EXPERIMENT – 9

Aim:

To establish a connection between MongoDB and Node.js and implement CRUD operations on a student database using both local and cloud MongoDB.

Description:

MongoDB can be deployed in two primary ways: locally, where the database runs on a user's own machine, or in the cloud, where it is hosted on a managed platform like MongoDB Atlas.

Local MongoDB:

Local MongoDB refers to a self-hosted database that runs on a user's system. It requires MongoDB installation and runs using mongod, the MongoDB server process. The database files are stored on the local machine, making it suitable for development, testing, and small-scale applications.

Cloud MongoDB (MongoDB Atlas):

Cloud MongoDB, such as MongoDB Atlas, is a managed database service hosted in the cloud. It allows users to store, access, and manage data remotely with built-in scalability, security, and backups. It provides a connection string to link with applications like Node.js.

Feature	**Local MongoDB**	**Cloud MongoDB**
Connection String	mongodb://localhost:27017	mongodb+srv://<username>:<password>@<cluster>.mongodb.net
Authentication	No authentication needed by default	Requires username & password
Hosting	Runs on your machine	Hosted by MongoDB Atlas
Access	Limited to your system or LAN	Accessible from anywhere (IP restrictions apply)

Both **local and cloud MongoDB** serve different purposes: **Local MongoDB** is great for development and testing, while **MongoDB Atlas** is ideal for production environments and scalable applications.

Programs:

1. Create a student cluster in MongoDB Atlas

MongoDB Atlas is a cloud-based, fully managed database service that allows users to deploy and manage MongoDB databases with ease. Creating a **student cluster** in MongoDB Atlas provides a **free-tier** cloud database that can be used for development, learning, and small-scale applications.

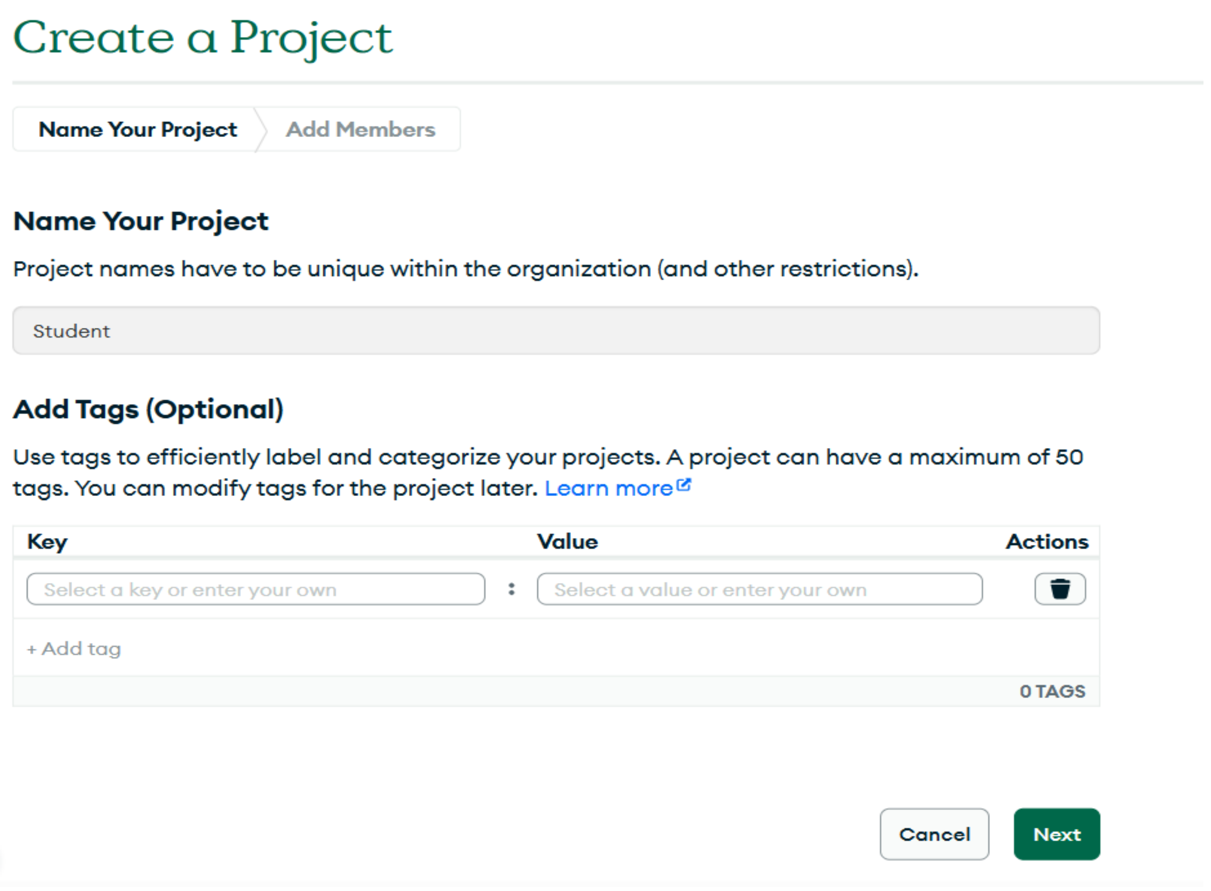
Steps to Create a Student Cluster in MongoDB Atlas

Step 1: Create a MongoDB Atlas Account

1. Visit **MongoDB Atlas**.
2. Click **"Sign Up"** (or log in if you already have an account).
3. Sign up using **email, Google, or GitHub**.
4. Once signed in, you will be redirected to the **Atlas dashboard**.

Step 2: Create a New Project

1. In the **Atlas dashboard**, click on **"New Project"** (found in the top-left corner).
2. Enter a **Project Name** (e.g., "StudentDB").
3. Click **"Next"** and then **"Create Project"**.


Create a Project

Name Your Project | Add Members

Name Your Project

Project names have to be unique within the organization (and other restrictions).

Student

Add Tags (Optional)

Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. Learn more

Key	Value	Actions
Select a key or enter your own	Select a value or enter your own	

+ Add tag

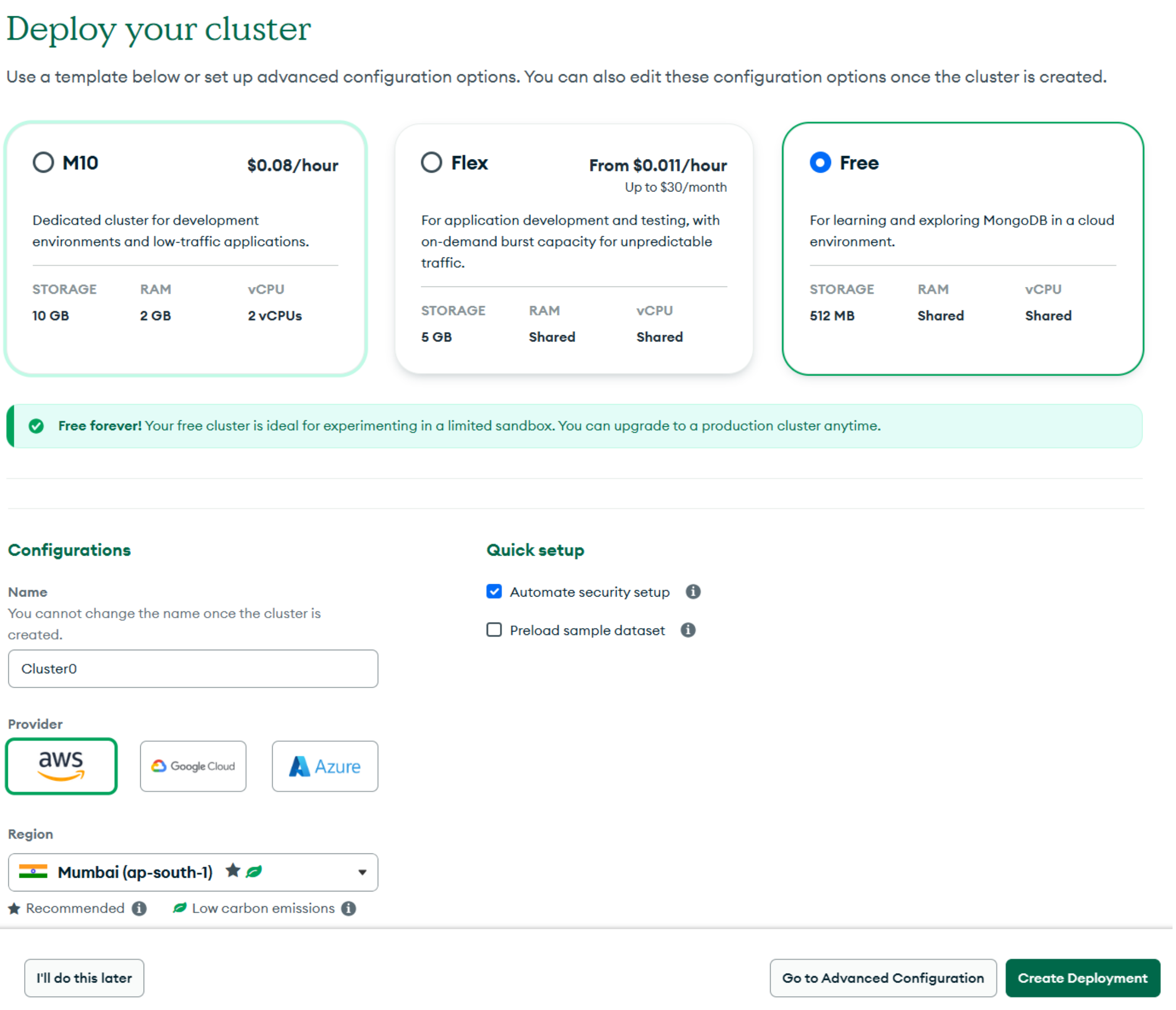
0 TAGS

Cancel Next

Fig 1: creating new project

Step 3: Create a Cluster

1. Inside your newly created project, click **"Create a Cluster"**.
2. Select **"Shared Cluster"** (free tier).
3. Choose a **Cloud Provider & Region** (e.g., AWS, Google Cloud, or Azure).
4. Under **Cluster Tier**, choose **M0 (Free Tier)**.
5. Give your cluster a **name**
6. Click **"Create Cluster"**



Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

	M10	Flex	[✓ Free]
Price	$0.08/hour	From $0.011/hour Up to $30/month	
Description	Dedicated cluster for development environments and low-traffic applications.	For application development and testing, with on-demand burst capacity for unpredictable traffic.	For learning and exploring MongoDB in a cloud environment.
STORAGE	10 GB	5 GB	512 MB
RAM	2 GB	Shared	Shared
vCPU	2 vCPUs	Shared	Shared

Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Configurations

Name
You cannot change the name once the cluster is created.
Cluster0

Provider: [✓ aws] Google Cloud Azure

Region: Mumbai (ap-south-1)
★ Recommended · Low carbon emissions

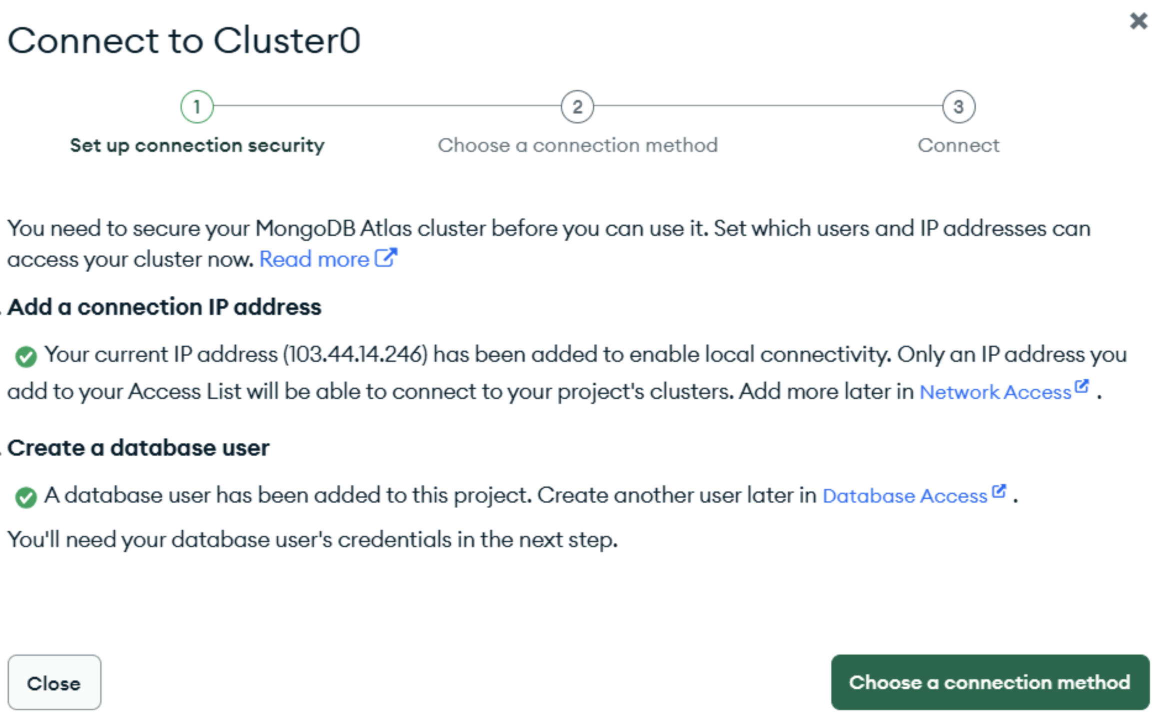
Quick setup
- [x] Automate security setup
- [ ] Preload sample dataset

Fig 2: choosing cluster details

Step 4: Create a Database User and Configure Network Access

1. Click **"Add New Database User"**.
2. Set a **Username and Password** (save these for later use).
3. Click **"Add User"**.
4. Click **"Add IP Address"**.
5. Select **"Allow Access From Anywhere"** (for easy access during development).

6. Click **"Confirm"**.


Connect to Cluster0

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

1. Add a connection IP address

✓ Your current IP address (103.44.14.246) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in Network Access .

2. Create a database user

✓ A database user has been added to this project. Create another user later in Database Access .

You'll need your database user's credentials in the next step.

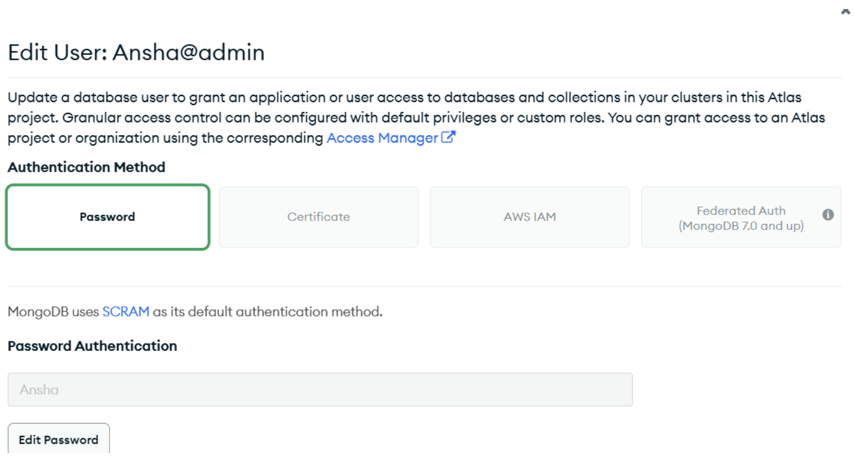
Fig 3: user and network configuration

Step 6: Connect the Cluster to Node.js

1. Once your cluster is ready, go to **"Database"** → **"Connect"**.
2. Choose **"Drivers"** and select **Node.js**.
3. Copy the **connection string** (it will look like this):

mongodb+srv://<username>:<password>@studentcluster.mongodb.net/?retryWrites=true&w=majority

4. Replace <username> and <password> with the database user credentials you created earlier.
5. Use this connection string in your **Node.js application** to interact with the database.


Edit User: Ansha@admin

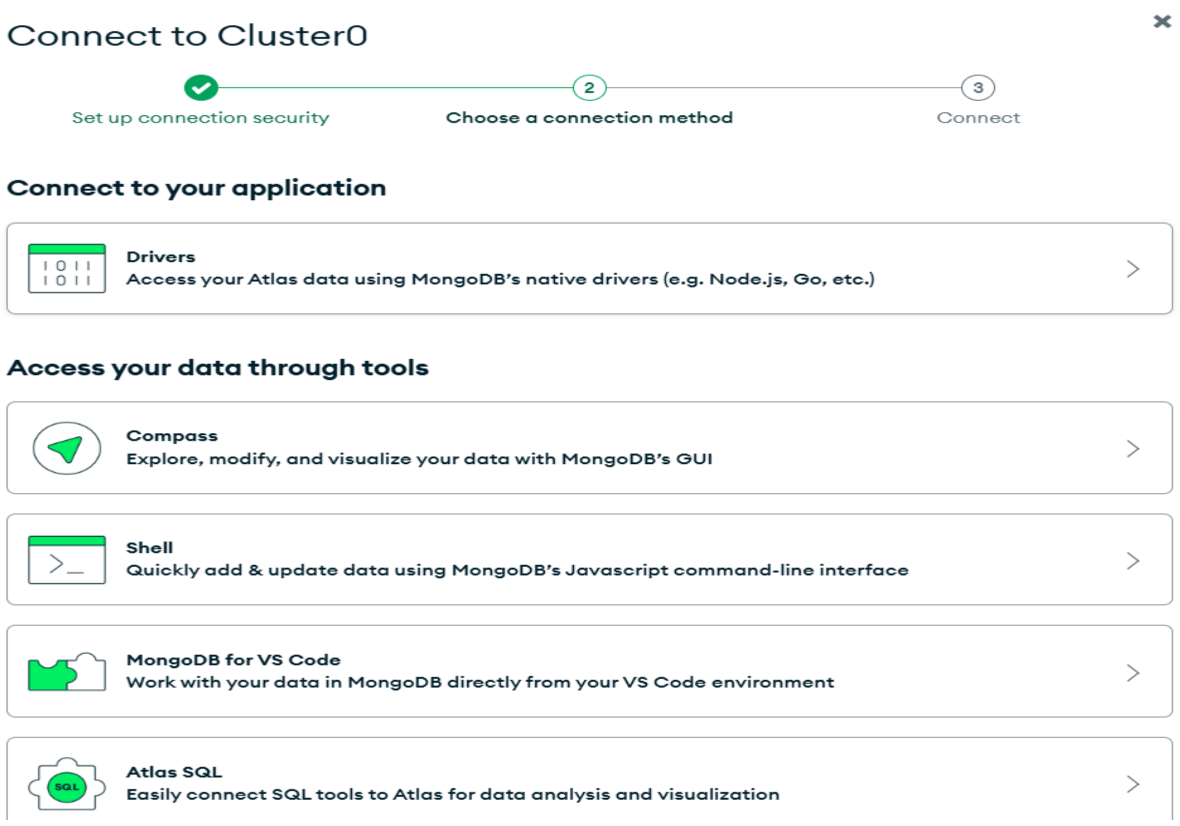
Update a database user to grant an application or user access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding Access Manager

Authentication Method

MongoDB uses SCRAM as its default authentication method.

Password Authentication

Ansha



Connect to Cluster0

Set up connection security — 2 Choose a connection method — 3 Connect

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

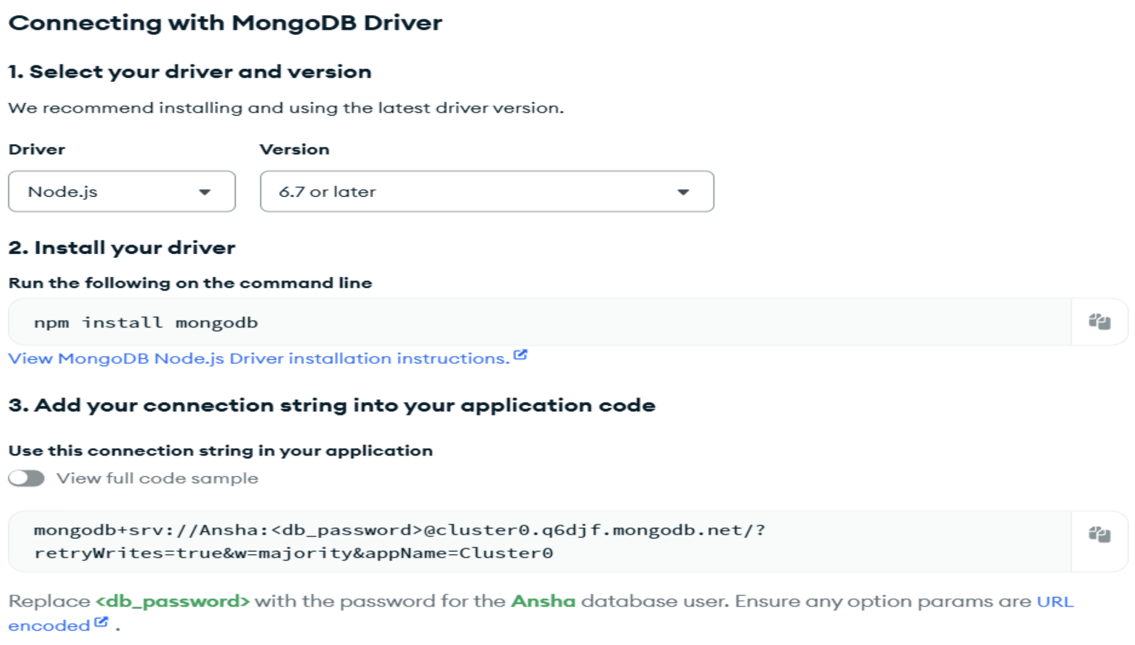
Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization

Fig 4: connecting cluster



Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Node.js
Version: 6.7 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

3. Add your connection string into your application code

Use this connection string in your application

View full code sample

```
mongodb+srv://Ansha:<db_password>@cluster0.q6djf.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0
```

Replace <db_password> with the password for the Ansha database user. Ensure any option params are URL encoded .

Fig 5: getting node connection string

2. Implement CRUD operations on student database using both local and cloud MongoDB, with Node'js.

1. Local MongoDB

Local MongoDB refers to a database setup on a personal computer or a dedicated server. The database runs directly on the system, giving developers full control over its configuration, security, and performance. Local MongoDB is useful for:

- Offline development and testing
- Faster read/write operations since data is stored locally
- No internet dependency

To use MongoDB locally, developers need to install the MongoDB software, start the database server, and connect their applications to it.

In a local MongoDB setup, the database server runs on your machine. By default, MongoDB listens on **port 27017**.

Connection String:

mongodb://localhost:27017/studentDB

localhost → MongoDB runs on the local machine

27017 → Default MongoDB port

studentDB → The database name

2. Cloud MongoDB (MongoDB Atlas)

MongoDB Atlas is a cloud-based managed database service that eliminates the need for manual setup and maintenance. It provides benefits such as:

- Scalability: Easily handle large amounts of data
- Automatic backups and security features
- Accessibility: Can be accessed from anywhere with an internet connection

To use MongoDB Atlas, developers create an online cluster, add authorized users, and connect their applications using a secure connection string.

Connection String:

mongodb+srv://<username>:<password>@<cluster-name>.mongodb.net/<database-name>?retryWrites=true&w=majority

Replace <username> and <password> with your database user credentials.

Replace <cluster-name> with your cluster’s name.

Replace <database-name> with the actual database name.

Program:

//App.jsx

```
import React, { useState, useEffect } from "react";
import axios from "axios";
```

```
export default function App() {
  const [students, setStudents] = useState([]);
  const [addForm, setAddForm] = useState({ roll: "", name: "", age: "", course: "" });
  const [updateForm, setUpdateForm] = useState({ roll: "", name: "", age: "", course: "" });
  const [searchRoll, setSearchRoll] = useState("");
  const [deleteRoll, setDeleteRoll] = useState("");
  const [searchedStudent, setSearchedStudent] = useState(null);
  const [addMessage, setAddMessage] = useState("");
  const [searchMessage, setSearchMessage] = useState("");
  const [updateMessage, setUpdateMessage] = useState("");
  const [deleteMessage, setDeleteMessage] = useState("");
  useEffect(() => { fetchStudents(); }, []);

  const fetchStudents = async () => {
    try {
      const res = await axios.get("http://localhost:5000/students");
      setStudents(res.data);
    } catch (err) {
      console.error(err);
    }
  };

  const handleAddChange = (e) => {
    setAddForm({ ...addForm, [e.target.name]: e.target.value });
  };

  const handleUpdateChange = (e) => {
    setUpdateForm({ ...updateForm, [e.target.name]: e.target.value });
  };

  const handleAddSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post("http://localhost:5000/students", addForm);
      setAddMessage("Student added successfully");
      fetchStudents();
      setAddForm({ roll: "", name: "", age: "", course: "" });
    } catch (err) {
      setAddMessage("Error adding student");
      console.error(err);
    }
  };

  const handleSearch = async () => {
    try {
      const res = await axios.get(`http://localhost:5000/students/${searchRoll}`);
      if (res.data) {
        setSearchedStudent(res.data);
        setSearchMessage("");
      } else {
        setSearchedStudent(null);
        setSearchMessage("Student not found");
      }
    } catch (err) {
      setSearchedStudent(null);
      setSearchMessage("Student not found");
      console.error(err);
    }
  };

  const handleUpdate = async () => {
    try {
      await axios.put(`http://localhost:5000/students/${updateForm.roll}`, updateForm);
      setUpdateMessage("Student updated successfully");
      fetchStudents();
    } catch (err) {
      setUpdateMessage("Error updating student");
      console.error(err);
    }
  };

  const handleDelete = async () => {
    try {
      await axios.delete(`http://localhost:5000/students/${deleteRoll}`);
      setDeleteMessage("Student deleted successfully");
      fetchStudents();
    } catch (err) {
      setDeleteMessage("Error deleting student");
      console.error(err);
    }
  };

  return (
    <div className="p-6 max-w-lg mx-auto space-y-4">
      <h1 className="text-2xl font-bold">Student Management</h1>

      {/* Add Student */}
      <div className="p-4 border rounded">
        <h2 className="text-lg font-bold">Add Student</h2>
        {addMessage && <p className="text-green-600">{addMessage}</p>}
        <form onSubmit={handleAddSubmit} className="space-y-2">
          <input name="roll" value={addForm.roll} onChange={handleAddChange}
placeholder="Roll" className="p-2 border w-full" required />
          <input name="name" value={addForm.name} onChange={handleAddChange}
placeholder="Name" className="p-2 border w-full" required />
          <input name="age" value={addForm.age} onChange={handleAddChange}
placeholder="Age" className="p-2 border w-full" required />
          <input name="course" value={addForm.course} onChange={handleAddChange}
placeholder="Course" className="p-2 border w-full" required />
          <button type="submit" className="p-2 bg-blue-500 text-white w-full">Add
Student</button>
        </form>
      </div>

      {/* Search Student */}
      <div className="p-4 border rounded">
        <h2 className="text-lg font-bold mb-2">Search Student</h2>
        <input
          value={searchRoll}
          onChange={(e) => setSearchRoll(e.target.value)}
          placeholder="Enter Roll No"
          className="p-2 border w-full mb-2"
        />
        <button onClick={handleSearch} className="p-2 bg-yellow-500 text-white w-full">
          Get Student
        </button>
        {searchMessage && <p className="text-red-600 mt-2">{searchMessage}</p>}
        {searchedStudent && (
          <div className="mt-4 p-4 border rounded shadow">
            <h3 className="text-md font-semibold">Student Details</h3>
            <table className="w-full mt-2 border-collapse border">
              <tbody>
                <tr>
                  <td className="border p-2 font-medium">Name:</td>
                  <td className="border p-2">{searchedStudent.name}</td>
                </tr>
                <tr>
                  <td className="border p-2 font-medium">Roll No:</td>
                  <td className="border p-2">{searchedStudent.roll}</td>
                </tr>
                <tr>
                  <td className="border p-2 font-medium">Course:</td>
                  <td className="border p-2">{searchedStudent.course}</td>
                </tr>
                <tr>
                  <td className="border p-2 font-medium">Age:</td>
                  <td className="border p-2">{searchedStudent.age}</td>
                </tr>
              </tbody>
            </table>
          </div>
        )}
      </div>

      {/* Update Student */}
      <div className="p-4 border rounded">
        <h2 className="text-lg font-bold">Update Student</h2>
        {updateMessage && <p className="text-green-600">{updateMessage}</p>}
        <input name="roll" value={updateForm.roll} onChange={handleUpdateChange}
    placeholder="Roll" className="p-2 border w-full" required />
        <input name="name" value={updateForm.name} onChange={handleUpdateChange}
    placeholder="Name" className="p-2 border w-full" required />
        <input name="age" value={updateForm.age} onChange={handleUpdateChange}
    placeholder="Age" className="p-2 border w-full" required />
        <input name="course" value={updateForm.course} onChange={handleUpdateChange}
    placeholder="Course" className="p-2 border w-full" required />
        <button onClick={handleUpdate} className="p-2 bg-green-500 text-white w-full mt-
    2">Update Student</button>
      </div>

      {/* Delete Student */}
      <div className="p-4 border rounded">
        <h2 className="text-lg font-bold">Delete Student</h2>
        {deleteMessage && <p className="text-green-600">{deleteMessage}</p>}
        <input
          value={deleteRoll}
          onChange={(e) => setDeleteRoll(e.target.value)}
          placeholder="Enter Roll No"
          className="p-2 border w-full"
        />
```

```
      <button onClick={handleDelete} className="p-2 bg-red-500 text-white w-full mt-
2">Delete Student</button>
      </div>
    </div>
  );
}
```

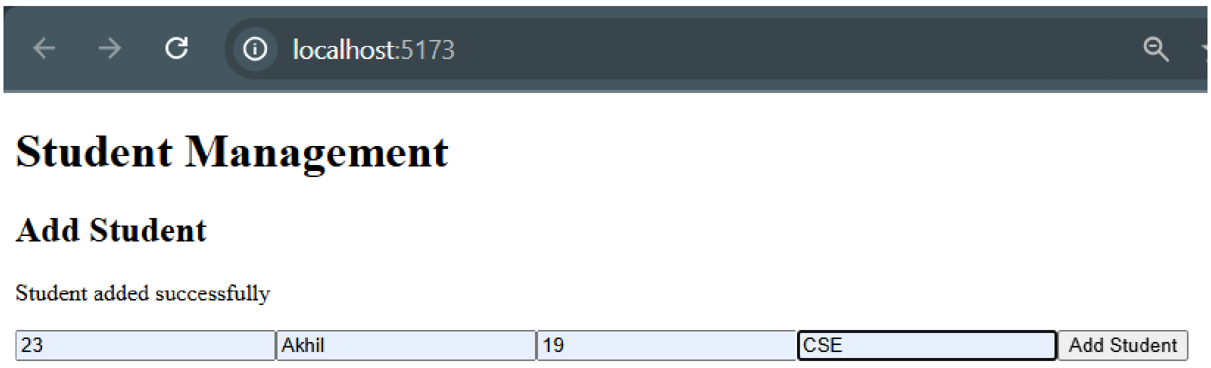
//Server.js

```
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");
const studentRoutes = require("./studentsAPI");

const app = express();
app.use(express.json());

const PORT = process.env.PORT || 5000;

// Toggle between local and cloud MongoDB
const isLocal = process.argv.includes("--local");
const MONGO_URI = isLocal ? process.env.MONGO_LOCAL_URI :
process.env.MONGO_CLOUD_URI;

// Connect to MongoDB
mongoose
  .connect(MONGO_URI)
  .then(() => console.log(`Connected to ${isLocal ? "Local" : "Cloud"} MongoDB`))
  .catch((err) => console.error(err));

app.use("/students", studentRoutes);
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

//StudentAPI.js

```
const express = require("express");
const Student = require("./Student");
const router = express.Router();

// Create a new student
router.post("/", async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    console.log("Student created:", student);
    res.status(201).send(student);
  } catch (err) {
    console.error(err);
    res.status(400).send(err);
  }
});

// Read all students
router.get("/", async (req, res) => {
  try {
    const students = await Student.find();
    console.log("Students retrieved:", students);
    res.status(200).send(students);
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
});

// Read particular student
router.get("/:roll", async (req, res) => {
  try {
    const student = await Student.findOne({ roll: req.params.roll }, req.body, { new: true });
    if (!student) return res.status(404).send({ message: "Student not found" });
    console.log("Student retrieved:", student);
    res.status(200).send(student);
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
});

// Update a student
router.put("/:roll", async (req, res) => {
  try {
    const student = await Student.findOneAndUpdate({ roll: req.params.roll }, req.body, {
new: true });
    if (!student) return res.status(404).send({ message: "Student not found" });

    console.log("Student updated:", student);
    res.status(200).send(student);
  } catch (err) {
    console.error(err);
    res.status(400).send(err);
  }
});

// Delete a student
router.delete("/:roll", async (req, res) => {
  try {
    const student = await Student.findOneAndDelete({ roll: req.params.roll });
    if (!student) return res.status(404).send({ message: "Student not found" });

    console.log(`Student with Roll ${req.params.roll} deleted`);
    res.status(200).send({ message: "Student deleted" });
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
```

```
});

module.exports = router;
```


localhost:5173

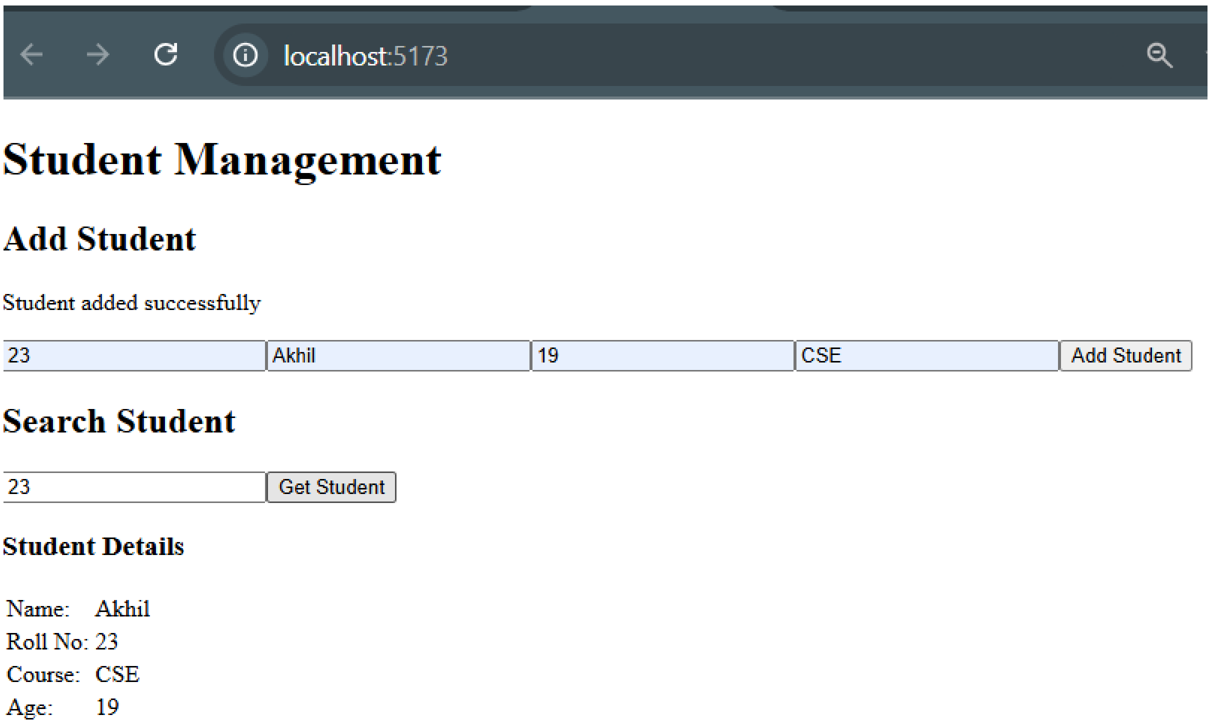
Student Management

Add Student

Student added successfully

23 | Akhil | 19 | CSE | Add Student

Fig 6: post student


localhost:5173

Student Management

Add Student

Student added successfully

23 | Akhil | 19 | CSE | Add Student

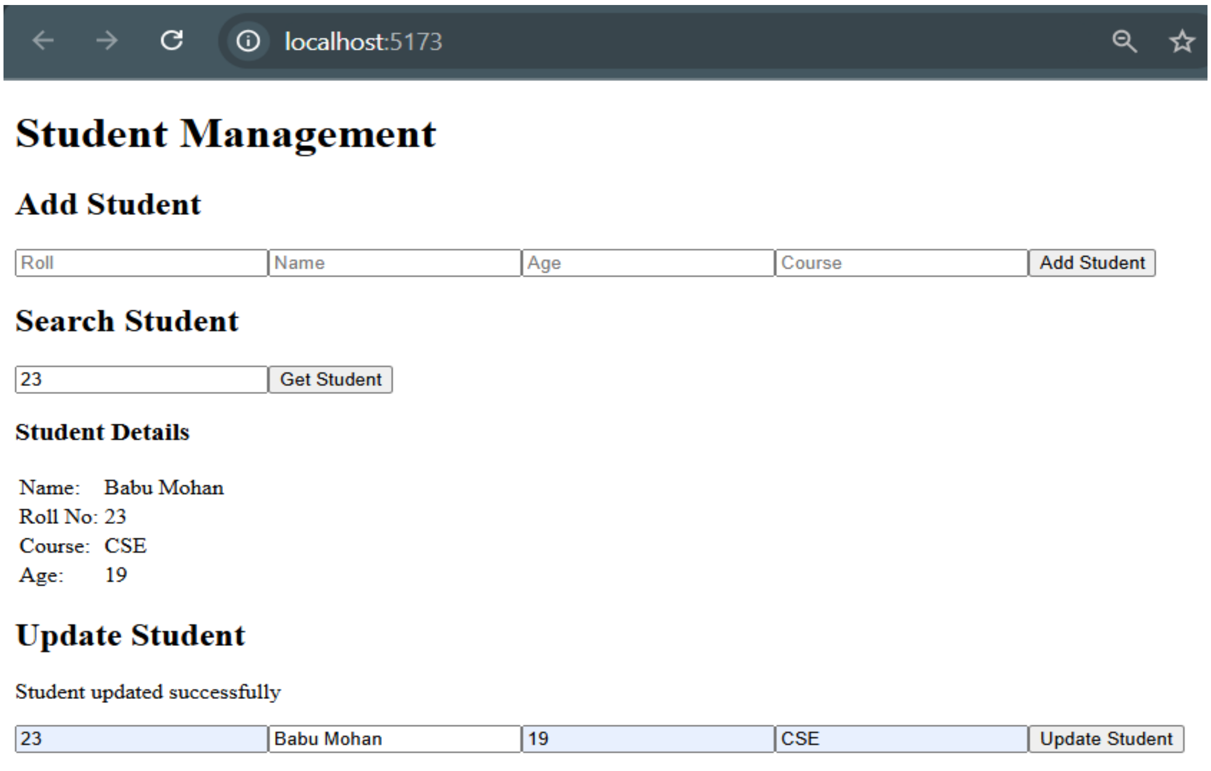
Search Student

23 | Get Student

Student Details

```
Name:    Akhil
Roll No: 23
Course:  CSE
Age:     19
```

Fig 7: get student



Student Management

Add Student

Search Student

23

Student Details

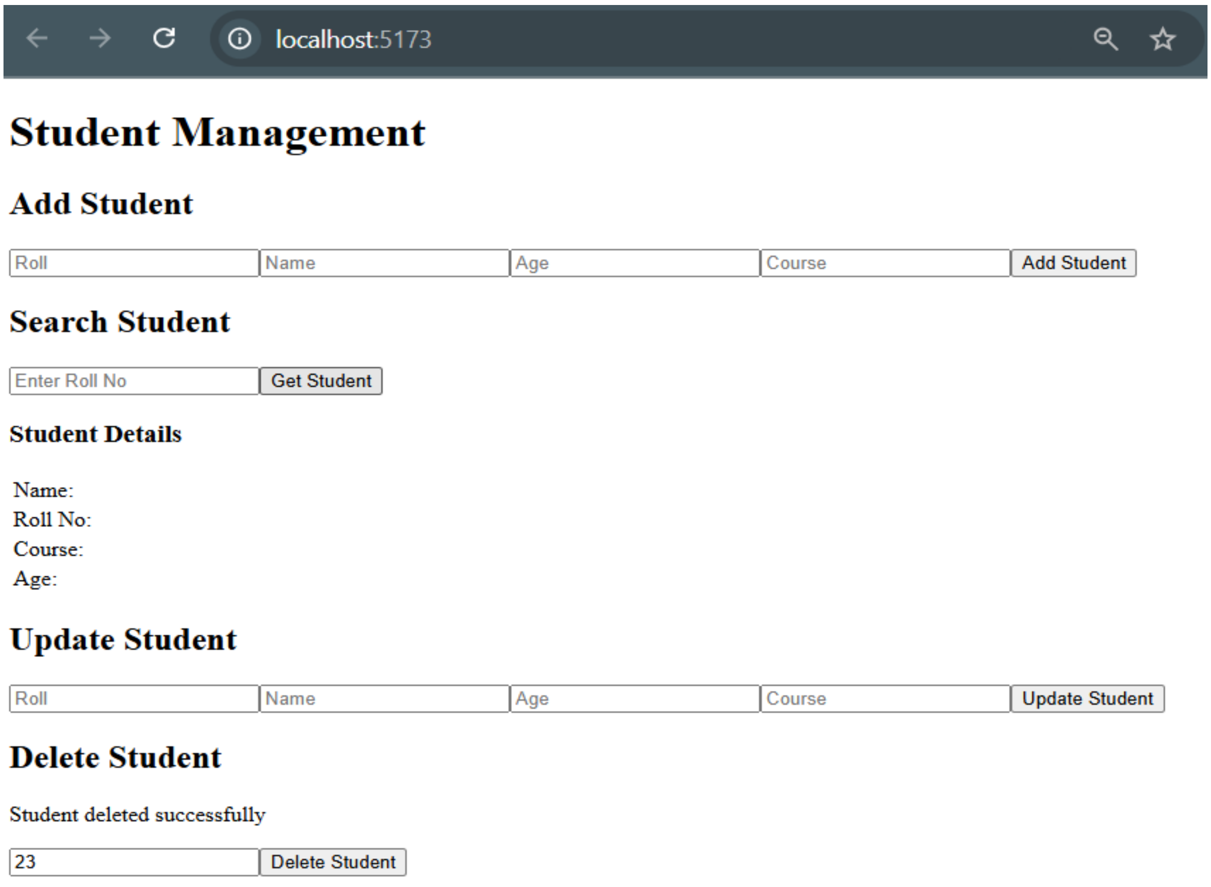
Name: Babu Mohan
Roll No: 23
Course: CSE
Age: 19

Update Student

Student updated successfully

23 | Babu Mohan | 19 | CSE

Fig 8: put student



Student Management

Add Student

Search Student

Student Details

Name:
Roll No:
Course:
Age:

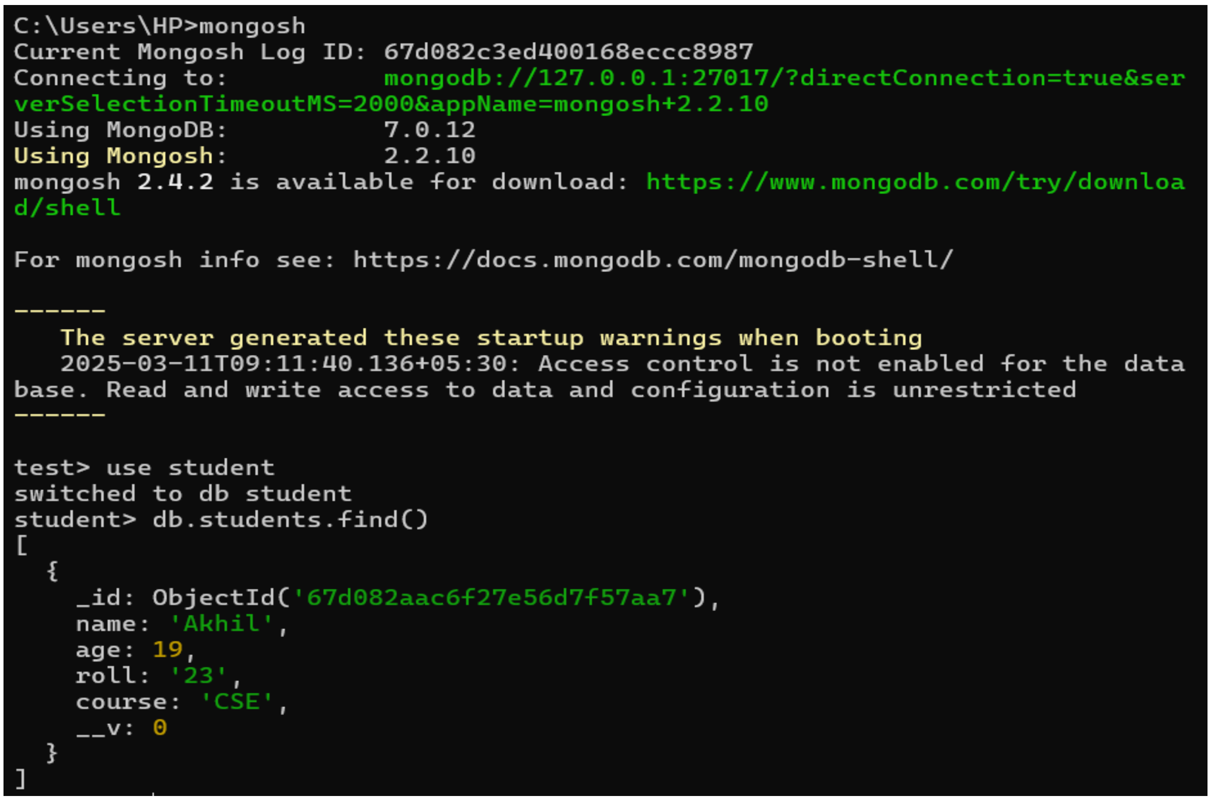
Update Student

Delete Student

Student deleted successfully

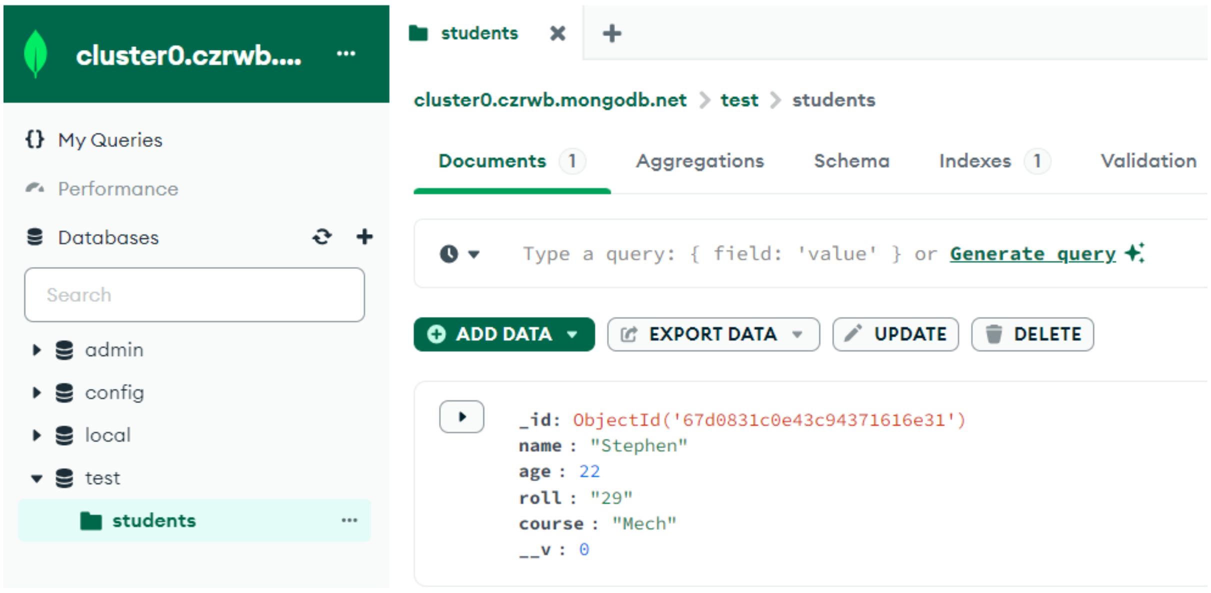
23

Fig 9: delete student



```
C:\Users\HP>mongosh
Current Mongosh Log ID: 67d082c3ed400168eccc8987
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&ser
verSelectionTimeoutMS=2000&appName=mongosh+2.2.10
Using MongoDB:          7.0.12
Using Mongosh:          2.2.10
mongosh 2.4.2 is available for download: https://www.mongodb.com/try/downloa
d/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-11T09:11:40.136+05:30: Access control is not enabled for the data
base. Read and write access to data and configuration is unrestricted
------

test> use student
switched to db student
student> db.students.find()
[
  {
    _id: ObjectId('67d082aac6f27e56d7f57aa7'),
    name: 'Akhil',
    age: 19,
    roll: '23',
    course: 'CSE',
    __v: 0
  }
]
```

Fig 10: data in local MongoDB



```
_id: ObjectId('67d0831c0e43c94371616e31')
name : "Stephen"
age : 22
roll : "29"
course : "Mech"
__v : 0
```

Fig 11: data in cloud MongoDB

Result:

Connection between MongoDB and Node.js using both local and cloud MongoDB is established successfully.