

### Experiment 7 : MongoDB CRUD Operations using RESTful Endpoints.

Name of Student	Niraj Kothawade
Class Roll No	D15A_24
D.O.P.	13/03/2025
D.O.S.	20/03/2025
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
- 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.

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
CopyEdit
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
CopyEdit
GET /api/products
```

Fetch a specific product by ID:

```
bash
CopyEdit
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

CopyEdit

```
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
CopyEdit
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.



OUTPUT:-

A.

## Create

```
> use IT_Department;
< switched to db IT_Department
> db.createCollection("students");
< { ok: 1 }
> db.students.insertOne({
  name: "Niraj Kothawade",
  rollNo: 24,
  className: "D15A"
});
< {
  acknowledged: true,
  insertedId: ObjectId('67fac3497a7b29486a58a359')
}

> db.students.insertMany([
  { name: "Bob Smith", rollNo: 102, className: "IT-A" },
  { name: "Charlie Brown", rollNo: 103, className: "IT-B" },
  { name: "Daisy Miller", rollNo: 104, className: "IT-A" }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67fac3857a7b29486a58a35a'),
    '1': ObjectId('67fac3857a7b29486a58a35b'),
    '2': ObjectId('67fac3857a7b29486a58a35c')
  }
}
```

## Read

```
> db.students.find({ className: "D15A" });
< {
  _id: ObjectId('67fac3497a7b29486a58a359'),
  name: 'Niraj Kothawade',
  rollNo: 24,
  className: 'D15A'
}
> db.students.find({ className: "IT-A", rollNo: 102 });
< {
  _id: ObjectId('67fac3857a7b29486a58a35a'),
  name: 'Bob Smith',
  rollNo: 102,
  className: 'IT-A'
}
```

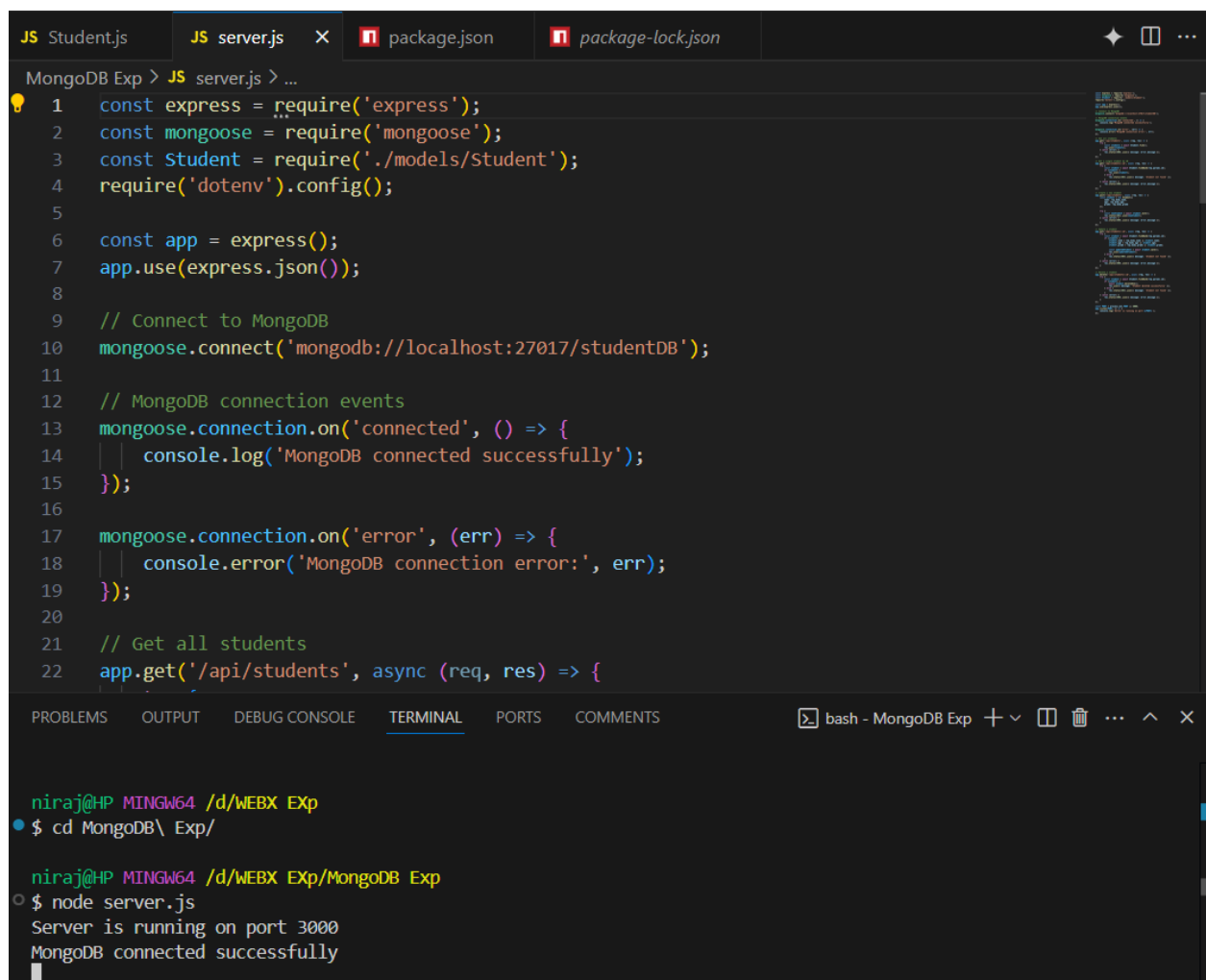
## Update

```
> db.students.updateOne(
  { name: "Bob Smith" }, // filter
  { $set: { rollNo: 202 } } // update
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

## Delete

```
> db.students.deleteOne({ name: "Charlie Brown" });  
< {  
  acknowledged: true,  
  deletedCount: 1  
}
```

## B.



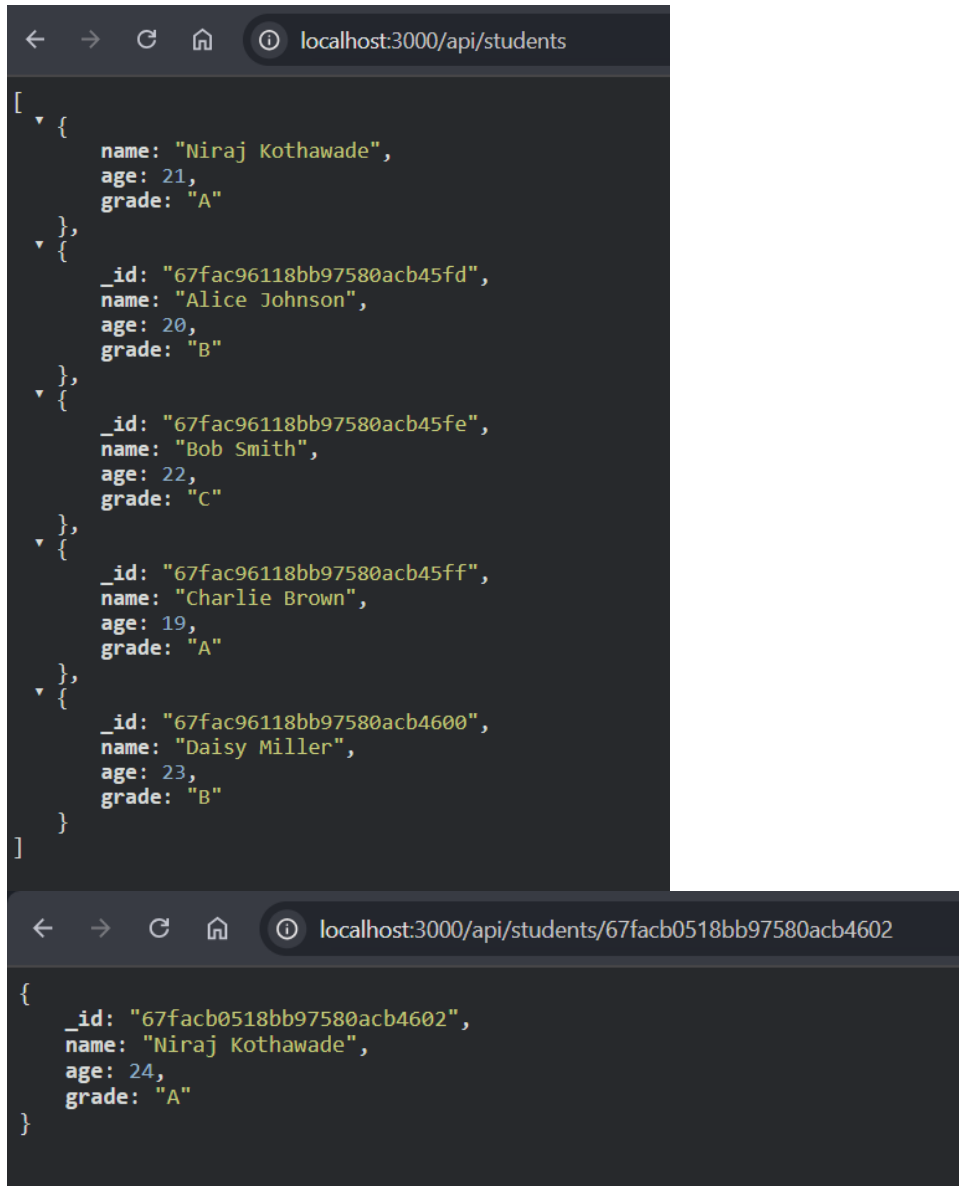
The screenshot displays a VS Code editor with a project named 'MongoDB Exp'. The 'server.js' file contains the following code:

```
1  const express = require('express');  
2  const mongoose = require('mongoose');  
3  const Student = require('./models/Student');  
4  require('dotenv').config();  
5  
6  const app = express();  
7  app.use(express.json());  
8  
9  // Connect to MongoDB  
10 mongoose.connect('mongodb://localhost:27017/studentDB');  
11  
12 // MongoDB connection events  
13 mongoose.connection.on('connected', () => {  
14   console.log('MongoDB connected successfully');  
15 });  
16  
17 mongoose.connection.on('error', (err) => {  
18   console.error('MongoDB connection error:', err);  
19 });  
20  
21 // Get all students  
22 app.get('/api/students', async (req, res) => {
```

The terminal output shows the following commands and results:

```
niraj@HP MINGW64 /d/WEBX Exp  
● $ cd MongoDB\ Exp/  
  
niraj@HP MINGW64 /d/WEBX Exp/MongoDB Exp  
○ $ node server.js  
Server is running on port 3000  
MongoDB connected successfully
```

## Read



```
[
  {
    name: "Niraj Kothawade",
    age: 21,
    grade: "A"
  },
  {
    _id: "67fac96118bb97580acb45fd",
    name: "Alice Johnson",
    age: 20,
    grade: "B"
  },
  {
    _id: "67fac96118bb97580acb45fe",
    name: "Bob Smith",
    age: 22,
    grade: "C"
  },
  {
    _id: "67fac96118bb97580acb45ff",
    name: "Charlie Brown",
    age: 19,
    grade: "A"
  },
  {
    _id: "67fac96118bb97580acb4600",
    name: "Daisy Miller",
    age: 23,
    grade: "B"
  }
]
```

```
{
  _id: "67facb0518bb97580acb4602",
  name: "Niraj Kothawade",
  age: 24,
  grade: "A"
}
```

## Create



```
> db.students.insertOne({
  name: "Niraj Kothawade",
  age: 24,
  grade: "A"
});
< {
  acknowledged: true,
  insertedId: ObjectId('67facb0518bb97580acb4602')
}
```

## Update

```
> db.students.updateOne(
  { name: "Niraj Kothawade" },      // filter
  { $set: { age: 25, grade: "A+" } } // update
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

## Delete

```
> db.students.deleteOne({ name: "Niraj Kothawade" });
< {
  acknowledged: true,
  deletedCount: 1
}
```