contains the query `{"ClassName": "IT-3A"}`. The results show only three documents from the previous list:

- `_id: ObjectId('67ec19ff8601ee907e84a94e')`  
Name : "John Doe"  
RollNo : 101  
ClassName : "IT-3A"
- `_id: ObjectId('67ec1a498601ee907e84a950')`  
Name : "Alice Smith"  
RollNo : 102  
ClassName : "IT-3A"
- `_id: ObjectId('67ec1a498601ee907e84a952')`  
Name : "Charlie Brown"  
RollNo : 104  
ClassName : "IT-3A"

## Display students of specific roll no in a class

The screenshot shows the Compass MongoDB interface. The left sidebar displays connections: cluster0.In3hikv.mongodb.net (selected), cluster0.ldmnyy.mongodb.net, and cluster0.hlwoyba.mongodb.net. Under the selected connection, the 'students' collection is expanded, showing documents for 'admin', 'config', 'inventory', 'local', and 'oplog.rs'. The main area shows a search result for a document with the query: { "RollNo": 102, "ClassName": "IT-3A" }. The document details are: \_id: ObjectId('67ec1a498601ee907e84a950'), Name: "Alice Smith", RollNo: 102, ClassName: "IT-3A". Below the search bar are buttons for ADD DATA, EXPORT DATA, UPDATE, and DELETE.

## Change the roll no of a student

The screenshot shows the Compass MongoDB interface with an open 'Update 1 document' dialog. The dialog is for the 'students' collection in the 'StudentDB' database. The 'Filter' field contains the query: { Name: 'Alice Smith' }. The 'Update' section shows the update operation: { "\$set": { "RollNo": 110 } }. To the right, a 'Preview' window shows the resulting document: \_id: ObjectId('67ec1a498601ee907e84a950'), Name: "Alice Smith", RollNo: 110, ClassName: "IT-3A". At the bottom are 'Save' 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', 'local', and 'products'. Under 'StudentDB', the 'students' collection is selected. The main panel shows the 'Documents' tab with 5 documents. A query selector at the top has the condition: { "Name": "Alice Smith" }. Below it, there are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The results pane shows one document:

```
_id: ObjectId('67ec1a498601ee907e84a950')
Name : "Alice Smith"
RollNo : 102
ClassName : "IT-3A"
```

At the bottom left, a message says '1 document has been updated.' and there is a 'REFRESH' button.

## Delete entries of particular student

The screenshot shows the Compass MongoDB interface with a delete confirmation dialog box overlaid. The dialog title is 'Delete 1 document' with a warning icon. It specifies the collection 'StudentDB.students' and the filter '{ "Name": "Bob Johnson" }'. The preview section shows the same document as in the previous screenshot:

```
_id: ObjectId('67ec1a498601ee907e84a951')
Name : "Bob Johnson"
RollNo : 103
ClassName : "IT-3B"
```

At the bottom of the dialog are 'Cancel' and 'Delete 1 document' buttons. The background shows the same connection tree and document list as the first screenshot.

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' });
  }
});

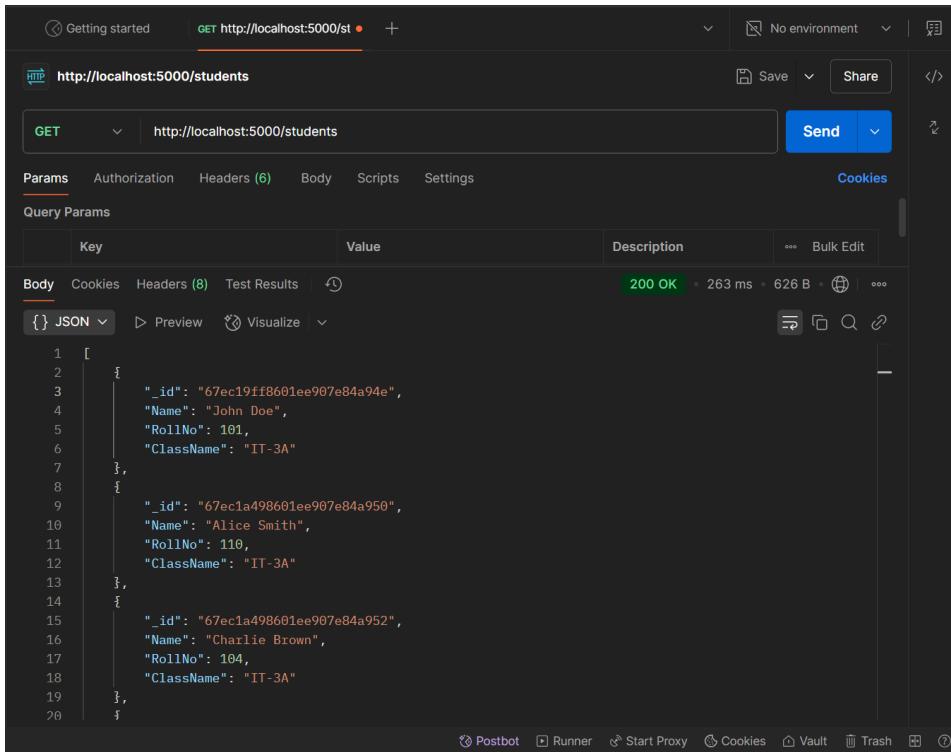
// Start Server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

```

Retrieve a list of all students.

The screenshot shows the Compass MongoDB interface. On the left, the 'Connections' sidebar lists three connections: 'cluster0.ln3hikv.mongodb.net', 'cluster0.dmmyny.mongodb.net', and 'cluster0.hiwayba.mongodb.net'. Under 'cluster0.ln3hikv.mongodb.net', the 'StudentDB' database is selected, and its 'students' collection is shown. The 'Documents' tab is active, displaying four documents. Each document is represented by a card with its \_id, Name, RollNo, and ClassName fields. The documents are:

- `_id: ObjectId('67ec19ff8601ee907e84a94e')`  
Name : "John Doe"  
RollNo : 101  
ClassName : "IT-3A"
- `_id: ObjectId('67eca498601ee907e84a950')`  
Name : "Alice Smith"  
RollNo : 110  
ClassName : "IT-3A"
- `_id: ObjectId('67eca498601ee907e84a952')`  
Name : "Charlie Brown"  
RollNo : 104  
ClassName : "IT-3A"
- `_id: ObjectId('67eca498601ee907e84a953')`  
Name : "David Miller"  
RollNo : 105  
ClassName : "IT-3B"



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

GET http://localhost:5000/students

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

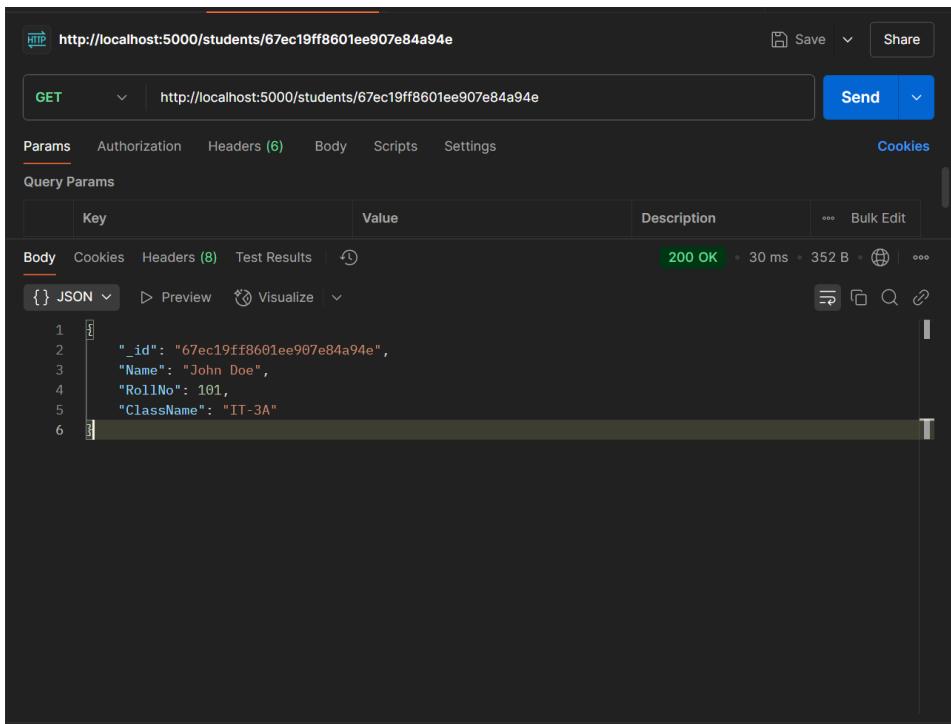
Key	Value	Description	...	Bulk Edit
Body	Cookies Headers (8) Test Results	200 OK	263 ms	626 B

{ } JSON ▾ ▷ Preview ⚡ Visualize

```
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

Params Authorization Headers (6) Body Scripts Settings Cookies

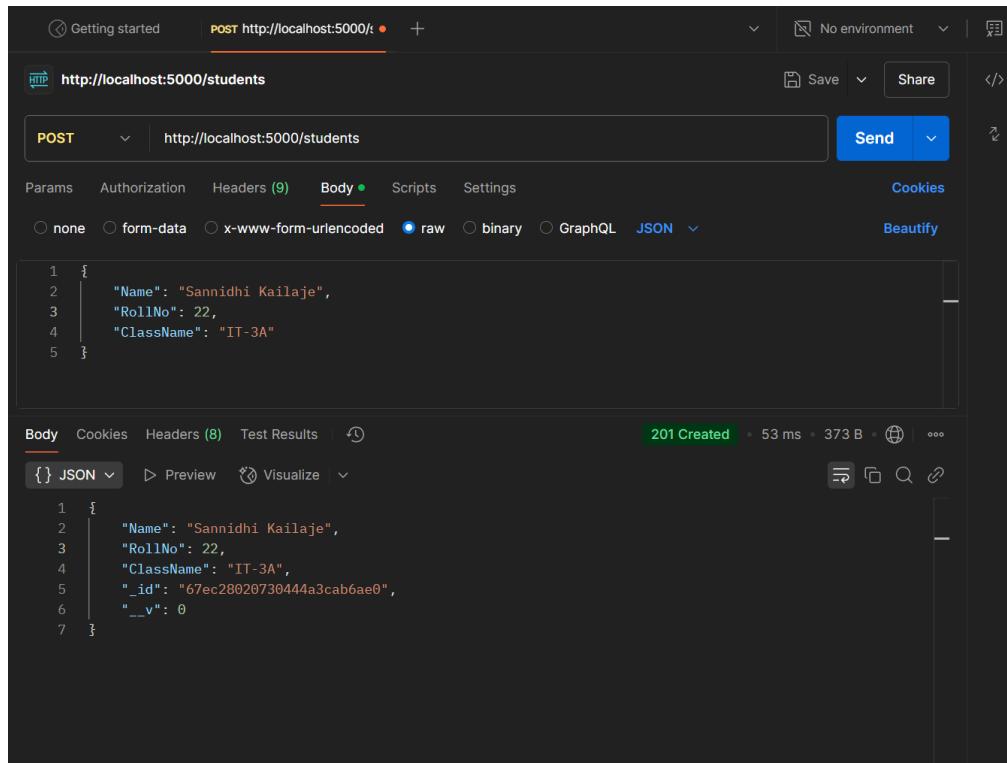
Query Params

Key	Value	Description	...	Bulk Edit
Body	Cookies Headers (8) Test Results	200 OK	30 ms	352 B

{ } JSON ▾ ▷ Preview ⚡ Visualize

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

Add a new student to the database.

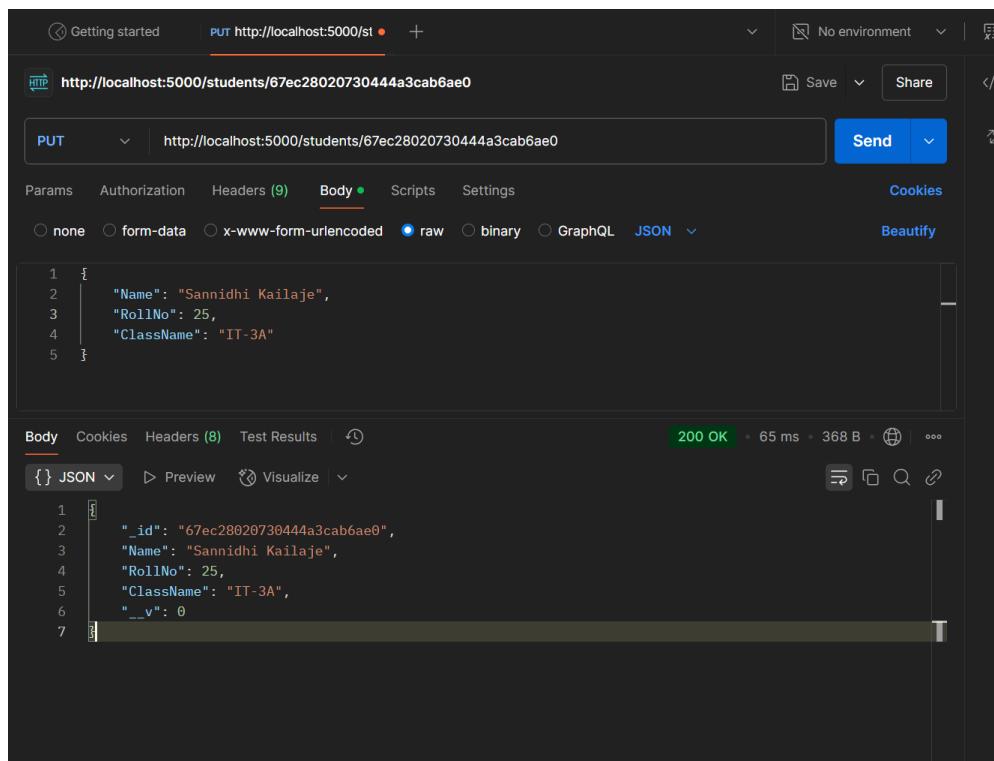


The screenshot shows the Postman interface with a POST request to `http://localhost:5000/students`. The request body contains the following JSON:

```
1 {  
2   "Name": "Sannidhi Kailaje",  
3   "RollNo": 22,  
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 'No environment'. Below it, the URL 'http://localhost:5000/students/67ec19ff8601ee907e84a94e' is entered in the address bar, with a 'DELETE' method selected. To the right are 'Save' and 'Share' buttons. The main workspace has tabs for 'Params', 'Authorization', 'Headers (9)', 'Body' (which is currently selected), 'Scripts', and 'Settings'. Under 'Body', the 'raw' tab is chosen, and the JSON response is displayed:

```
1 {  
2 |   "message": "Student Deleted"  
3 }
```

At the bottom of the workspace, status information is shown: '200 OK', '30 ms', '296 B', and three small icons.