

Experiment – 7: MongoDB

Name of Student	Spandan Deb
Class Roll No	13
D.O.P.	
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.

OUTPUT:

A) Create a Database ITDeptDB and collection students

Create Database ✕

Database Name

ITDeptDB

Collection Name

students

☐ Time-Series

Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

➤ Additional preferences (e.g. Custom collation, Clustered collections)

Cancel

Create Database

Insert 1 Document

```
> use ITDeptDB
< switched to db ITDeptDB
> db.students.insertOne({
  name: "Spandan Deb",
  rollNo: 13,
  className: "IT-101"
})
< {
  acknowledged: true,
  insertedId: ObjectId('67eb5ad5420f754efddae655')
}
```

Insert Multiple student documents

```
> db.students.insertMany([
  {
    name: "Jane Smith",
    rollNo: 102,
    className: "CS-101"
  },
  {
    name: "Michael Johnson",
    rollNo: 103,
    className: "CS-102"
  },
  {
    name: "Emily Davis",
    rollNo: 104,
    className: "CS-102"
  },
  {
    name: "Robert Wilson",
    rollNo: 105,
    className: "CS-103"
  }
])
```

Display a student record for a particular className

```
> db.students.find({ className: "IT-101" })
< {
  _id: ObjectId('67eb5ad5420f754efddae655'),
  name: 'Spandan Deb',
  rollNo: 13,
  className: 'IT-101'
}
```

Display a particular student record for a given className and rollNo

```
> db.students.find({
  rollNo: 13,
  className: "IT-101"
})
< {
  _id: ObjectId('67eb5ad5420f754efddae655'),
  name: 'Spandan Deb',
  rollNo: 13,
  className: 'IT-101'
}
```

Update one student record

```
> db.students.updateOne(
  { name: "Jane Smith" },
  { $set: { rollNo: 110 } }
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Delete one student record

```
> db.students.deleteOne({ name: "Robert Wilson" })
< {
  acknowledged: true,
  deletedCount: 1
}
```

B) Restful API

Server.js

// File: server.js

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

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

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

```
const PORT = process.env.PORT || 3000;
```

// Middleware

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

// Connect to MongoDB

```
mongoose.connect('mongodb://localhost:27017/ITDepartmentDB')
```

```
.then(() => {
```

```
  console.log('MongoDB connected');
```

```
  // Seed the database with initial data
```

```
  initializeDatabase();
```

```
})
```

```
.catch(err => console.log('MongoDB connection error:', err));
```

// Student Schema & Model

```
const Student = mongoose.model('Student', {
```

```
  name: String,
```

```
  age: Number,
```

```
  grade: String
```

```
});
```

```
// Function to initialize the database with sample data
```

```
async function initializeDatabase() {
```

```
  // Check if the database is empty
```

```
  const count = await Student.countDocuments();
```

```
  if (count === 0) {
```

```
    // Add sample data if the database is empty
```

```
    const sampleStudents = [
```

```
      { name: "John Doe", age: 19, grade: "A" },
```

```
      { name: "Jane Smith", age: 20, grade: "A-" },
```

```
      { name: "Michael Johnson", age: 18, grade: "B+" },
```

```
      { name: "Emily Davis", age: 21, grade: "A+" },
```

```
      { name: "Robert Wilson", age: 19, grade: "B" }
```

```
    ];
```

```
    try {
```

```
      await Student.insertMany(sampleStudents);
```

```
      console.log('Database initialized with sample data');
```

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

```
      console.error('Error initializing database:', err);
```

```
    }
```

```
  } else {
```

```
    console.log('Database already contains data, skipping initialization');
```

```
}  
}
```

// Routes - All student operations in a much more concise format

app

// Get all students

```
.get('/api/students', async (req, res) => {  
  try {  
    res.json(await Student.find());  
  } catch (err) {  
    res.status(500).json({ error: err.message });  
  }  
})
```

// Get student by ID

```
.get('/api/students/:id', async (req, res) => {  
  try {  
    const student = await Student.findById(req.params.id);  
    student ? res.json(student) : res.status(404).json({ error: 'Student not found' });  
  } catch (err) {  
    res.status(500).json({ error: err.message });  
  }  
})
```

// Add new student

```
.post('/api/students', async (req, res) => {  
  try {  
    const newStudent = await new Student(req.body).save();  
    res.status(201).json(newStudent);  
  } catch (err) {  
    res.status(400).json({ error: err.message });  
  }  
})
```

// Update student by ID

```
.put('/api/students/:id', async (req, res) => {  
  try {  
    const student = await Student.findByIdAndUpdate(  
      req.params.id,  
      req.body,  
      { new: true }  
    );  
    student ? res.json(student) : res.status(404).json({ error: 'Student not found' });  
  } catch (err) {  
    res.status(400).json({ error: err.message });  
  }  
})
```

// Delete student by ID

```
.delete('/api/students/:id', async (req, res) => {
```



```

try {

  const result = await Student.findByIdAndDelete(req.params.id);

  result ? res.json({ message: 'Student deleted' }) : res.status(404).json({ error:
'Student not found' });

} catch (err) {

  res.status(500).json({ error: err.message });

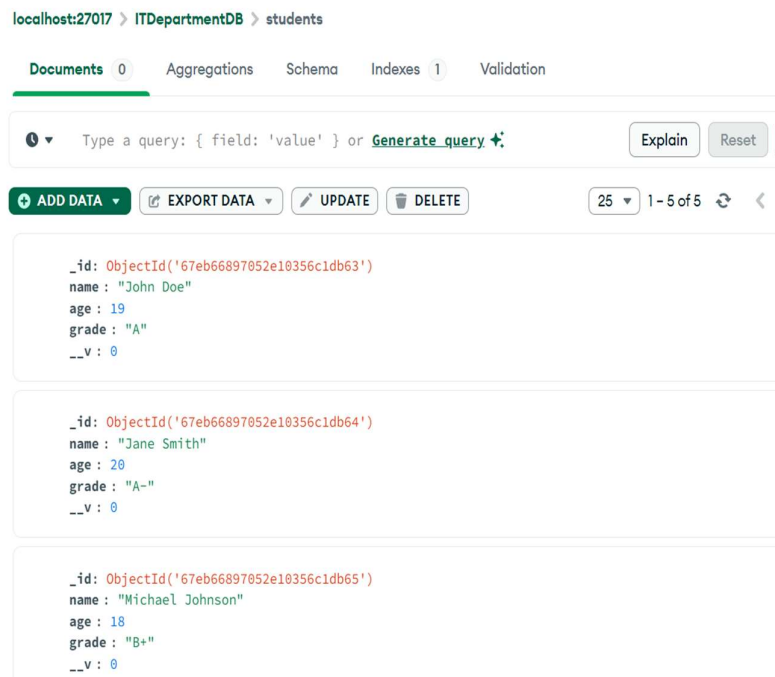
}

});

// Start server

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

```



Retrieve a list of students

GET http://localhost:3000/api/students Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK 1.18 s 668 B Save Response

{ } JSON Preview Visualize

```
1 [
2   {
3     "_id": "67eb66897052e10356c1db63",
4     "name": "John Doe",
5     "age": 19,
6     "grade": "A",
7     "__v": 0
8   },
9   {
10    "_id": "67eb66897052e10356c1db64",
11    "name": "Jane Smith",
12    "age": 20,
13    "grade": "A-",
```

Retrieve a student by id

GET http://localhost:3000/api/students/67eb66897052e10356c1db64 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK 96 ms 319 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "_id": "67eb66897052e10356c1db64",
3   "name": "Jane Smith",
4   "age": 20,
5   "grade": "A-",
6   "__v": 0
7 }
```

Adding a student to the database

POST http://localhost:3000/api/students/ Send

Params Authorization Headers (9) Body Scripts Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "name": "Spandan Deb",
3   "age": 20,
4   "grade": "A+"
5 }
```

Body Cookies Headers (7) Test Results 201 Created • 1.40 s • 325 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "name": "Spandan Deb",
3   "age": 20,
4   "grade": "A+",
5   "_id": "67eb6b1c7052e10356c1db6c",
6   "__v": 0
7 }
```

Update a student record here changed the age and grade

PUT http://localhost:3000/api/students/67eb6b1c7052e10356c1db6c

Params Authorization Headers (9) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "age": 21,
3   "grade": "A++"
4 }
5 }
```

Body Cookies Headers (7) Test Results 200 OK • 445 ms • 321 B

{ } JSON Preview Visualize

```
1 {
2   "_id": "67eb6b1c7052e10356c1db6c",
3   "name": "Spandan Deb",
4   "age": 21,
5   "grade": "A++",
6   "__v": 0
7 }
```

Delete a student from database

The screenshot shows a REST client interface. At the top, a **DELETE** method is selected for the URL `http://localhost:3000/api/students/67eb6b1c7052e10356c1db6c`. Below the URL bar are tabs for **Params**, **Authorization**, **Headers (7)**, **Body**, **Scripts**, and **Settings**. The **Query Params** section contains a table with two columns: **Key** and **Value**. The table has one row with the values `Key` and `Value`.

Below the query params, there are tabs for **Body**, **Cookies**, **Headers (7)**, **Test Results**, and a refresh icon. The **Body** tab is active, showing a JSON response. The response is displayed in a code editor with line numbers 1, 2, and 3. The JSON object is `{ "message": "Student deleted" }`. A green status bar at the top right of the response area shows **200 OK**.

Key	Value
Key	Value

```
1 {  
2   |   "message": "Student deleted"  
3   }  
200 OK
```

CONCLUSION:

Implemented CRUD Operations in MongoDB and implemented a Restful API using Node.js, express and mongoose .We learned about create,read,update and delete student records both via MongoDB shell commands and API endpoints.