

## Experiment – 7: MongoDB Flask connection

Name of Student	Rujuta Medhi
Class Roll No	D15A-27
D.O.P.	06.03.25
D.O.S.	
Sign and Grade	

1) **Aim:** To study CRUD operations in MongoDB

2) **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.

3) **Output:**

A.

**B. Creating student model**

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

```
const { Schema } = mongoose;
```

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

```
  username: {
```

```
    type: String,
```

```
        required: true,
        maxlength: 50
    },
    class_name: {
        type: String,
        required: true,
        maxlength: 25
    },
    roll_number: {
        type: Number,
        required: true,
        min: 0,
        max: 120,
        unique:true
    },
    age:{
        type: Number,
        required: true,
        min: 0,
        max: 120
    },
    grade:{
        type: String,
        required: true
    }
});

StudentSchema.index({ class_name: 1, roll_number: 1 }, { unique: true });
```

```
module.exports = mongoose.model('Student', StudentSchema);
```

1. Add a new student to the database.

```
router.post('/add', async (req, res) => {
  const { username, roll_number, class_name, grade, age } = req.body;
  try {
    const existingUser = await User.findOne({ class_name, roll_number });
    if (existingUser) {
      return res.status(400).json({ message: 'User with this class_name and roll_number already exists' });
    }
    const newUser = new User({
      username,
      roll_number,
      class_name,
      grade,
      age
    });
    await newUser.save();
    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error('Error signing up:', error);
    res.status(500).json({ message: 'Error signing up' });
  }
});
```

POST

Status: 201 Created Size: 42 Bytes Time: 50 ms

Query	Headers <sup>3</sup>	Auth	Body <sup>1</sup>	Tests	Pre Run	
JSON	XML	Text	Form	Form-encode	GraphQL	Binary

JSON Content Format

```
1 {
2   "username": "Rujuta Medhi",
3   "class_name": "D15A",
4   "roll_number": 27,
5   "age": 20,
6   "grade": "A"
7 }
```

Response	Headers <sup>7</sup>	Cookies	Results	Docs
1 {				
2   "message": "User registered successfully"				
3 }				

```
_id: ObjectId('67ead11b467d473ece0b1d74')
username : "Rujuta Medhi"
class_name : "D15A"
roll_number : 27
age : 20
grade : "A"
__v : 0
```

2. Retrieve details of an individual student by ID.

```
router.get('/student/:class_name/:roll_number', async (req, res) => {
  try {
    const {class_name,roll_number}=req.params
    console.log(class_name,roll_number)
    const user = await User.findOne({ class_name: class_name, roll_number: roll_number });
    if (user) {
      return res.json(user); // Return the user_id
    } else {
      return res.status(404).json({ message: 'User not found' });
    }
  } catch (error) {
    return res.status(500).json({ message: 'Internal server error' });
  }
});
```

The screenshot shows a REST client interface. The top bar displays the method 'GET' and the URL 'http://localhost:5000/student/D15A/27'. The status bar indicates 'Status: 200 OK', 'Size: 126 Bytes', and 'Time: 16 ms'. The 'Body' tab is selected, showing the JSON response content. The response is a JSON object with the following fields: '\_id', 'username', 'class\_name', 'roll\_number', 'age', 'grade', and '\_\_v'.

Query	Headers <sup>3</sup>	Auth	Body <sup>1</sup>	Tests	Pre Run	
JSON	XML	Text	Form	Form-encode	GraphQL	Binary

JSON Content

```
1 {
2   "_id": "67ead11b467d473ece0b1d74",
3   "username": "Rujuta Medhi",
4   "class_name": "D15A",
5   "roll_number": 27,
6   "age": 20,
7   "grade": "A",
8   "__v": 0
9 }
```

3. Retrieve a list of all students.

```
router.get('/student/all', async (req, res) => {
  try {
    const user = await User.find();
    if (user) {
      return res.json(user); // Return the user_id
    } else {
      return res.status(404).json({ message: 'User not found' });
    }
  } catch (error) {
    return res.status(500).json({ message: 'Internal server error' });
  }
});
```

The screenshot shows a REST client interface. The top bar displays the method 'GET' and the URL 'http://localhost:5000/student/all'. The 'Send' button is visible. Below the URL bar, there are tabs for 'Query', 'Headers', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Body' tab is selected, showing 'JSON Content' and a 'Format' button. The response area on the right shows a 'Status: 200 OK', 'Size: 251 Bytes', and 'Time: 18 ms'. The response body is a JSON array with two student objects.

```

1  [
2    {
3      "_id": "67eacef3467d473ece0b1d67",
4      "username": "Student",
5      "class_name": "D15A",
6      "roll_number": 100,
7      "age": 20,
8      "grade": "A",
9      "__v": 0
10   },
11   {
12     "_id": "67ead11b467d473ece0b1d74",
13     "username": "Rujuta Medhi",
14     "class_name": "D15A",
15     "roll_number": 27,
16     "age": 20,
17     "grade": "A",
18     "__v": 0
19   }
20 ]

```

#### 4. Update details of an existing student by ID.

```

router.put('/update/:class_name/:roll_number', async (req, res) => {
  try {
    const class_name = req.params.class_name;
    const roll_number = req.params.roll_number;
    const updatedData = req.body;

    console.log('Payload Received:', updatedData);

    const user = await User.findOneAndUpdate(
      { class_name: class_name, roll_number: roll_number },
      { $set: updatedData }, // Update fields
      { new: true, runValidators: true } // Return updated document and validate fields
    );

    if (!user) {
      console.log('No user found for email:');
      return res.status(404).json({ message: 'User not found' });
    }

    console.log('Updated User:', user);
    res.status(200).json({ message: 'Profile updated successfully', user });
  } catch (error) {
    console.error('Error updating user profile:', error);

    // Handle Duplicate Key Error explicitly
    if (error.code === 11000) {
      return res.status(400).json({ message: 'Duplicate email detected' });
    }

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

```

```
});
```

The screenshot shows a REST client interface. The URL bar displays `PUT http://localhost:5000/student/update/D15A/27` with a `Send` button. The `Body` tab is selected, showing a JSON payload: 

```
{ 1 { 2 "username": "Rujuta Medhi", 3 "class_name": "D15A", 4 "roll_number": 27, 5 "age": 20, 6 "grade": "A+" 7 }
```

. The `Response` tab shows the server's reply: 

```
1 { 2 "message": "Profile updated successfully", 3 "user": { 4 "_id": "67ead11b467d473ece0b1d74", 5 "username": "Rujuta Medhi", 6 "class_name": "D15A", 7 "roll_number": 27, 8 "age": 20, 9 "grade": "A+", 10 "__v": 0 11 } 12 }
```

. The status bar at the top right indicates `Status: 200 OK`, `Size: 177 Bytes`, and `Time: 33 ms`.

```
_id: ObjectId('67ead11b467d473ece0b1d74')
username : "Rujuta Medhi"
class_name : "D15A"
roll_number : 27
age : 20
grade : "A+"
__v : 0
```

5. Delete a student from the database by ID.

```
router.delete('/student/delete/:class_name/:roll_number', async (req, res) => {
  try {
    const {class_name,roll_number}=req.params
    console.log(class_name,roll_number)
    const user = await User.findOneAndDelete({ class_name: class_name, roll_number: roll_number
  });
    if (user) {
      return res.json({user,message:"deleted"}); // Return the user_id
    } else {
      return res.status(404).json({ message: 'User not found' });
    }
  } catch (error) {
    return res.status(500).json({ message: 'Internal server error' });
  }
});
```

DELETE

⌵

http://localhost:5000/student/delete/D15A/27

Send

Query

Headers<sup>3</sup>

Auth

Body

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1

Status: 200 OK

Size: 156 Bytes

Time: 1.99 s

Response

Headers<sup>7</sup>

Cookies

Results

Docs

1

{

2

"user": {

3

"\_id": "67ead11b467d473ece0b1d74",

4

"username": "Rujuta Medhi",

5

"class\_name": "D15A",

6

"roll\_number": 27,

7

"age": 20,

8

"grade": "A+",

9

"\_\_v": 0

10

},

11

"message": "deleted"

12

}