Experiment 7 : MongoDB CRUD Operations using RESTful Endpoints.

Name of Student	**Mohit Patil**
Class Roll No	**D15A_36**
D.O.P.	**13/03/2025**
D.O.S.	**20/03/2025D**
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
   a) Insert one student details
   b) Insert at once multiple student details
   c) Display student for a particular class
   d) Display students of specific roll no in a class
   e) Change the roll no of a student
   f) 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.

Theory:-

Introduction

MongoDB is a NoSQL database that allows for flexible and scalable data storage in a document-oriented format using BSON (Binary JSON). In modern web applications, MongoDB is often integrated with RESTful APIs to enable Create, Read, Update, and Delete (CRUD) operations over HTTP. RESTful services allow applications to interact with a MongoDB database through standardized HTTP methods, facilitating seamless communication between the client and server.

CRUD Operations in MongoDB using RESTful Endpoints

1. Create Operation (POST Method)

The **Create** operation is used to insert new documents into a MongoDB collection. In a RESTful API, the **POST** method is used to send data to the server, which then stores it in the database.

Example Endpoint:

bash
CopyEdi
t
POST /api/products

-

Implementation in Express.js (Node.js) with MongoDB:

javascript
CopyEdit

```
app.post('/api/products', async (req, res) => {    try {

        const newProduct = new Product(req.body);     await
newProduct.save();       res.status(201).json(newProduct);
        } catch (error) {
      res.status(400).json({ error: error.message });
        }
```

});

- **How it Works:**
    - The client sends a JSON request body containing product details.
        - The server receives the request and inserts the new document into the MongoDB collection.
        - If successful, the server responds with the created document and a **201 Created** status.

2. Read Operation (GET Method)

The **Read** operation retrieves data from MongoDB. The **GET** method is used to fetch either a list of documents or a single document by ID.

- **Example Endpoints:**

Fetch all products:

bash
CopyEdi
t
GET /api/products

-

Fetch a specific product by ID:

bash
CopyEdi
t
GET /api/products/:id

-

Implementation:

javascript
CopyEdit

```
app.get('/api/products', async (req, res) => { try {
        const products = await Product.find();        res.json(products);
        } catch (error) {
      res.status(500).json({ error: error.message });
        }
app.get('/api/products/:id', async (req, res) => {
        try {
                        const product = await
Product.findById(req.params.id);
        if (!product) return res.status(404).json({ message: 'Product not found' });
        res.json(product);       } catch (error) {
      res.status(500).json({ error: error.message });
        }
});
```

- **How it Works:**
    - The first endpoint retrieves all products from the database.
        - The second endpoint fetches a specific product by its unique **ObjectId**.
    - If the document is not found, a **404 Not Found** response is returned.

3. Update Operation (PUT/PATCH Method)

The **Update** operation modifies an existing document in MongoDB. RESTful APIs use the **PUT** or **PATCH** method to update records:

- **PUT**: Updates the entire document.
- **PATCH**: Updates only specific fields.

Example Endpoint:

bash
CopyEdi
t
PUT /api/products/:id

Implementation:
javascript
CopyEdit

```
app.put('/api/products/:id', async (req, res) => {        try {
                    const updatedProduct = await
Product.findByIdAndUpdate(req.params.id, req.body, { new: true });
                    if (!updatedProduct) return
res.status(404).json({ message: 'Product not found' });        res.json(updatedProduct);
          } catch (error) {
       res.status(400).json({ error: error.message });
          }
});
```

- **How it Works:**
  - The client sends the updated data in the request body.
  - The server finds the document by ID and updates it.
  - If successful, the server responds with the modified document.

4. Delete Operation (DELETE Method)

The **Delete** operation removes a document from the MongoDB collection. The **DELETE** method is used in RESTful APIs to delete records based on an identifier.

Example Endpoint:

bash

CopyEdi

t

```
DELETE /api/products/:id
```

Implementation:

javascript

CopyEdit

```
app.delete('/api/products/:id', async (req, res) => {    try {

                    const deletedProduct = await
Product.findByIdAndDelete(req.params.id);

                    if (!deletedProduct) return
res.status(404).json({ message: 'Product not found' });        res.json({ message: 'Product deleted
successfully' });          } catch (error) {

       res.status(500).json({ error: error.message });

          }

});
```

- **How it Works:**
    - The client sends a request to delete a document by ID.
        - The server searches for the document and removes it from the collection.
    - If deletion is successful, a confirmation message is sent.

Advantages of Using RESTful APIs with MongoDB

1. **Scalability** – MongoDB's NoSQL architecture combined with REST APIs allows handling large volumes of data.
2. **Flexibility** – JSON-based communication ensures easy integration with front-end applications.

3. **Statelessness** – REST APIs ensure each request is independent, making it easy to deploy and scale.
4. **Cross-Platform Compatibility** – Any client (web, mobile, IoT) can communicate using standard HTTP methods.

Creating Database using mongosh(MongoDB shell) and inserting values

```
test> use IT_Dept_Students
switched to db IT_Dept_Students
IT_Dept_Students> db.students.insertMany([
...     { name: "Aman Gupta", roll_no: 101, class_name: "IT-A" },{ name: "Rajesh Kumar", roll_no: 102, class_name: "IT-A" },
...     { name: "Priya Sharma", roll_no: 103, class_name: "IT-B" },
...     { name: "Vikram Sinha", roll_no: 104, class_name: "IT-A" },
...     { name: "Neha Verma", roll_no: 105, class_name: "IT-B" }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67e9a3f35569ae6183cc8988'),
    '1': ObjectId('67e9a3f35569ae6183cc8989'),
    '2': ObjectId('67e9a3f35569ae6183cc898a'),
    '3': ObjectId('67e9a3f35569ae6183cc898b'),
    '4': ObjectId('67e9a3f35569ae6183cc898c')
  }
}
```

CRUD operations:

```
IT_Dept_Students> db.students.find({ class_name: "IT-A" }).pretty()
[
  {
    _id: ObjectId('67e9a3f35569ae6183cc8988'),
    name: 'Aman Gupta',
    roll_no: 101,
    class_name: 'IT-A'
  },
  {
    _id: ObjectId('67e9a3f35569ae6183cc8989'),
    name: 'Rajesh Kumar',
    roll_no: 102,
    class_name: 'IT-A'
  },
  {
    _id: ObjectId('67e9a3f35569ae6183cc898b'),
    name: 'Vikram Sinha',
    roll_no: 104,
    class_name: 'IT-A'
  }
]
IT_Dept_Students> db.students.find({ roll_no: 104, class_name: "IT-A" }).pretty()
[
  {
    _id: ObjectId('67e9a3f35569ae6183cc898b'),
    name: 'Vikram Sinha',
    roll_no: 104,
    class_name: 'IT-A'
  }
]
IT_Dept_Students> db.students.updateOne({ roll_no: 103 }, { $set: { roll_no: 120 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
IT_Dept_Students> db.students.deleteOne({ roll_no: 105 })
{ acknowledged: true, deletedCount: 1 }
IT_Dept_Students>
```

Code:

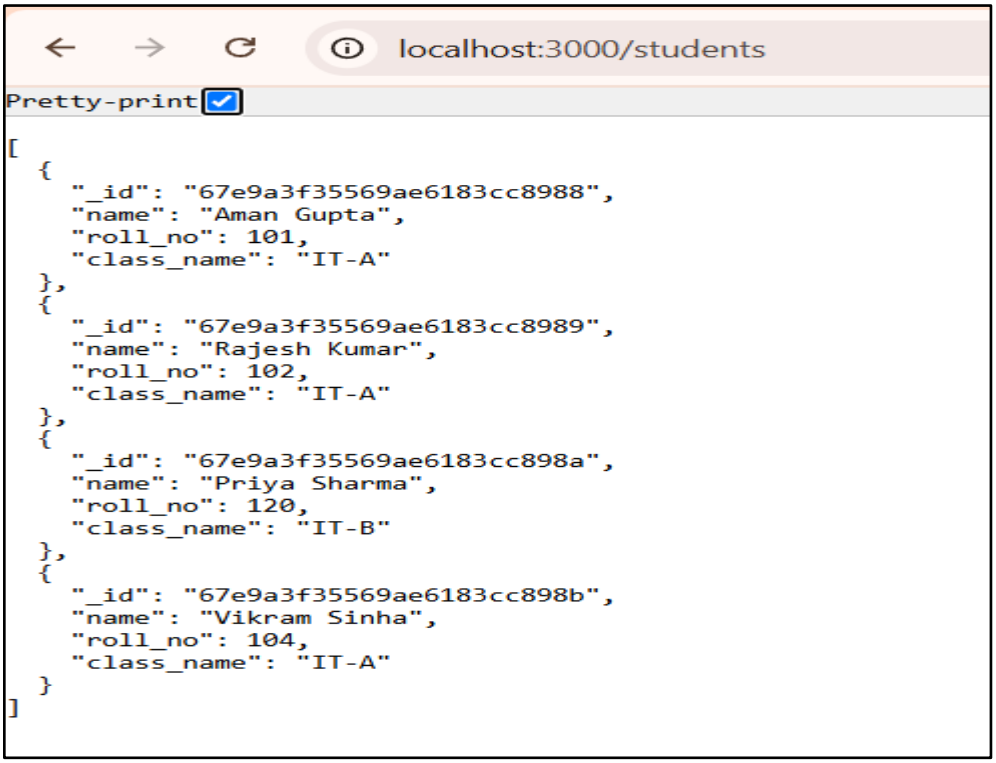
```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");

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

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/IT_Dept_Students", {
}).then(() => console.log("MongoDB Connected"))
  .catch(err => console.log(err));

// Define Student Schema
const studentSchema = new mongoose.Schema({
    name: String,
    roll_no: Number,
```

```
    class_name: String
});

const Student = mongoose.model("Student", studentSchema);

app.get("/students", async (req, res) => {
    const students = await Student.find();
    res.json(students);
});

app.get("/students/:id", async (req, res) => {
    try {
        const student = await Student.findById(req.params.id);
        if (!student) return res.status(404).json({ error: "Student not found" });
        res.json(student);
    } catch (error) {
        res.status(500).json({ error: "Invalid ID format" });
    }
});

app.get("/students/class/:className", async (req, res) => {
    const students = await Student.find({ class_name: req.params.className });
    res.json(students);
});



app.get("/students/roll/:rollNo", async (req, res) => {
    const student = await Student.findOne({ roll_no: req.params.rollNo });
    if (!student) return res.status(404).json({ error: "Student not found" });
    res.json(student);
});

app.post("/students", async (req, res) => {
    const newStudent = new Student(req.body);
    await newStudent.save();
    res.json({ message: "Student Added", student: newStudent });
});

app.put("/students/:id", async (req, res) => {
    const updatedStudent = await Student.findByIdAndUpdate(req.params.id, { roll_no: req.body.roll_no }, {
new: true });
    if (!updatedStudent) return res.status(404).json({ error: "Student not found" });
    res.json({ message: "Roll No Updated", student: updatedStudent });
});

app.delete("/students/:id", async (req, res) => {
    const deletedStudent = await Student.findByIdAndDelete(req.params.id);
    if (!deletedStudent) return res.status(404).json({ error: "Student not found" });
```

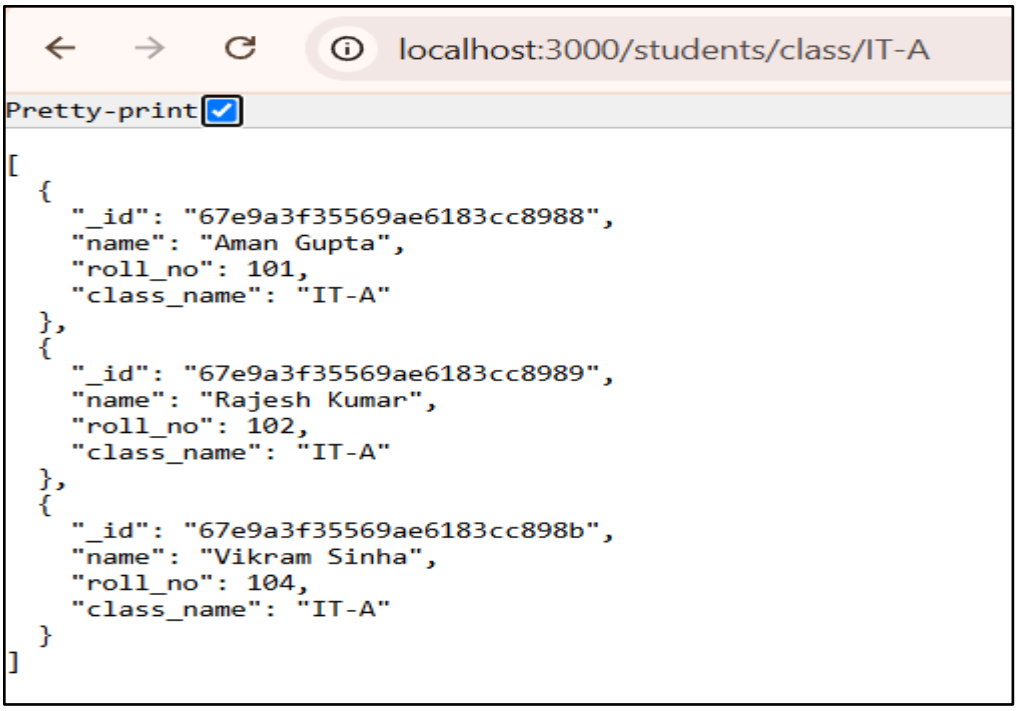
```
    res.json({ message: "Student Deleted" });
});

app.listen(3000, () => console.log("Server running on port 3000"));
```

Output:

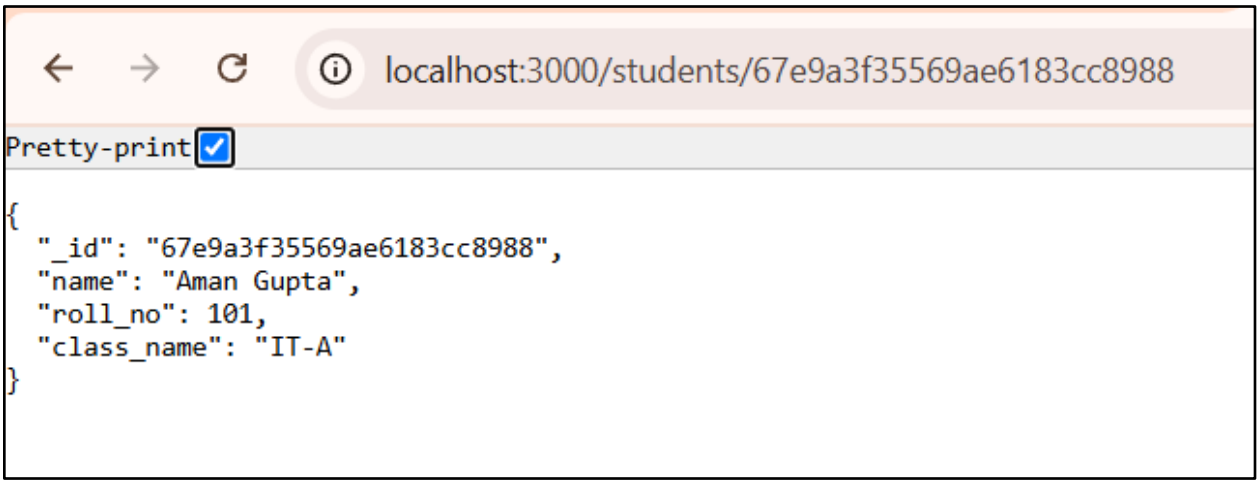
Get all students



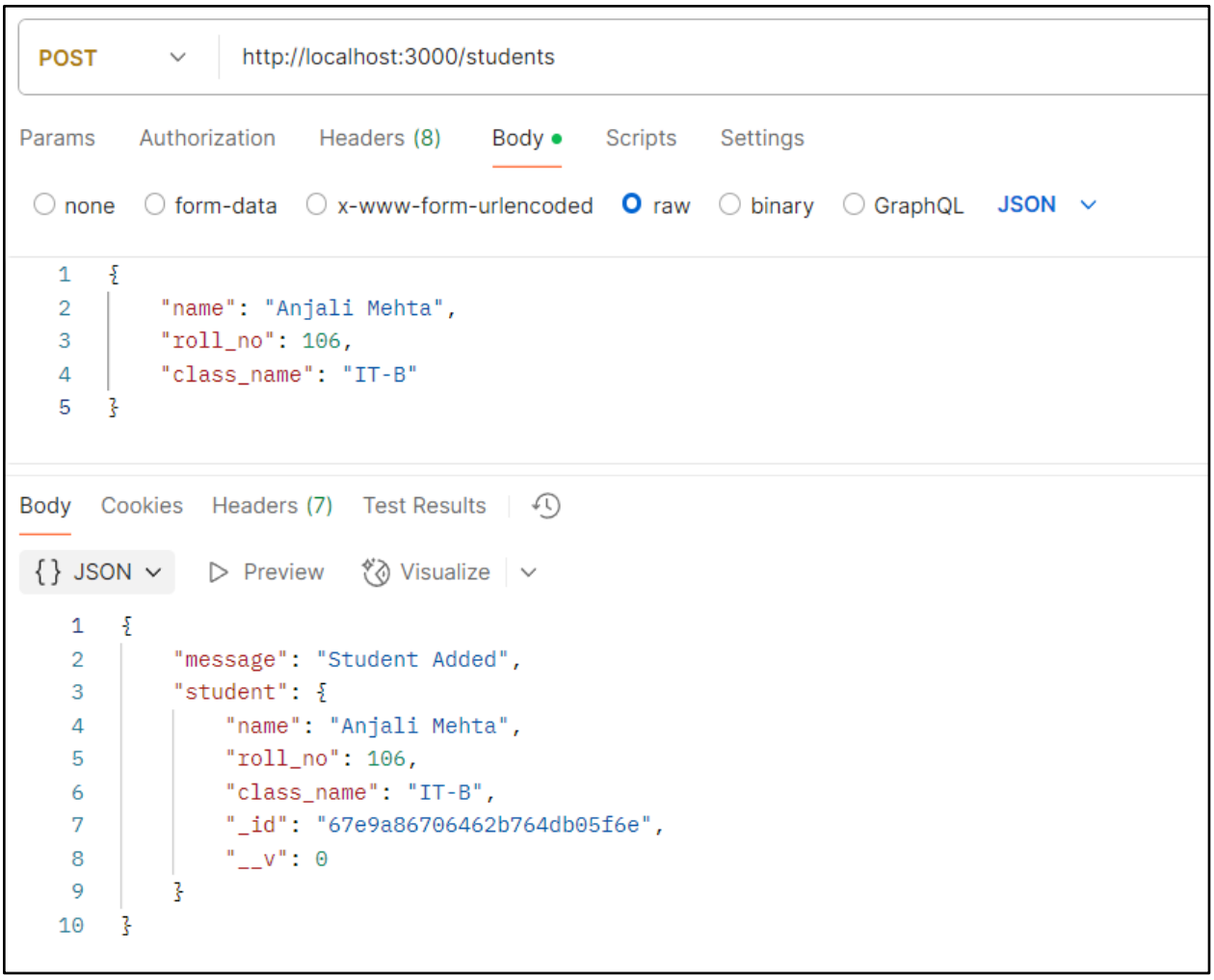
Get students of a particular class



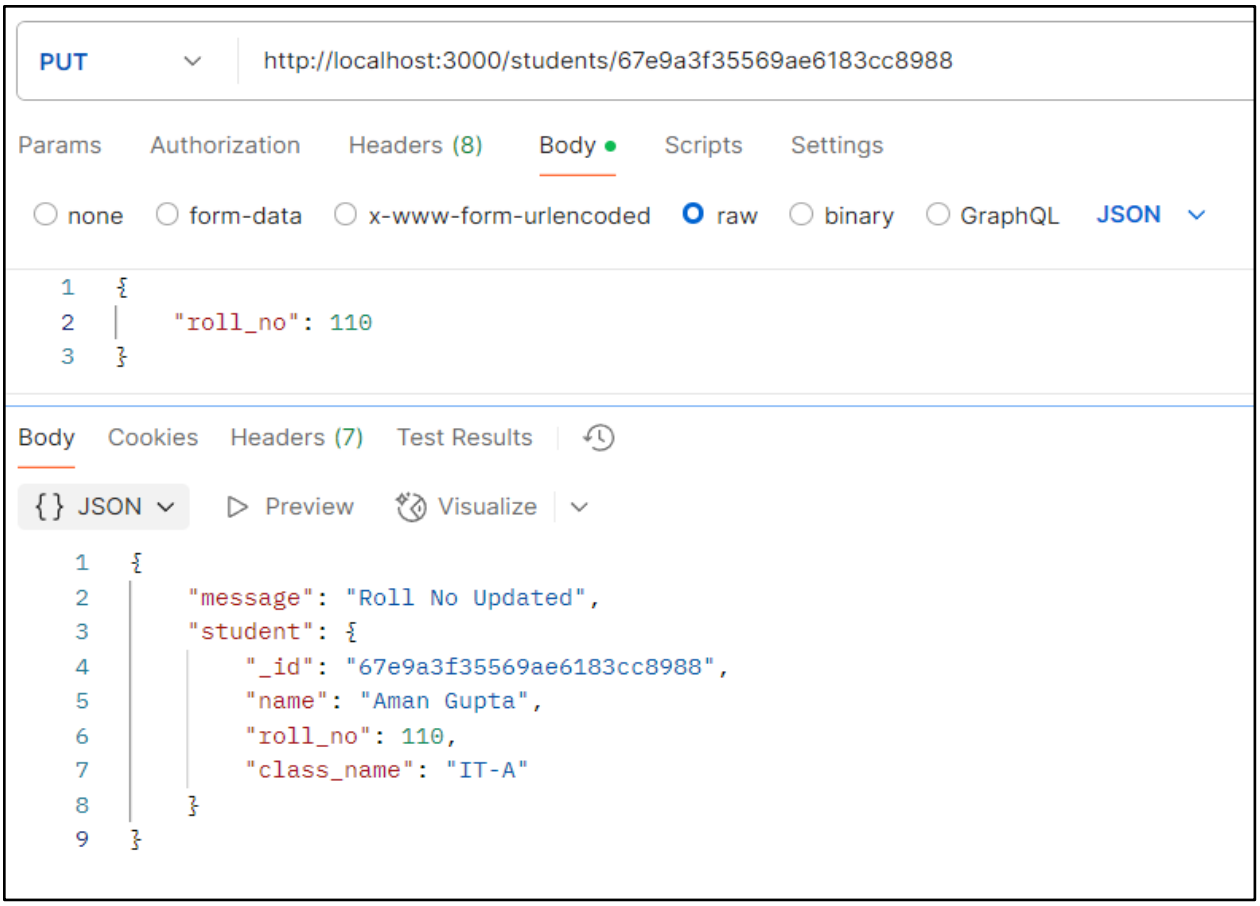
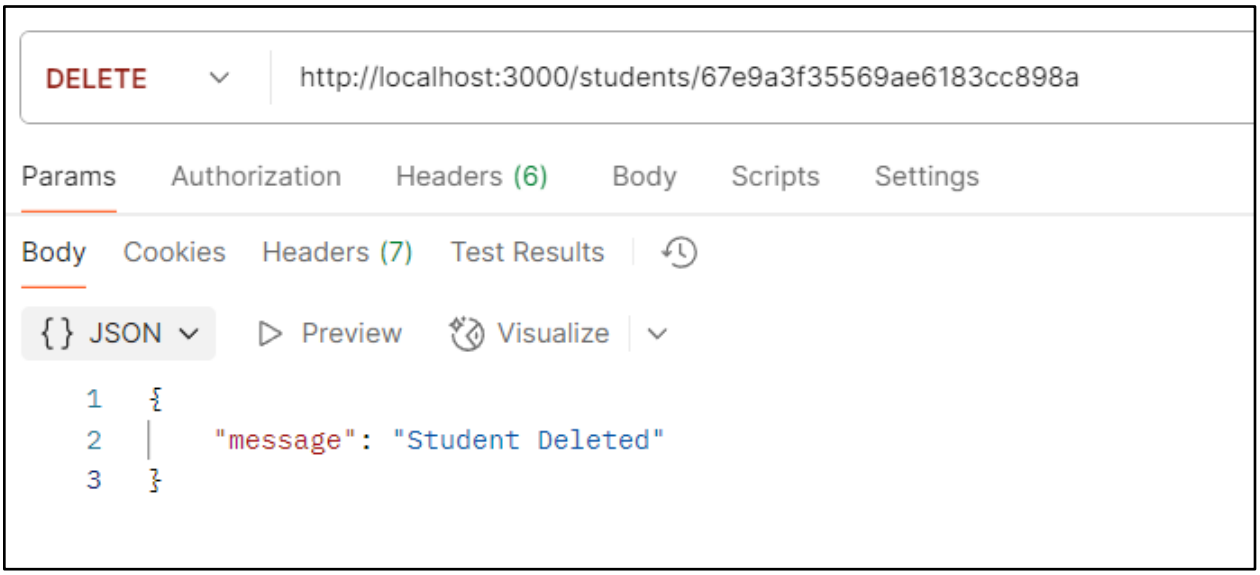
Get student by id



Add a student



Update a student's details



Delete a student