

## Experiment – 7: MongoDB Flask connection

Name of Student	Shravani Anil Patil
Class Roll no	D15A-37
D.O.P	20.03.25
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' });
  }
});
```

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/student/add
- Status:** 201 Created
- Size:** 42 Bytes
- Time:** 92 ms
- Request Body (JSON):**

```
{
  "username": "Shravani Patil",
  "class_name": "D15A",
  "roll_number": 37,
  "age": 20,
  "grade": "A"
}
```
- Response Body (JSON):**

```
{
  "message": "User registered successfully"
}
```

```

_id: ObjectId('67eacff8467d473ece0b1d6d')
username : "Shravani Patil"
class_name : "D15A"
roll_number : 37
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' });
  }
});

```

GET	http://localhost:5000/student/D15A/37	Send	Status: 200 OK	Size: 128 Bytes	Time: 21 ms
Query	Headers <sup>2</sup>	Auth	Body <sup>1</sup>	Tests	Pre Run
JSON	XML	Text	Form	Form-encode	GraphQL
JSON Content			Format		
1	{				
2	"username": "Shravani Patil",				
3	"class_name": "D15A",				
4	"roll_number": 37,				
5	"age": 20,				
6	"grade": "A"				
7	}				

### 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 indicates a GET request to `http://localhost:5000/student/all` with a status of 200 OK, size of 253 Bytes, and time of 22 ms. The 'Body' tab is selected, showing the JSON response. The response is an array of two student objects. The first object has an ID, username 'Student', class 'D15A', roll number 100, age 20, grade 'A', and a version number 0. The second object has an ID, username 'Shravani Patil', class 'D15A', roll number 37, age 20, grade 'A', and a version number 0.

```
[
  {
    "_id": "67eacef3467d473ece0b1d67",
    "username": "Student",
    "class_name": "D15A",
    "roll_number": 100,
    "age": 20,
    "grade": "A",
    "__v": 0
  },
  {
    "_id": "67eacff8467d473ece0b1d6d",
    "username": "Shravani Patil",
    "class_name": "D15A",
    "roll_number": 37,
    "age": 20,
    "grade": "A",
    "__v": 0
  }
]
```

### 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 with a PUT request to `http://localhost:5000/student/update/D15A/37`. The response is a 200 OK status with 179 bytes and a time of 26 ms. The response body is a JSON object containing a success message and user details.

Method	URL	Status	Size	Time
PUT	http://localhost:5000/student/update/D15A/37	200 OK	179 Bytes	26 ms

  

Tab	Content
Body	<pre> 1 { 2   "username": "Shravani Patil", 3   "class_name": "D15A", 4   "roll_number": 37, 5   "age": 20, 6   "grade": "A+" 7 } </pre>
Response	<pre> 1 { 2   "message": "Profile updated successfully", 3   "user": { 4     "_id": "67eacff8467d473ece0b1d6d", 5     "username": "Shravani Patil", 6     "class_name": "D15A", 7     "roll_number": 37, 8     "age": 20, 9     "grade": "A+", 10    "__v": 0 11  } 12 } </pre>

```

_id: ObjectId('67eacff8467d473ece0b1d6d')
username : "Shravani Patil"
class_name : "D15A"
roll_number : 37
age : 20
grade : "A"
__v : 0

```

```

_id: ObjectId('67eacff8467d473ece0b1d6d')
username : "Shravani Patil"
class_name : "D15A"
roll_number : 37
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/37	Send	Status: 200 OK	Size: 158 Bytes	Time: 21 ms
Query	Headers <sup>3</sup>	Auth	Body	Tests	Pre Run
JSON	XML	Text	Form	Form-encode	GraphQL
JSON Content	Format				
<pre> 1 </pre>			<pre> 1  { 2    "user": { 3      "_id": "67eacff8467d473ece0b1d6d", 4      "username": "Shravani Patil", 5      "class_name": "D15A", 6      "roll_number": 37, 7      "age": 20, 8      "grade": "A+", 9      "__v": 0 10   }, 11   "message": "deleted" 12 } </pre>		