

## Experiment – 7: MongoDB

|                 |                        |
|-----------------|------------------------|
| Name of Student | Snehal Anilkumar Patil |
| Class Roll No   | D15A / 38              |
| 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. A modal window titled 'Create Database' is open in the center. In the 'Database Name' field, 'StudentDB' is typed. In the 'Collection Name' field, 'student' is typed. There is a checkbox for 'Time-Series' which is unchecked. 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 with the 'StudentDB' database selected. A modal window titled 'Insert Document' is open, prompting to insert into the 'students' collection. The document being inserted is a JSON object:

```
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 'Connections' sidebar lists three connections: 'cluster0.In3hikv.mongodb.net', 'cluster0.dmymy.mongodb.net', and 'cluster0.hlwayba.mongodb.net'. Under 'cluster0.In3hikv.mongodb.net', the 'StudentDB' database is selected, and its 'students' collection is shown. The 'Documents' tab is active, displaying one document. The document details are as follows:

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

Below the document, there 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 background shows the same MongoDB interface as the first screenshot, with the 'students' collection now containing multiple documents.

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 right panel shows the 'students' collection with five documents listed:

- `_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

Buttons at the top include 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. A toolbar at the bottom includes '25', '1 - 5 of 5', and navigation icons.

## Display student for a particular class

The screenshot shows the Compass MongoDB interface with a query filter applied: `{"ClassName": "IT-3A"}`. The results show 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"

The interface is identical to the first screenshot, with the same sidebar and toolbar.

## Display students of specific roll no in a class

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays 'cluster0.In3hikv.mongodb.net' with 'StudentDB' selected, which contains the 'students' collection. 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 target collection is 'StudentDB.students'. The 'Filter' field contains { Name: 'Alice Smith' }. The 'Update' field contains the command: { "\$set": { "RollNo": 110 } }. A preview window shows the updated 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'. A preview of a document is shown: '\_id: ObjectId('67ec1a498601ee907e84a950')', 'Name : "Alice Smith"', 'RollNo : 102', and 'ClassName : "IT-3A"'. At the bottom right of the main panel are buttons for 'Generate query', 'Explain', 'Reset', 'Find', 'Options', and a refresh icon. A message bar at the bottom left says '1 document has been updated.' and a 'REFRESH' button is available.

## 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" }'. A preview section shows a single document: '\_id: ObjectId('67ec1a498601ee907e84a951')', 'Name : "Bob Johnson"', 'RollNo : 103', and '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 previous screenshot, with a message bar at the bottom left indicating '1 document has been updated.' and a 'REFRESH' button.

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}`));

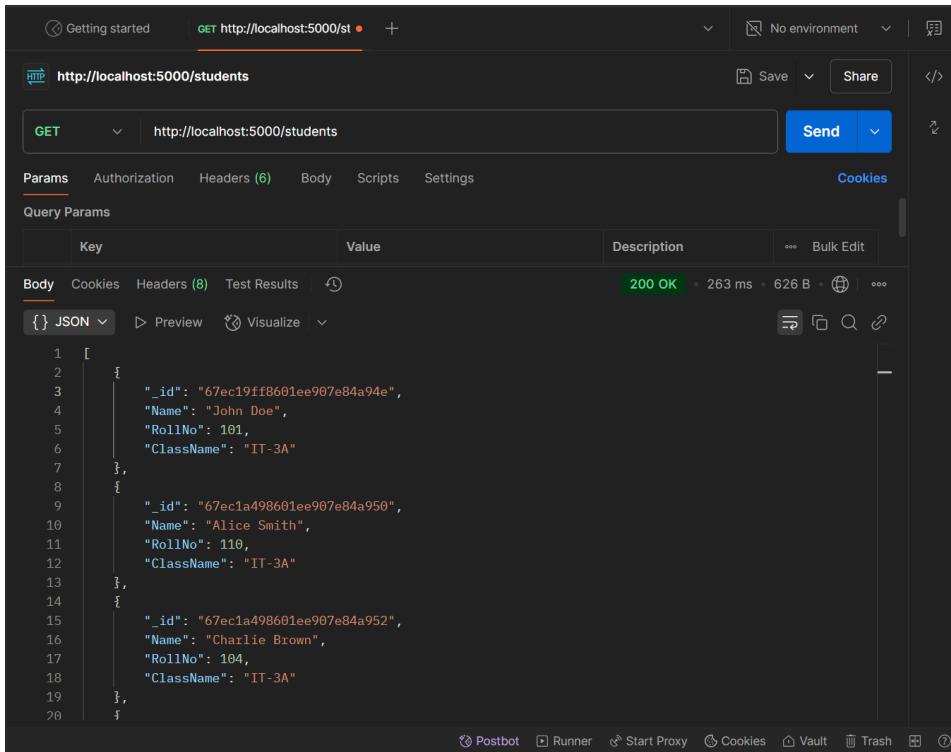
```

Retrieve a list of all students.

The screenshot shows the Compass MongoDB interface. On the left, the connection tree displays 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, roll number, and class name. The documents are:

- Document 1:** \_id: ObjectId('67ec19ff8601ee907e84a94e'), Name: "John Doe", RollNo: 101, ClassName: "IT-3A"
- Document 2:** \_id: ObjectId('67eca498601ee907e84a950'), Name: "Alice Smith", RollNo: 110, ClassName: "IT-3A"
- Document 3:** \_id: ObjectId('67eca498601ee907e84a952'), Name: "Charlie Brown", RollNo: 104, ClassName: "IT-3A"
- Document 4:** \_id: ObjectId('67eca498601ee907e84a953'), Name: "David Miller", RollNo: 105, ClassName: "IT-3B"

At the top right, there are buttons for 'Generate query', 'Explain', 'Reset', 'Find', and 'Options'. Below the documents, a pagination bar indicates '1 - 4 of 4'.



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

GET http://localhost:5000/students

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

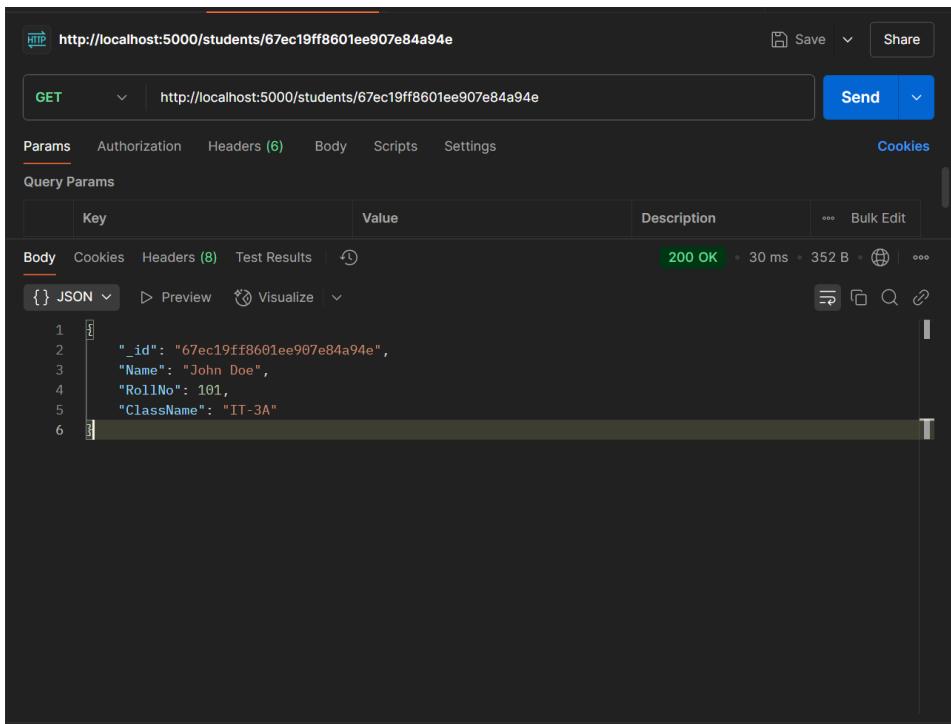
| Key  | Value                            | Description | ...    | Bulk Edit |
|------|----------------------------------|-------------|--------|-----------|
| Body | Cookies Headers (8) Test Results | 200 OK      | 263 ms | 626 B     |

{ } 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 application interface. At the top, there are tabs for 'GET New' (red), 'POST New' (red), 'GET Untitled' (green), and 'POST New' (red). The current tab is 'POST New'. To the right, there's a dropdown for 'No environment' and a 'Save' button. Below the tabs, the URL 'http://localhost:5000/students' is entered under a 'POST' method. On the right, there are 'Save' and 'Share' buttons. Under the URL input, there are tabs for 'Params', 'Auth', 'Headers (8)', 'Body' (highlighted in green), 'Scripts', and 'Settings'. The 'Body' tab has a 'raw' dropdown set to 'JSON'. The JSON payload is:

```
1 {
2   "Name": "Snehal patil",
3   "RollNo": 38,
4   "ClassName": "IT-3A"
5 }
```

The screenshot shows the Postman interface after sending the POST request. The status bar at the top indicates 'POST http://localhost:5000/students' and '201 Created'. The main area shows the same setup as the first screenshot, but the JSON payload is now part of a larger response object. The 'Body' tab is selected, showing the raw JSON response:

```
1 {
2   "Name": "Sannidhi Kailaje",
3   "RollNo": 22,
4   "ClassName": "IT-3A"
5 }
```

Below the JSON, there are tabs for 'Body' (selected), 'Cookies', 'Headers (8)', and 'Test Results'. The 'Test Results' tab shows a green '201 Created' status with metrics: 53 ms, 373 B, and a link icon. At the bottom, there are buttons for 'JSON' (selected), 'Preview', 'Visualize', and a dropdown.

The screenshot shows the Postman interface with a successful API call. The status bar at the top right indicates "201 Created" with a response time of "53 ms" and a body size of "373 B". The "Body" tab is selected, displaying a JSON response object with fields: Name, RollNo, ClassName, \_id, and \_\_v. The "Headers" tab shows 8 items, and the "Test Results" tab is visible.

```
1 {  
2   "Name": "Sannidhi Kailaje",  
3   "RollNo": 22,  
4   "ClassName": "IT-3A",  
5   "_id": "67ec28020730444a3cab6ae0",  
6   "__v": 0  
7 }
```

Update details of an existing student by ID.

The screenshot shows the Postman interface with a successful API call. The status bar at the top right indicates "200 OK" with a response time of "65 ms" and a body size of "368 B". The "Body" tab is selected, displaying a JSON response object with fields: \_id, Name, RollNo, and ClassName. The "Headers" tab shows 8 items, and the "Test Results" tab is visible.

```
1 {  
2   "_id": "67ec28020730444a3cab6ae0",  
3   "Name": "Sannidhi Kailaje",  
4   "RollNo": 25,  
5   "ClassName": "IT-3A"  
6 }  
7
```

Delete a student from the database by ID.

The screenshot shows the Postman application interface. At the top, there's a header bar with tabs for "Getting started", "DEL http://localhost:5000/students/67ec19ff8601ee907e84a94e", and a "+" button. To the right of the URL is a dropdown for "No environment". Below the header, the main workspace shows a request card with "DELETE" selected, the URL "http://localhost:5000/students/67ec19ff8601ee907e84a94e", and a "Send" button. The "Body" tab is active, showing the raw JSON response: 

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

 Above the body, there are tabs for "Params", "Authorization", "Headers (9)", "Body", "Scripts", and "Settings". The "Body" tab has sub-options: "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected), "binary", "GraphQL", and "JSON". To the right of the body tabs are "Cookies" and "Beautify" buttons. Below the body tabs, there are buttons for "JSON", "Preview", "Visualize", and "Copy". On the far right, there are icons for "Save", "Share", and other actions. At the bottom of the workspace, there's a status bar with "200 OK", "30 ms", "296 B", and three dots.