

**EXPERIMENT NO. 7 : MongoDB**

<b>Name of Student</b>	<b><u>Komal Milind Deolekar</u></b>
<b>Class Roll No</b>	<b><u>14</u></b>
<b>D.O.P.</b>	<b><u>25-03-2025</u></b>
<b>D.O.S.</b>	<b><u>01-04-2025</u></b>
<b>Sign and Grade</b>	

**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
- Insert one student details
  - Insert at once multiple student details
  - Display student for a particular class
  - Display students of specific roll no in a class
  - Change the roll no of a student
  - 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.

**Github Link :**

[https://github.com/KomalDeolekar0607/Webx\\_Lab/tree/main/Webx\\_Lab\\_Exp\\_7](https://github.com/KomalDeolekar0607/Webx_Lab/tree/main/Webx_Lab_Exp_7)

**Theory :****RESTful API Overview**

**RESTful APIs** (Representational State Transfer) are web services that allow communication between systems using HTTP requests. They are built on the principles of **REST**, which is an architectural style

for designing networked applications. RESTful APIs are stateless and use standard HTTP methods like GET, POST, PUT, DELETE, etc., to perform operations on resources.

### Key Concepts of RESTful API:

1. **Resources:** Resources are the data or objects exposed by the API. These resources can be any kind of data, such as users, orders, or products.
2. **HTTP Methods:**
  - **GET:** Retrieve data from the server (e.g., get a user profile).
  - **POST:** Create new data on the server (e.g., create a new user).
  - **PUT:** Update existing data on the server (e.g., update a user's profile).
  - **DELETE:** Delete data from the server (e.g., remove a user).

**URL Structure:** A URL (Uniform Resource Locator) identifies the resource. REST APIs often use hierarchical URLs like:

GET /users - Get all users

POST /users - Create a new user

GET /users/{id} - Get a specific user by ID

PUT /users/{id} - Update a specific user by ID

DELETE /users/{id} - Delete a specific user by ID

**HTTP Status Codes:** HTTP responses return status codes that indicate the result of the operation.

- **200 OK:** Successful request.
- **201 Created:** Successfully created resource.
- **400 Bad Request:** Invalid input.
- **404 Not Found:** Resource not found.
- **500 Internal Server Error:** Server-side error.

### Testing a RESTful API using Postman

**Postman** is a popular tool for testing APIs. It allows you to send HTTP requests to an API, test endpoints, and view the responses. It's useful for both **developers** and **QA testers** to ensure APIs work as expected.

**Steps to Test an API on Postman:**

1. **Open Postman:** Install and launch Postman from [Postman's official website](#).
2. **Create a New Request:**
  - Click on the **"New"** button and select **"Request"**.
  - Name your request and add it to a collection.
3. **Choose the HTTP Method:**
  - Select the **HTTP method** you want to use (GET, POST, PUT, DELETE, etc.) from the dropdown on the left side of the URL input field.
4. **Enter the API Endpoint:**
  - Enter the **API URL** (e.g., <https://jsonplaceholder.typicode.com/users>).
  - This URL is the endpoint of the API you want to interact with.
5. **Add Headers (Optional):**
  - Some APIs require specific **headers** (e.g., authentication, content type).
  - In Postman, click the **"Headers"** tab and add the required headers. For example:
    - Authorization: Bearer <token>
    - Content-Type: application/json
6. **Body (For POST, PUT Requests):**
  - If you're testing a **POST** or **PUT** request, you'll need to include the request **body** (data).
  - Click on the **"Body"** tab and choose the format (e.g., **raw** for JSON).

Example:

```
{  
  "name": "John Doe",  
  "email": "johndoe@example.com"  
}
```

7. **Send the Request:**
  - Click on the **"Send"** button to send the request to the API.
  - Postman will show the **response** in the lower section of the screen. The response will include:

- **Status code:** e.g., 200 OK, 201 Created, etc.
- **Response body:** The returned data (usually in JSON format).
- **Response time:** The time taken for the request to process.

#### 8. Check the Response:

- Check if the **status code** is what you expect (200 for success, 404 for not found, etc.).
- Review the **response body** to make sure the data returned is correct.
- If testing a POST request, confirm that the data was successfully created or modified.

#### Example of Testing a RESTful API with Postman:

Let's say you want to test a **GET** request to fetch a list of users from an API.

- **Step 1:** Set the method to **GET**.
- **Step 2:** Enter the URL, e.g., <https://jsonplaceholder.typicode.com/users>.
- **Step 3:** Click on **Send**.
- **Step 4:** Postman will display the response, which should look something like this:

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz"
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv"
  }
  ...
]
```

#### Testing POST request to create a user:

For a **POST** request to create a new user:

- **Step 1:** Set the method to **POST**.
- **Step 2:** Enter the API endpoint, e.g., <https://jsonplaceholder.typicode.com/users>.
- **Step 3:** Add a JSON body with the user details:

```
{  
  "name": "John Doe",  
  "username": "johndoe123",  
  "email": "john.doe@example.com"  
}
```

- **Step 4:** Click **Send**.
- **Step 5:** If successful, you should receive a **201 Created** status and a response body similar to:

```
{  
  "name": "John Doe",  
  "username": "johndoe123",  
  "email": "john.doe@example.com",  
  "id": 11  
}
```

### Key Features of Postman for API Testing:

**Environment Variables:** Postman supports environments, allowing you to manage different configurations (e.g., development, production).

**Collections:** Grouping requests into collections makes it easier to organize and share tests.

**Automated Testing:** Postman allows you to write scripts for testing and assertions.

**Mock Servers:** Simulate API responses for testing without actually calling the backend.

**Documentation:** Generate API documentation directly from Postman collections.

### Postman

**Postman** is a popular and powerful tool used for **API testing**, development, and collaboration. It is widely used by **developers**, **QA testers**, and **API engineers** to interact with APIs (Application Programming Interfaces), verify their behavior, and ensure that they function correctly. Postman simplifies the process of working with APIs by providing an easy-to-use graphical interface that allows users to send HTTP requests, inspect responses, and automate tests.

### Core Features of Postman:

1. **Request Building:** Postman provides a graphical interface to build different types of HTTP requests, such as **GET**, **POST**, **PUT**, **DELETE**, and more. You can define the **URL**, **HTTP method**, **headers**, **parameters**, and **body content** for requests.
2. **Testing and Debugging:** It enables users to test and debug APIs by sending requests and checking the responses. Developers can validate the data returned by the API, check status codes, and ensure that the API behaves as expected.
3. **Request Collection:** Postman allows users to group related API requests into **collections**. This helps organize requests for various API endpoints or test scenarios. Collections also allow you to share them with other team members.
4. **Environment Variables:** Postman allows users to create **environments**, which are sets of variables (like base URL, API keys, etc.) that can be used across requests. This makes it easy to switch between different environments, such as **development**, **staging**, and **production**.
5. **Automation and Scripting:** Postman supports writing **pre-request scripts** and **test scripts**. With scripting, you can automate repetitive tasks, validate responses, and even integrate API tests into your continuous integration (CI) pipelines.
6. **Mock Servers:** Postman allows users to create **mock servers** to simulate API responses. This is useful when the backend is not yet implemented or if you want to test with predefined responses.
7. **Collaboration:** Postman enables easy collaboration within teams. You can share **collections**, **environments**, and **workspaces** with others to streamline team workflows and keep everyone on the same page.
8. **Documentation:** You can generate comprehensive **API documentation** directly from Postman collections. This documentation can be shared with other developers or external stakeholders, making it easy to onboard new developers to your project.
9. **Automated Testing:** With **Newman**, Postman's command-line collection runner, you can automate running tests in CI/CD pipelines. It allows you to execute Postman collections in different environments without needing the Postman app.
10. **History and Versions:** Postman keeps a **history** of all the requests you've made. This lets you revisit old requests without needing to remember the details. You can also track versioning of requests and ensure you are testing with the most current version.

### How Does Postman Work?

1. **Creating Requests:** Postman lets users create **requests** by specifying the HTTP method (GET, POST, PUT, DELETE, etc.), the API endpoint (URL), headers, parameters, and request body (in formats like JSON, XML, or form-data).
2. **Sending Requests:** Once the request is set up, you can **send** the request to the server by pressing the "Send" button. Postman will then process the request and display the response from the server,

including:

- **Status Code:** Indicating whether the request was successful (e.g., 200 OK, 404 Not Found).
  - **Response Body:** The actual data returned by the server (often in JSON format).
  - **Response Time:** How long it took to process the request.
  - **Headers:** Additional information about the response (e.g., content-type, authorization).
3. **Testing and Validating Responses:** You can use **test scripts** in Postman to validate the response. For example, you can check that the status code is **200 OK**, verify the response body contains specific values, or ensure that certain headers are present.
  4. **Organizing Requests:** Requests are often grouped into **collections**, which can represent an API's endpoints or a specific test suite. Collections make it easier to manage and organize multiple requests. Collections can be shared among team members.

### How Can You Use Postman for API Testing?

- **GET Requests:** Retrieve data from a server.
  - Example: Testing an API endpoint that returns a list of users or products.
- **POST Requests:** Send data to the server to create new records.
  - Example: Testing a registration API where you submit user data like name and email.
- **PUT Requests:** Update existing data on the server.
  - Example: Updating a user's profile information.
- **DELETE Requests:** Delete data from the server.
  - Example: Deleting a specific user by ID.
- **Handling Form Data and JSON:** Postman allows you to send **form-data** or **raw JSON** payloads in POST/PUT requests, which is helpful for testing APIs that handle different types of data.

### Why Use Postman?

- 1) **Easy to Use:** The graphical interface makes it user-friendly for developers to send requests without writing complex code.
- 2) **API Exploration:** Postman is ideal for exploring APIs, testing different endpoints, and experimenting with various HTTP methods.

- 3) **Collaboration:** It allows multiple users to share collections and environments, making it easier to collaborate on API development.
- 4) **Automated Testing:** With test scripts and Newman, Postman helps integrate API testing into the CI/CD pipeline.

## Output :

### In mogosh

Creating and Switching to studentDB Database

```
studentDB>
switched to db studentDB
```

Inserting a Single Student Document into the students Collection

```
studentDB> db.students.insertOne({ Name: "Komal Deolekar", RollNo: 101, ClassName: "IT-A" })
{
  acknowledged: true,
  insertedId: ObjectId('67f9fdaac80e1146104eeb91')
}
```

Inserting Multiple Student Documents into the students Collection

```
studentDB> db.students.insertMany([ { Name: "Rahul Sharma", RollNo: 102, ClassName: "IT-A" }, { Name: "Sneha Patel", RollNo: 103, ClassName: "IT-B" }, { Name: "Ankit Verma", RollNo: 104, ClassName: "IT-A" }, { Name: "Megha Joshi", RollNo: 105, ClassName: "IT-C" } ] )
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67f9fe15c80e1146104eeb96'),
    '1': ObjectId('67f9fe15c80e1146104eeb97'),
    '2': ObjectId('67f9fe15c80e1146104eeb98'),
    '3': ObjectId('67f9fe15c80e1146104eeb99')
  }
}
studentDB>
```



Finding All Students in Class IT-A

```
studentDB> db.students.find({ ClassName: "IT-A" })
[
  {
    _id: ObjectId('67f9fcdcc80e1146104eeb90'),
    Name: 'Komal Deolekar',
    RollNo: 101,
    ClassName: 'IT-A'
  },
  {
    _id: ObjectId('67f9fe15c80e1146104eeb96'),
    Name: 'Rahul Sharma',
    RollNo: 102,
    ClassName: 'IT-A'
  },
  {
    _id: ObjectId('67f9fe15c80e1146104eeb98'),
    Name: 'Ankit Verma',
    RollNo: 104,
    ClassName: 'IT-A'
  }
]
studentDB> _
```

Finding Student with Roll No 104 in Class IT-A

```
studentDB> db.students.find({ ClassName: "IT-A", RollNo: 104 })
[
  {
    _id: ObjectId('67f9fe15c80e1146104eeb98'),
    Name: 'Ankit Verma',
    RollNo: 104,
    ClassName: 'IT-A'
  }
]
studentDB>
```

## Updating Roll Number of Student Named Ankit Verma

```
studentDB> db.students.updateOne( { Name: "Ankit Verma" }, { $set: { RollNo: 122 } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

After Updating

```
studentDB>
{
  _id: ObjectId('67f9fe15c80e1146104eeb98'),
  Name: 'Ankit Verma',
  RollNo: 122,
  ClassName: 'IT-A'
}
```

## Deleting Student Record of Megha Joshi

```
studentDB> db.students.deleteOne({ Name: "Megha Joshi" })
{ acknowledged: true, deletedCount: 1 }
```

After deletion

```
studentDB> db.students.findOne({ Name: "Megha Joshi" })
null
studentDB>
```

**In python****Code :****exp7.py**

```
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
print("\nConnected to database")

# Create database
db = client["StudentsDB"] # Database will be created when a collection is added
print("\nCreated the database")

# Create collection
collection = db["IT_dept"]
print("\nCreated the Collection")

student = {"Name": "Komal Deolekar", "RollNo": 131, "Class": "IT-A"}
# collection.insert_one(student)
# collection.insert_one({"Name": "Komal", "RollNo": 101, "Class": "IT-A"})
print("\nInserted one student record")

for stu in collection.find({}):
    print(stu)

students = [
    {"Name": "Amit Sharma", "RollNo": 101, "Class": "D5A"},
    {"Name": "Priya Mehta", "RollNo": 102, "Class": "D5B"},
    {"Name": "Rahul Verma", "RollNo": 103, "Class": "D5C"},
    {"Name": "Sneha Patil", "RollNo": 104, "Class": "D10A"},
    {"Name": "Rohan Kulkarni", "RollNo": 105, "Class": "D10B"},
    {"Name": "Neha Gupta", "RollNo": 106, "Class": "D10C"},
    {"Name": "Vikas Yadav", "RollNo": 107, "Class": "D15A"},
    {"Name": "Anjali Nair", "RollNo": 108, "Class": "D15B"},
    {"Name": "Arjun Singh", "RollNo": 109, "Class": "D15C"},
    {"Name": "Komal Deolekar", "RollNo": 110, "Class": "D20A"},
    {"Name": "Siddharth Desai", "RollNo": 111, "Class": "D20B"},
    {"Name": "Pooja Iyer", "RollNo": 112, "Class": "D20C"},
    {"Name": "Varun Malhotra", "RollNo": 113, "Class": "D5A"},
    {"Name": "Kiran Rao", "RollNo": 114, "Class": "D10B"},
```

```
{ "Name": "Meera Joshi", "RollNo": 115, "Class": "D15A"},
{ "Name": "Rajesh Kumar", "RollNo": 116, "Class": "D5B"},
{ "Name": "Nidhi Agarwal", "RollNo": 117, "Class": "D5C"},
{ "Name": "Akash Singh", "RollNo": 118, "Class": "D10A"},
{ "Name": "Simran Kaur", "RollNo": 119, "Class": "D10C"},
{ "Name": "Yash Raj", "RollNo": 120, "Class": "D15B"},
{ "Name": "Deepak Tiwari", "RollNo": 121, "Class": "D15C"},
{ "Name": "Harsha Reddy", "RollNo": 122, "Class": "D20A"},
{ "Name": "Vivek Chauhan", "RollNo": 123, "Class": "D20B"},
{ "Name": "Shweta Pandey", "RollNo": 124, "Class": "D20C"},
{ "Name": "Ayesha Khan", "RollNo": 125, "Class": "D5A"},
{ "Name": "Pankaj Sinha", "RollNo": 126, "Class": "D10A"},
{ "Name": "Gaurav Saxena", "RollNo": 127, "Class": "D15A"},
{ "Name": "Bhavana Menon", "RollNo": 128, "Class": "D20C"},
{ "Name": "Manoj Pillai", "RollNo": 129, "Class": "D5B"},
{ "Name": "Tanya D'Souza", "RollNo": 130, "Class": "D10B"}
]
# collection.insert_many(students)
print("\nMultiple student records inserted!")

for stu in collection.find({}):
    print(stu)

print("\nD15A Student List : ")
for product in collection.find({"Class": "D15A"}):
    print(product)

student = collection.find_one({"RollNo" : 131})
print(f"\nStudent with Roll No. 131 : ${student}")

print("\nRoll number in [101, 105, 110, 120, 130]")

# List of roll numbers to search for
roll_numbers = [101, 105, 110, 120, 130]

# Find students with these roll numbers
students = collection.find({"RollNo": {"$in": roll_numbers}})

# Print the results
for student in students:
    print(student)

print("\nwith direct List : ")
```

```
# Find students with these roll numbers
students = collection.find({"RollNo": {"$in": [101, 105, 110, 120, 130]}})

# Print the results
for student in students:
    print(student)

collection.update_one({"Name" : "Vivek Chauhan"}, {"$set":{"RollNo": 132}})
print("\nRoll number updated")

student = collection.find_one({"RollNo" : 132})
print("\nUpdated data :",student)

collection.delete_one({"Name" : "Manoj Pillai"})
print("\nManoj Pillai's record deleted")

print("\nPrinting the data..")
for stu in collection.find({}):
    print(stu)
```

**Output :**

```
Connected to database
Created the database
Created the Collection
```

```
Inserted one student record
{'_id': ObjectId('67eb2cc84d8da0305eae4b92'), 'Name': 'Komal Deolekar', 'RollNo': 131, 'Class': 'IT-A'}
```

Multiple student records inserted!

```
{'_id': ObjectId('67eb2cc84d8da0305eae4b92'), 'Name': 'Komal Deolekar', 'RollNo': 131, 'Class': 'IT-A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b93'), 'Name': 'Amit Sharma', 'RollNo': 101, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b94'), 'Name': 'Priya Mehta', 'RollNo': 102, 'Class': 'D5B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b95'), 'Name': 'Rahul Verma', 'RollNo': 103, 'Class': 'D5C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b96'), 'Name': 'Sneha Patil', 'RollNo': 104, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b97'), 'Name': 'Rohan Kulkarni', 'RollNo': 105, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b98'), 'Name': 'Neha Gupta', 'RollNo': 106, 'Class': 'D10C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b99'), 'Name': 'Vikas Yadav', 'RollNo': 107, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9a'), 'Name': 'Anjali Nair', 'RollNo': 108, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9b'), 'Name': 'Arjun Singh', 'RollNo': 109, 'Class': 'D15C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9c'), 'Name': 'Komal Deolekar', 'RollNo': 110, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9d'), 'Name': 'Siddharth Desai', 'RollNo': 111, 'Class': 'D20B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9e'), 'Name': 'Pooja Iyer', 'RollNo': 112, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9f'), 'Name': 'Varun Malhotra', 'RollNo': 113, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba0'), 'Name': 'Kiran Rao', 'RollNo': 114, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba1'), 'Name': 'Meera Joshi', 'RollNo': 115, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba2'), 'Name': 'Rajesh Kumar', 'RollNo': 116, 'Class': 'D5B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba3'), 'Name': 'Nidhi Agarwal', 'RollNo': 117, 'Class': 'D5C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba4'), 'Name': 'Akash Singh', 'RollNo': 118, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba5'), 'Name': 'Simran Kaur', 'RollNo': 119, 'Class': 'D10C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba6'), 'Name': 'Yash Raj', 'RollNo': 120, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba7'), 'Name': 'Deepak Tiwari', 'RollNo': 121, 'Class': 'D15C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba8'), 'Name': 'Harsha Reddy', 'RollNo': 122, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba9'), 'Name': 'Vivek Chauhan', 'RollNo': 123, 'Class': 'D20B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4baa'), 'Name': 'Shweta Pandey', 'RollNo': 124, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bab'), 'Name': 'Ayesha Khan', 'RollNo': 125, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bac'), 'Name': 'Pankaj Sinha', 'RollNo': 126, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bad'), 'Name': 'Gaurav Saxena', 'RollNo': 127, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bae'), 'Name': 'Bhavana Menon', 'RollNo': 128, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4baf'), 'Name': 'Manoj Pillai', 'RollNo': 129, 'Class': 'D5B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bb0'), 'Name': 'Tanya D'Souza', 'RollNo': 130, 'Class': 'D10B'}
```

D15A Student List :

```
{'_id': ObjectId('67eb2cc84d8da0305eae4b99'), 'Name': 'Vikas Yadav', 'RollNo': 107, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba1'), 'Name': 'Meera Joshi', 'RollNo': 115, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bad'), 'Name': 'Gaurav Saxena', 'RollNo': 127, 'Class': 'D15A'}
```

Student with Roll No. 131 : `{'_id': ObjectId('67eb2cc84d8da0305eae4b92'), 'Name': 'Komal Deolekar', 'RollNo': 131, 'Class': 'IT-A'}`

Roll number in [101, 105, 110, 120, 130]

```
{'_id': ObjectId('67eb2cc84d8da0305eae4b93'), 'Name': 'Amit Sharma', 'RollNo': 101, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b97'), 'Name': 'Rohan Kulkarni', 'RollNo': 105, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9c'), 'Name': 'Komal Deolekar', 'RollNo': 110, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba6'), 'Name': 'Yash Raj', 'RollNo': 120, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bb0'), 'Name': 'Tanya D'Souza', 'RollNo': 130, 'Class': 'D10B'}
```

with direct List :

```
{'_id': ObjectId('67eb2cc84d8da0305eae4b93'), 'Name': 'Amit Sharma', 'RollNo': 101, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b97'), 'Name': 'Rohan Kulkarni', 'RollNo': 105, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9c'), 'Name': 'Komal Deolekar', 'RollNo': 110, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba6'), 'Name': 'Yash Raj', 'RollNo': 120, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bb0'), 'Name': 'Tanya D'Souza', 'RollNo': 130, 'Class': 'D10B'}
```

Roll number updated

Updated data : `{'_id': ObjectId('67eb2cc84d8da0305eae4ba9'), 'Name': 'Vivek Chauhan', 'RollNo': 132, 'Class': 'D20B'}`

## Manoj Pillai's record deleted

Printing the data..

```
{'_id': ObjectId('67eb2cc84d8da0305eae4b92'), 'Name': 'Komal Deolekar', 'RollNo': 131, 'Class': 'IT-A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b93'), 'Name': 'Amit Sharma', 'RollNo': 101, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b94'), 'Name': 'Priya Mehta', 'RollNo': 102, 'Class': 'D5B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b95'), 'Name': 'Rahul Verma', 'RollNo': 103, 'Class': 'D5C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b96'), 'Name': 'Sneha Patil', 'RollNo': 104, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b97'), 'Name': 'Rohan Kulkarni', 'RollNo': 105, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b98'), 'Name': 'Neha Gupta', 'RollNo': 106, 'Class': 'D10C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b99'), 'Name': 'Vikas Yadav', 'RollNo': 107, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9a'), 'Name': 'Anjali Nair', 'RollNo': 108, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9b'), 'Name': 'Arjun Singh', 'RollNo': 109, 'Class': 'D15C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9c'), 'Name': 'Komal Deolekar', 'RollNo': 110, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9d'), 'Name': 'Siddharth Desai', 'RollNo': 111, 'Class': 'D20B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9e'), 'Name': 'Pooja Iyer', 'RollNo': 112, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4b9f'), 'Name': 'Varun Malhotra', 'RollNo': 113, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba0'), 'Name': 'Kiran Rao', 'RollNo': 114, 'Class': 'D10B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba1'), 'Name': 'Meera Joshi', 'RollNo': 115, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba2'), 'Name': 'Rajesh Kumar', 'RollNo': 116, 'Class': 'D5B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba3'), 'Name': 'Nidhi Agarwal', 'RollNo': 117, 'Class': 'D5C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba4'), 'Name': 'Akash Singh', 'RollNo': 118, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba5'), 'Name': 'Simran Kaur', 'RollNo': 119, 'Class': 'D10C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba6'), 'Name': 'Yash Raj', 'RollNo': 120, 'Class': 'D15B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba7'), 'Name': 'Deepak Tiwari', 'RollNo': 121, 'Class': 'D15C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba8'), 'Name': 'Harsha Reddy', 'RollNo': 122, 'Class': 'D20A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4ba9'), 'Name': 'Vivek Chauhan', 'RollNo': 123, 'Class': 'D20B'}
{'_id': ObjectId('67eb2cc84d8da0305eae4baa'), 'Name': 'Shweta Pandey', 'RollNo': 124, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bab'), 'Name': 'Ayesha Khan', 'RollNo': 125, 'Class': 'D5A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bac'), 'Name': 'Pankaj Sinha', 'RollNo': 126, 'Class': 'D10A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bad'), 'Name': 'Gaurav Saxena', 'RollNo': 127, 'Class': 'D15A'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bae'), 'Name': 'Bhavana Menon', 'RollNo': 128, 'Class': 'D20C'}
{'_id': ObjectId('67eb2cc84d8da0305eae4bb0'), 'Name': 'Tanya D'Souza', 'RollNo': 130, 'Class': 'D10B'}
```

## **For restful api**

Npm init -y

Npm i express mongoose body-parser

### **server.js**

```
const express = require("express");
```

```
const mongoose = require("mongoose");
```

```
const app = express();
```

```
app.use(express.json());
```

```
// Connect to MongoDB
```

```
mongoose
```

```
.connect("mongodb://127.0.0.1:27017/student_db")
```

```
.then(() => console.log("Database Connected..."))
```

```
.catch((err) => console.error("Database Connection Error:", err));
```

```
// Define Schema & Model
```

```
const StudentSchema = new mongoose.Schema( {
```

```
  name: { type: String, required: true },
```

```
  age: { type: Number, required: true },
```

```
  grade: { type: String, required: true },
```

```
});
```

```
const Student = mongoose.model("Student", StudentSchema);
```

```
// Get all students
```

```
app.get("/students", async (req, res) => {
```

```
  try {
```

```
    const students = await Student.find();
```

```
    res.status(200).json(students);
```

```
  } catch (err) {
```

```
    res.status(500).json({ error: "Internal Server Error" });
```

```
  }
```

```
});
```

```
// Get 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({ error: "Student not found" });
```

```
    res.status(200).json(student);
```

```
  } catch (err) {
```



```
    res.status(500).json({ error: "Invalid Student ID" });
  }
});

// Add a new student
// app.post("/students", async (req, res) => {
//   try {
//     const newStudent = new Student(req.body);
//     await newStudent.save();
//     res.status(201).json({ message: "Student added successfully", newStudent });
//   } catch (err) {
//     res.status(400).json({ error: "Invalid data format" });
//   }
// });

app.post("/students", async (req, res) => {
  try {
    const { name, age, grade } = req.body;

    // Custom validation
    if (!name || !age || !grade) {
      return res.status(400).json({ error: "All fields (name, age, grade) are required" });
    }

    const newStudent = new Student({ name, age, grade });
    await newStudent.save();

    res.status(201).json({ message: "Student added successfully", student: newStudent });
  } catch (err) {
    res.status(400).json({ error: "Invalid data format" });
  }
});

// Update student details by ID
app.put("/students/:id", async (req, res) => {
  try {
    const student = await Student.findByIdAndUpdate(req.params.id, req.body, {
      new: true, // Returns updated document
      runValidators: true, // Ensures updates follow schema rules
    });
  }

  if (!student) return res.status(404).json({ error: "Student not found" });
  res.status(200).json({ message: "Student updated successfully", student });
} catch (err) {
```

```
    res.status(500).json({ error: "Invalid Student ID" });
  }
});

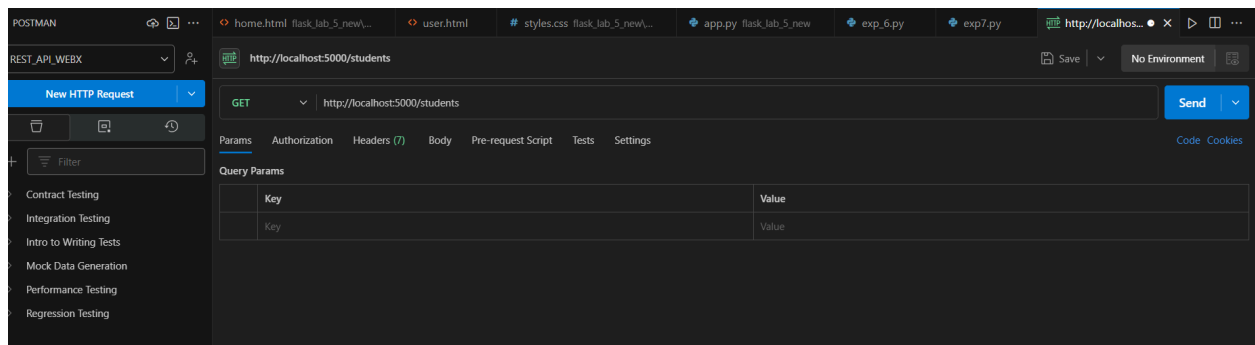
// Delete a student by ID
app.delete("/students/:id", async (req, res) => {
  try {
    const student = await Student.findByIdAndDelete(req.params.id);
    if (!student) return res.status(404).json({ error: "Student not found" });
    res.status(200).json({ message: "Student deleted successfully" });
  } catch (err) {
    res.status(500).json({ error: "Invalid Student ID" });
  }
});

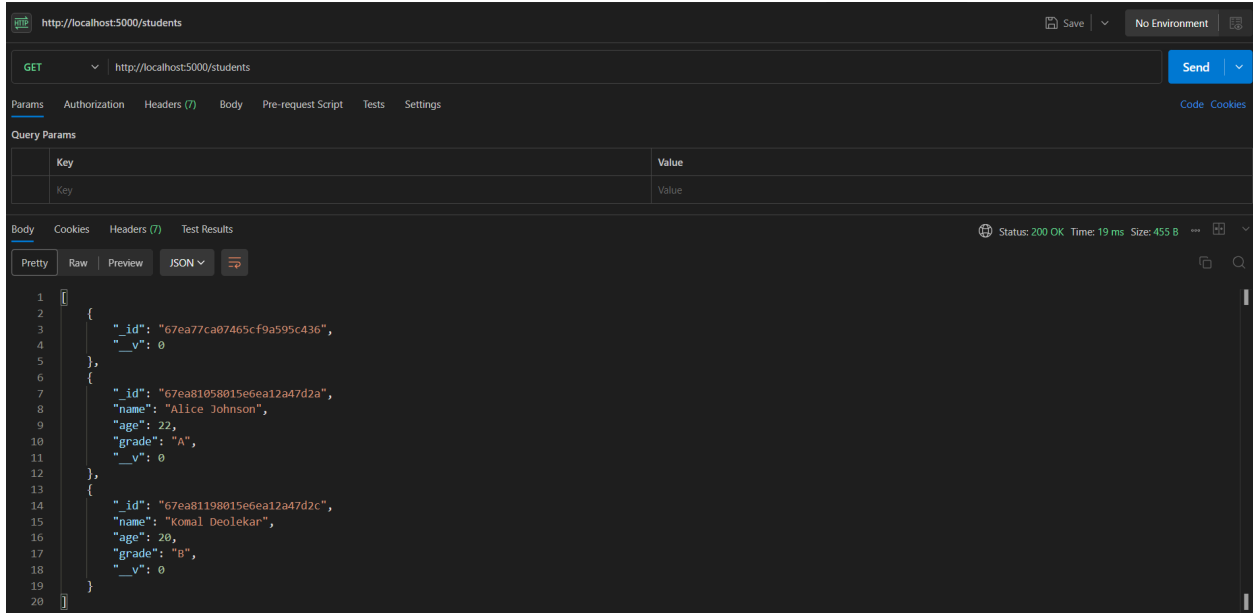
// Start server
const PORT = 5000;
app.listen(PORT, () => console.log(`🚀 Server running on port: ${PORT}`));
```

## Output :

```
D:\Users\Komal\OneDrive\Desktop\sem 6\webx_lab\mongodb_lab_7>node server.js
🚀 Server running on port: 5000
Database Connected...
```

## Get all Students by **GET** request





http://localhost:5000/students

GET http://localhost:5000/students

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

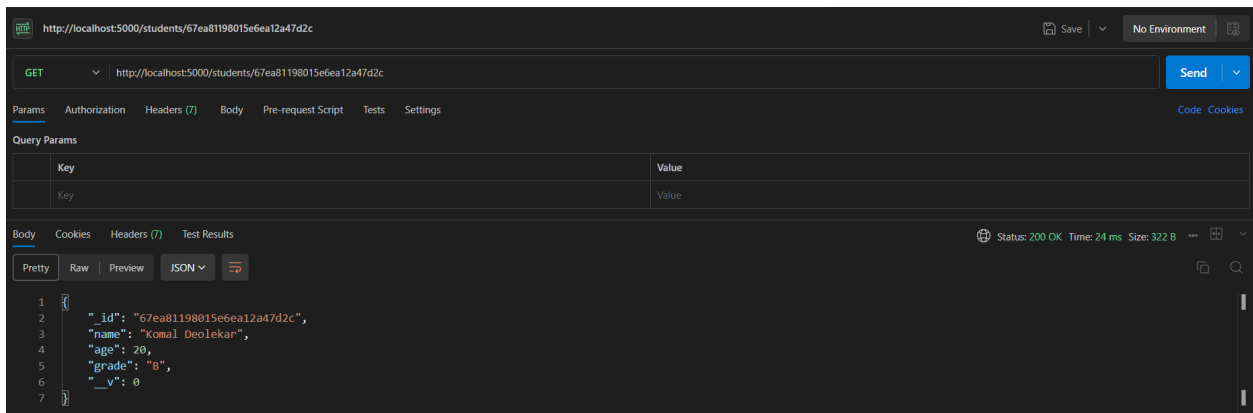
Body Cookies Headers (7) Test Results

Status: 200 OK Time: 19 ms Size: 455 B

Pretty Raw Preview JSON

```
1 {
2   {
3     "id": "67ea77ca07465cf9a595c436",
4     "v": 0
5   },
6   {
7     "id": "67ea81058015e6ea12a47d2a",
8     "name": "Alice Johnson",
9     "age": 22,
10    "grade": "A",
11    "v": 0
12  },
13  {
14    "id": "67ea81198015e6ea12a47d2c",
15    "name": "Komal Deolekar",
16    "age": 20,
17    "grade": "B",
18    "v": 0
19  }
20 }
```

Get Student By ID by **GET** request



http://localhost:5000/students/67ea81198015e6ea12a47d2c

GET http://localhost:5000/students/67ea81198015e6ea12a47d2c

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 24 ms Size: 322 B

Pretty Raw Preview JSON

```
1 {
2   "id": "67ea81198015e6ea12a47d2c",
3   "name": "Komal Deolekar",
4   "age": 20,
5   "grade": "B",
6   "v": 0
7 }
```

Inserting Student details via **POST** request

The screenshot displays an HTTP client interface with the following details:

- URL:** `http://localhost:5000/students`
- Method:** **POST**
- Body Type:** **raw** (selected), with **JSON** selected as the format.
- Request Body (Raw):**

```
1 {  
2   "name": "John Doe",  
3   "age": 22,  
4   "grade": "A"  
5 }  
6
```
- Response Panel:**
  - Body:** Pretty, Raw, Preview, JSON (selected), and a refresh icon.
  - Response Body (Pretty):**

```
1 {  
2   "message": "Student added successfully",  
3   "student": {  
4     "name": "John Doe",  
5     "age": 22,  
6     "grade": "A",  
7     "_id": "67eb2fa2c4c0f21324f8b9f0",  
8     "_v": 0  
9   }  
10 }
```

Updating student data with **PUT** request

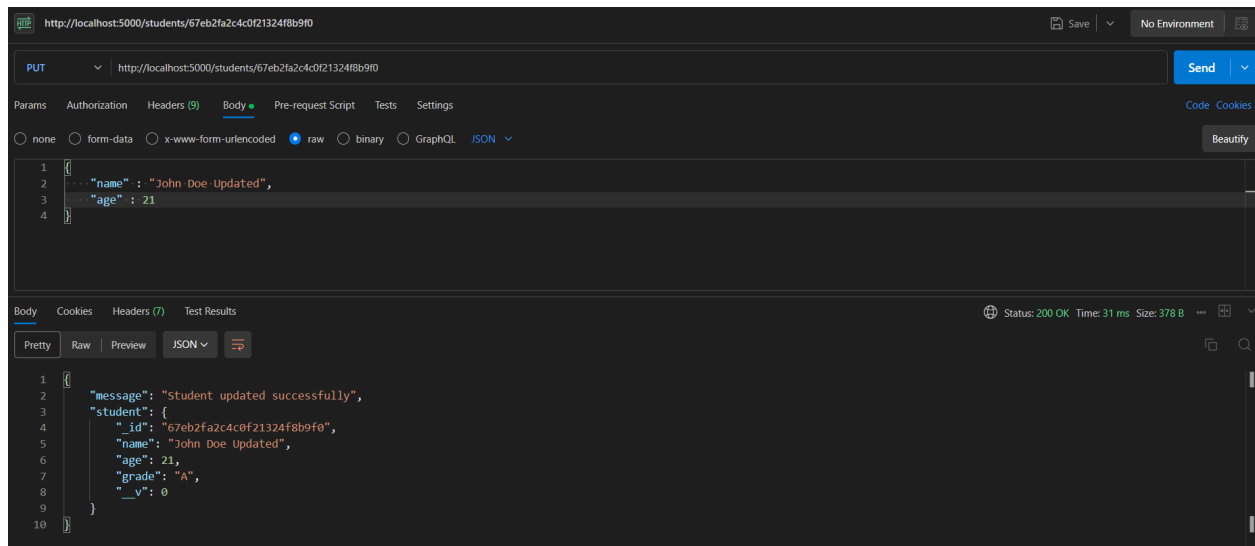
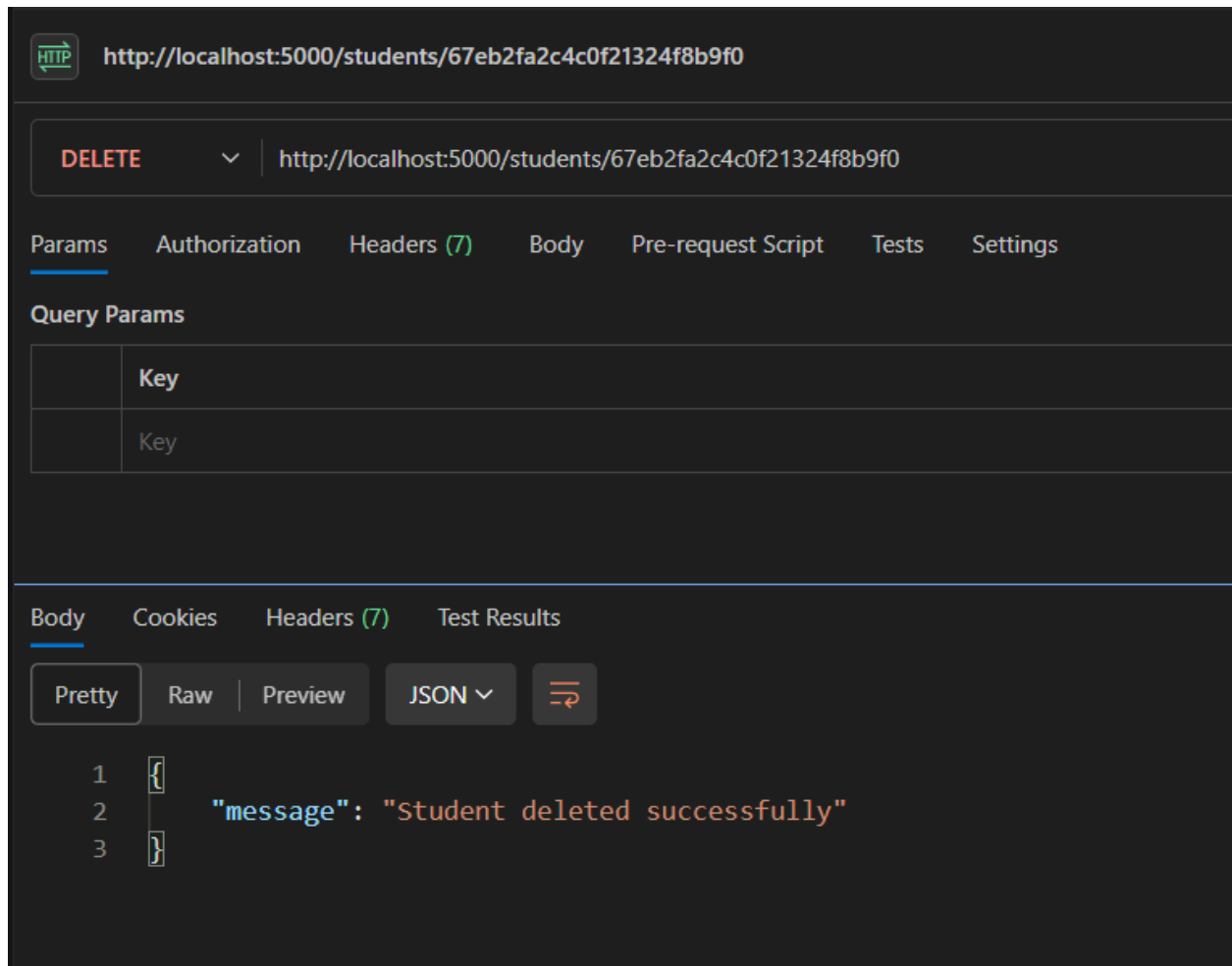
The screenshot shows a REST client interface with the URL `http://localhost:5000/students/67eb2fa2c4c0f21324f8b9f0`. The request method is **PUT**. The response is displayed in the **Body** tab, showing a JSON object with a success message and updated student data.

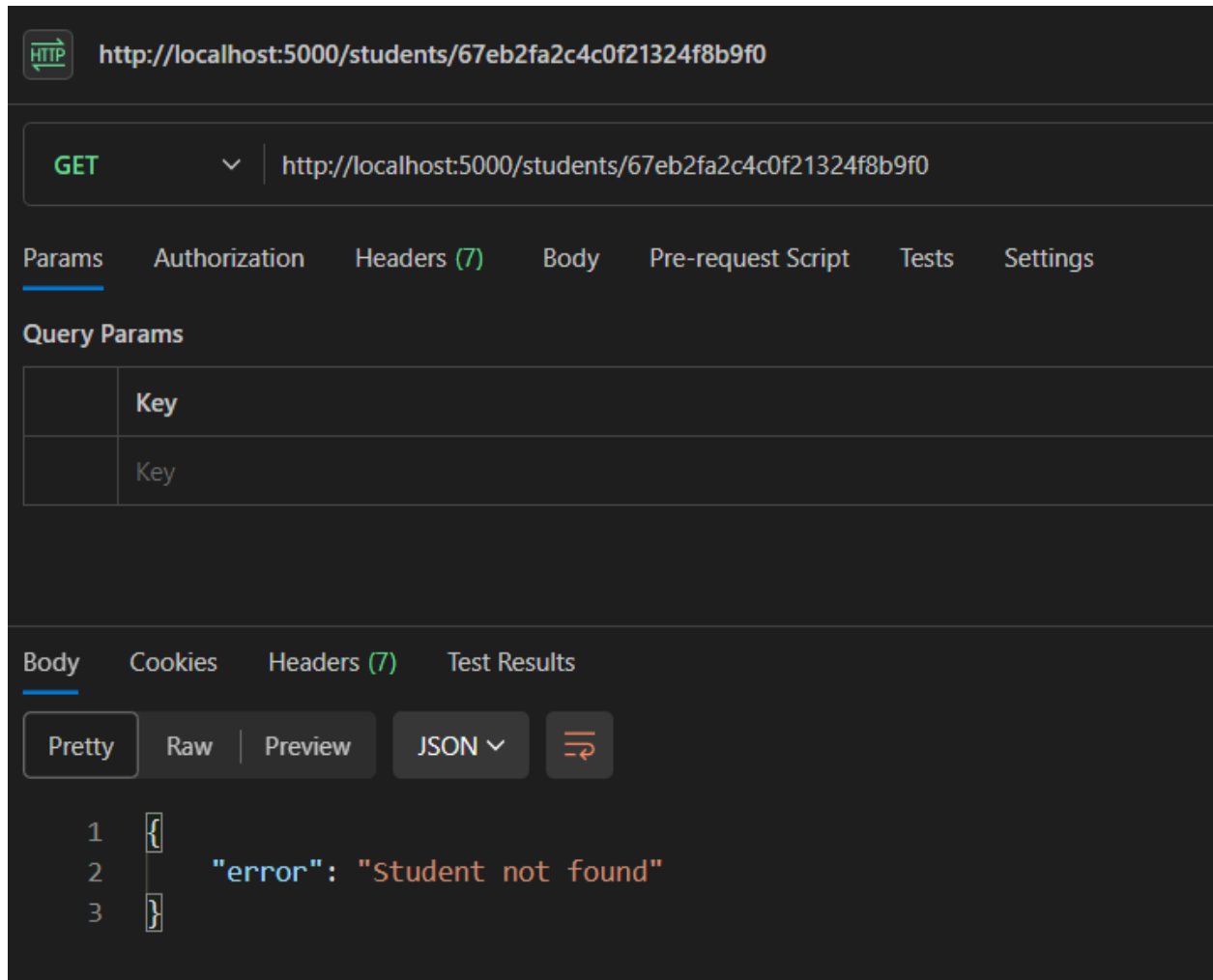
```
{
  "message": "Student updated successfully",
  "student": {
    "_id": "67eb2fa2c4c0f21324f8b9f0",
    "name": "John Doe",
    "age": 22,
    "grade": "A",
    "__v": 0
  }
}
```

The screenshot shows the same REST client interface with the URL `http://localhost:5000/students/67eb2fa2c4c0f21324f8b9f0`. The request method is **PUT**. The **Body** tab is selected, and the request body is shown in raw format as a JSON object.

```
{
  "name": "John Doe Updated",
  "age": 21
}
```

## After updation

Deleting student data with **DELETE** request



## Conclusion :

Postman is a powerful tool for **testing RESTful APIs**, providing a user-friendly interface for making requests, sending data, and inspecting responses. It simplifies the process of **interacting with APIs**, especially when dealing with authentication, headers, and different request methods. By using Postman, you can easily validate your API's behavior, ensuring it performs as expected.