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
      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.

3) **Output:**

A.

B. Creating student model

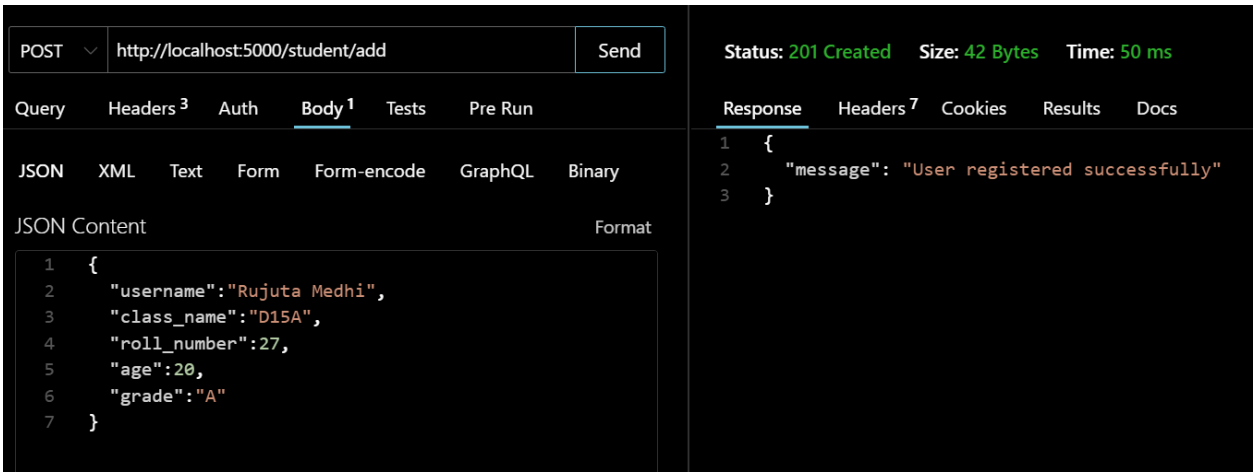
```
const mongoose = require('mongoose');

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.

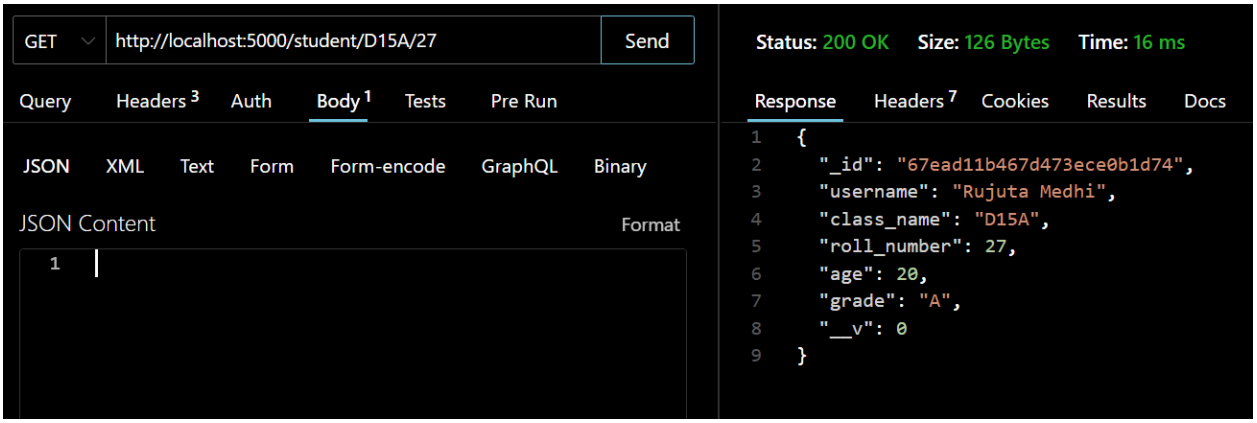
```
router.post('/add', async (req, res) => {
  const { username, roll_number, class_name,grade,age } = req.body;
  try {
    const existingUser = await User.findOne({ class_name, roll_number });
    if (existingUser) {
        return res.status(400).json({ message: 'User with this class_name and roll_number 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' });
  }
});
```




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

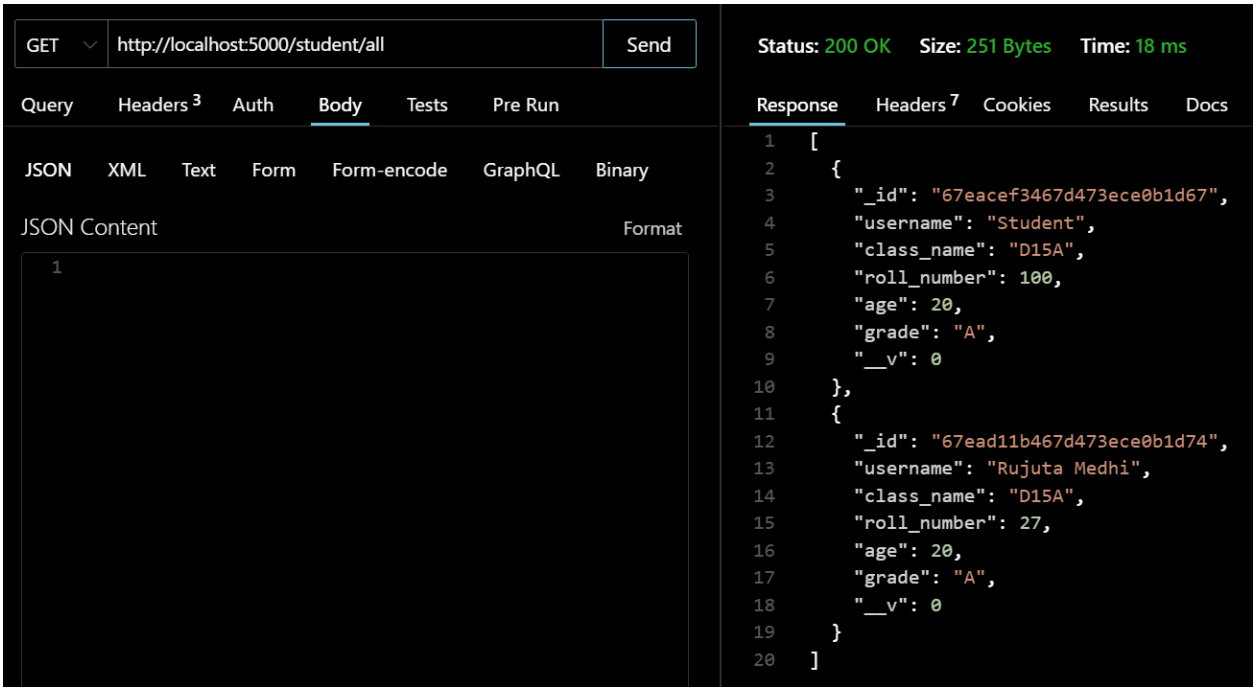
2. Retrieve details of an individual student by ID.

```
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' });
 }
});
```



3. Retrieve a list of all students.

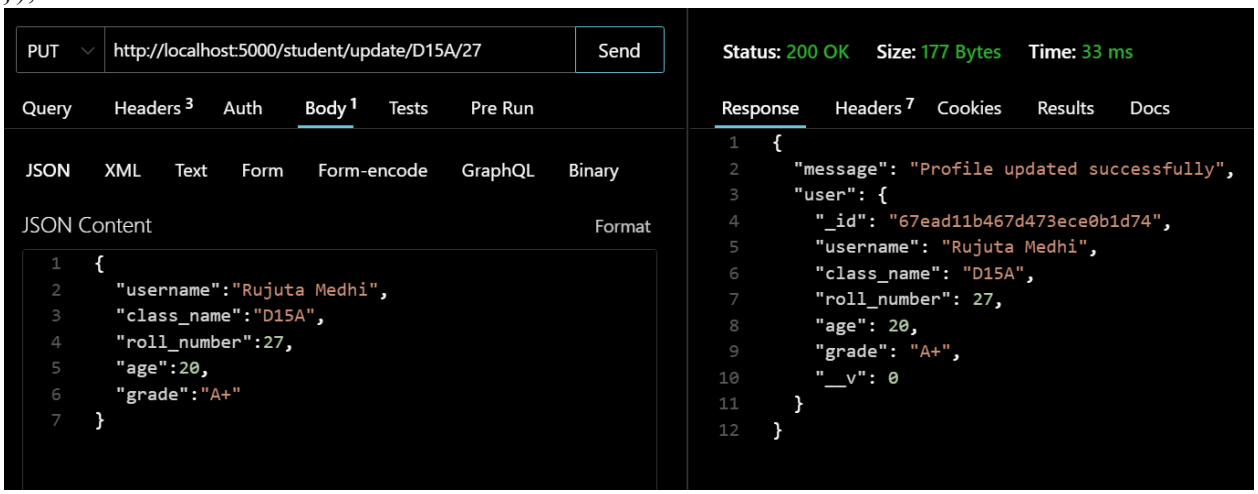
```
router.get('/student/all', async (req, res) => {
 try {

  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' });
 }
});
```

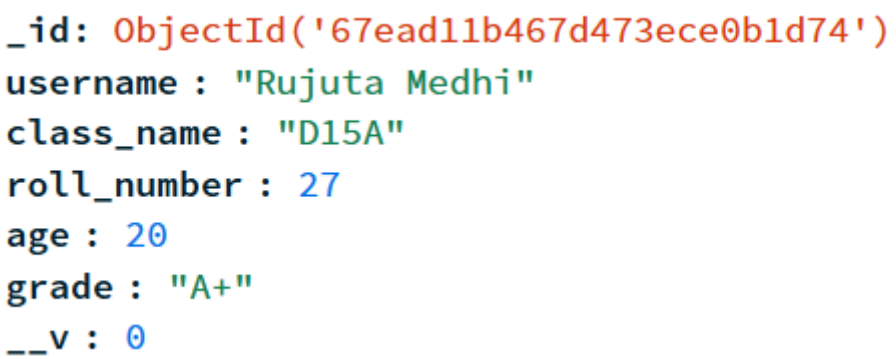


4. Update details of an existing student by ID.

```
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 });
  }
```

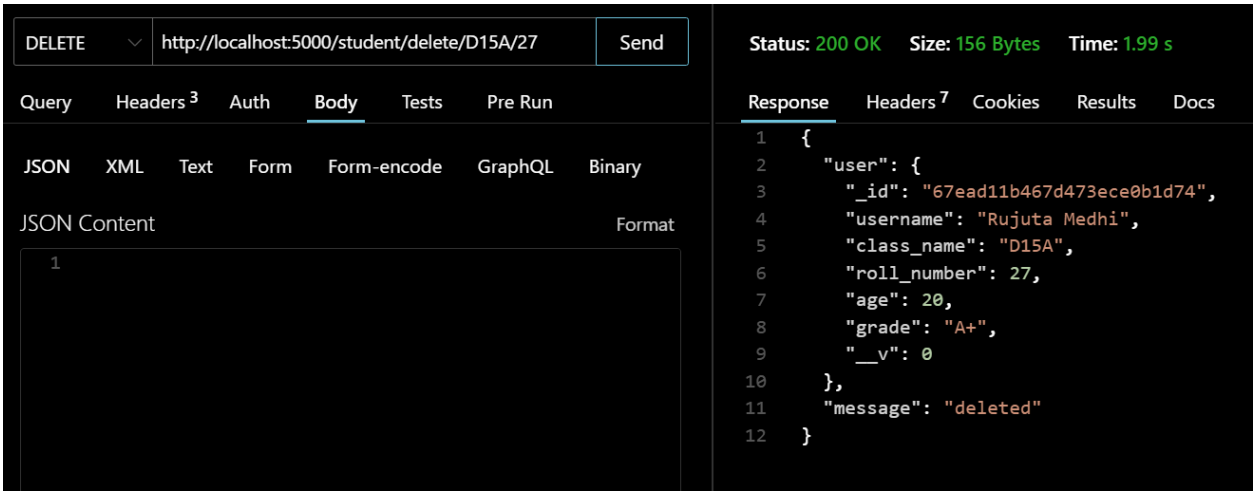
});





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 3
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 7
Cookies
Results
Docs
{
"user": {
"_id": "67ead11b467d473ece0b1d74",
"username": "Rujuta Medhi",
"class_name": "D15A",
"roll_number": 27,
"age": 20,
"grade": "A+",
"__v": 0
},
"message": "deleted"
}