

Name : Harsh Gawali

Class : D15A

Roll no. : 19

Experiment – 4: MongoDB

Aim: To study CRUD operations in MongoDB

Theory:

1) Describe some of the features of MongoDB?

Flexible schema: MongoDB does not require a predefined schema, allowing for dynamic and polymorphic data structures.

Scalability: It can handle large amounts of data and scale horizontally through sharding.

High availability: Provides features like replica sets for maintaining data redundancy and ensuring availability.

Querying: Supports rich query expressions, including geospatial queries, text search, and aggregation.

Indexing: Allows for efficient querying by creating indexes on fields.

Aggregation Framework: Provides powerful aggregation pipeline for data aggregation operations.

GridFS: Enables storage and retrieval of large files, exceeding the document size limit.

Native JSON support: Stores data in JSON-like documents, making it easy to work with data in application code.

2) What are Documents and Collections in MongoDB?

Document: In MongoDB, a document is a unit of data storage, similar to a row in a relational database. It is represented in BSON (Binary JSON) format and can contain nested structures.

Collection: A collection is a group of MongoDB documents. It is analogous to a table in a relational database. Collections do not enforce a schema, so documents within a collection can have different fields.

3) When to use MongoDB?

MongoDB is a popular NoSQL database, that shines in situations where relational databases are less ideal. Here are some cases where MongoDB is a good choice:

Unstructured or evolving data: MongoDB stores data in flexible documents, which can hold various data types and don't require a predefined schema. This makes it perfect for data that doesn't fit neatly into rigid tables or where the structure is constantly changing.

Rapid application development: MongoDB's schema-less nature makes it easier and faster to develop applications, especially when you're prototyping or the data model is uncertain. You can add new fields to documents without affecting existing data.

Big data and scalability: MongoDB excels at handling large amounts of data and scales well horizontally by adding more servers to the cluster. This makes it suitable for applications that deal with massive datasets.

4) What is Sharding in MongoDB?

Sharding is a technique used to horizontally partition data across multiple servers or nodes in a distributed database environment. The primary goal of sharding is to distribute the data and workload evenly across multiple machines to improve scalability and performance.

In MongoDB, sharding involves the following key components and processes: Shard: A shard is a subset of data that is stored on a single server or node within a MongoDB cluster. Each shard typically contains a portion of the total dataset. MongoDB supports automatic data distribution across shards based on shard keys.

Shard Key: The shard key is a field or combination of fields chosen to determine how data is distributed across shards. MongoDB uses the shard key to partition data into chunks, which are then distributed across shards. It's crucial to choose an appropriate shard key to ensure even distribution of data and efficient query routing.

- 5) What role does Mongoose play in building a RESTful API with MongoDB, and why is it commonly used?

In building a RESTful API with MongoDB, Mongoose acts as an Object Data Modeling (ODM) layer that simplifies interaction between your application and the database. Here's why it's commonly used:

1. Schema Definition and Data Validation:

Mongoose lets you define schemas for your data, similar to tables in relational databases. These schemas specify the structure of your documents, including data types, validation rules, and relationships. This schema-based approach enforces data integrity and consistency within your API.

2. Easier CRUD operations:

Mongoose provides a clean and intuitive way to perform CRUD (Create, Read, Update, Delete) operations on your data. It offers methods like `create`, `find`, `findByIdAndUpdate`, and `deleteOne` that map to common API functionalities. This reduces the need for writing raw MongoDB queries, making your code more readable and maintainable.

3. Mongoose Middleware:

Mongoose offers middleware functionality that allows you to intercept and modify data before or after database operations. This is useful for tasks like data pre-processing, validation checks, and automatic population of related data.

4. Object-oriented Interaction:

Mongoose lets you interact with your data using objects that represent your documents. This object-oriented approach makes working with data in your API more intuitive and feels more natural for developers familiar with object-oriented programming.

- 6) List features of REST architectural style.

Statelessness: Each request from a client to the server must contain all the information necessary to understand the request, and the server should not store any client context between requests.

Uniform Interface: Resources are uniquely

identified and manipulated through a standard set of methods (HTTP verbs) like GET, POST, PUT, DELETE.

Client-Server Architecture: The client and server are independent of each other, allowing them to evolve separately.

Cacheability: Responses from the server can be explicitly or implicitly marked as cacheable to improve performance.

Layered System: The architecture can be composed of multiple layers (e.g., load balancers, caches, gateways) to improve scalability and flexibility.

7) What are the advantages of using APIs?

APIs (Application Programming Interfaces) offer a bunch of advantages for both developers and businesses alike. Here are some of the key benefits:

Efficiency and Reusability: APIs allow developers to access and leverage functionalities from other applications or services without having to build everything from scratch. This saves time and effort, promoting faster development cycles.

Enhanced Functionality: By integrating APIs, developers can add features and functionalities to their applications that wouldn't be feasible to develop in-house. This can significantly improve the user experience of the final application.

Improved Scalability: APIs enable applications to scale effectively by incorporating functionalities from external sources that can handle increased loads. This makes the application more robust and adaptable to growth.

Innovation and Mashups: APIs open doors to innovation by enabling developers to combine functionalities from various sources to create entirely new applications or mashups. This fosters creativity and leads to unique user experiences.

Problem Statement:

A) Create a database, create a collection, insert data, query and manipulate data using various MongoDB operations.

1. Create a database named "inventory".

```
test> use inventory
switched to db inventory
```

2. Create a collection named "products" with the fields: (ProductID, ProductName, Category, Price, Stock).

```
inventory> db.createCollection("products")
{ ok: 1 }
```

3. Insert 10 documents into the "products" collection.

```
inventory> db.products.insertMany([
... { ProductID: 1, ProductName: "Laptop", Category: "Electronics", Price: 1000, Stock: 20 },
... { ProductID: 2, ProductName: "Smartphone", Category: "Electronics", Price: 800, Stock: 15 },
... { ProductID: 3, ProductName: "Tablet", Category: "Electronics", Price: 500, Stock: 25 },
... { ProductID: 4, ProductName: "Headphones", Category: "Electronics", Price: 100, Stock: 30 },
... { ProductID: 5, ProductName: "TV", Category: "Electronics", Price: 1500, Stock: 10 },
... { ProductID: 6, ProductName: "Shirt", Category: "Clothing", Price: 25, Stock: 50 },
... { ProductID: 7, ProductName: "Jeans", Category: "Clothing", Price: 50, Stock: 40 },
... { ProductID: 8, ProductName: "Shoes", Category: "Footwear", Price: 80, Stock: 35 },
... { ProductID: 9, ProductName: "Backpack", Category: "Accessories", Price: 40, Stock: 60 },
... { ProductID: 10, ProductName: "Watch", Category: "Accessories", Price: 120, Stock: 20 }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65fb1fa5eb704f9499d14a0e'),
    '1': ObjectId('65fb1fa5eb704f9499d14a0f'),
    '2': ObjectId('65fb1fa5eb704f9499d14a10'),
    '3': ObjectId('65fb1fa5eb704f9499d14a11'),
    '4': ObjectId('65fb1fa5eb704f9499d14a12'),
    '5': ObjectId('65fb1fa5eb704f9499d14a13'),
    '6': ObjectId('65fb1fa5eb704f9499d14a14'),
    '7': ObjectId('65fb1fa5eb704f9499d14a15'),
    '8': ObjectId('65fb1fa5eb704f9499d14a16'),
    '9': ObjectId('65fb1fa5eb704f9499d14a17')
  }
}
```

4. Display all the documents in the "products" collection.

```
inventory> db.products.find()
[
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0e'),
    ProductID: 1,
    ProductName: 'Laptop',
    Category: 'Electronics',
    Price: 1000,
    Stock: 20
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0f'),
    ProductID: 2,
    ProductName: 'Smartphone',
    Category: 'Electronics',
    Price: 800,
    Stock: 15
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a10'),
    ProductID: 3,
    ProductName: 'Tablet',
    Category: 'Electronics',
    Price: 500,
    Stock: 25
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a11'),
    ProductID: 4,
    ProductName: 'Headphones',
    Category: 'Electronics',
    Price: 100,
    Stock: 30
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a12'),
    ProductID: 5,
    ProductName: 'TV',
    Category: 'Electronics',
    Price: 1500,
    Stock: 10
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a13'),
    ProductID: 6,
    ProductName: 'Shirt',
    Category: 'Clothing',
    Price: 25,
    Stock: 50
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a14'),
    ProductID: 7,
    ProductName: 'Jeans',
    Category: 'Clothing',
    Price: 50,
    Stock: 40
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a15'),
    ProductID: 8,
    ProductName: 'Shoes',
    Category: 'Footwear',
    Price: 80,
    Stock: 35
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a16'),
    ProductID: 9,
    ProductName: 'Backpack',
    Category: 'Accessories',
    Price: 40,
    Stock: 60
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a17'),
    ProductID: 10,
    ProductName: 'Watch',
    Category: 'Accessories',
    Price: 120,
    Stock: 20
  }
]
```

Display

5. all the products in the "Electronics" category.

```
inventory> db.products.find({ Category: "Electronics" })
[
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0e'),
    ProductID: 1,
    ProductName: 'Laptop',
    Category: 'Electronics',
    Price: 1000,
    Stock: 20
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0f'),
    ProductID: 2,
    ProductName: 'Smartphone',
    Category: 'Electronics',
    Price: 800,
    Stock: 15
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a10'),
    ProductID: 3,
    ProductName: 'Tablet',
    Category: 'Electronics',
    Price: 500,
    Stock: 25
  },
]
```

6. Display all the products in ascending order of their names.

```
inventory> db.products.find().sort({ ProductName: 1 })
[
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a16'),
    ProductID: 9,
    ProductName: 'Backpack',
    Category: 'Accessories',
    Price: 40,
    Stock: 60
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a11'),
    ProductID: 4,
    ProductName: 'Headphones',
    Category: 'Electronics',
    Price: 100,
    Stock: 30
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a14'),
    ProductID: 7,
    ProductName: 'Jeans',
    Category: 'Clothing',
    Price: 50,
    Stock: 40
  },
]
```

7. the details of the first 5 products.

Display

```
inventory> db.products.find().limit(5)
[
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0e'),
    ProductID: 1,
    ProductName: 'Laptop',
    Category: 'Electronics',
    Price: 1000,
    Stock: 20
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a0f'),
    ProductID: 2,
    ProductName: 'Smartphone',
    Category: 'Electronics',
    Price: 800,
    Stock: 15
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a10'),
    ProductID: 3,
    ProductName: 'Tablet',
    Category: 'Electronics',
    Price: 500,
    Stock: 25
  },
  {
    _id: ObjectId('65fb1fa5eb704f9499d14a11'),
    ProductID: 4,
    ProductName: 'Headphones',
    Category: 'Electronics',
    Price: 100,
    Stock: 30
  }
]
```

8. Display the categories of products with a specific name.

```
inventory> db.products.distinct("Category", { ProductName: "Laptop" })
[ 'Electronics' ]
inventory> |
```

9. Display the number of products in the "Electronics" category.

```
inventory> db.products.count({ Category: "Electronics" })
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
5
```

10. Display all the products without showing the "_id" field.

```
inventory> db.products.find({}, { _id: 0 })
[
  {
    ProductID: 1,
    ProductName: 'Laptop',
    Category: 'Electronics',
    Price: 1000,
    Stock: 20
  },
  {
    ProductID: 2,
    ProductName: 'Smartphone',
    Category: 'Electronics',
    Price: 800,
    Stock: 15
  },
  {
    ProductID: 3,
    ProductName: 'Tablet',
    Category: 'Electronics',
    Price: 500,
    Stock: 25
  },
  {
    ProductID: 4,
    ProductName: 'Headphones',
    Category: 'Electronics',
    Price: 100,
    Stock: 30
  }
]
```

11. all the distinct categories of products.

Display

```
inventory> db.products.distinct("Category")
[ 'Accessories', 'Clothing', 'Electronics', 'Footwear' ]
inventory> |
```

12. Display products in the "Electronics" category with prices greater than 50 but less than 100.

```
inventory> db.products.find({ Category: "Electronics", Price: { $gt: 50, $lt: 100 } })
```

13. Change the price of a product.

```
inventory> db.products.updateOne({ ProductName: "Smartphone" }, { $set: { Price: 900 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

14. Delete a particular product entry.

```
inventory> db.products.deleteOne({ ProductID: 10 })
{ acknowledged: true, deletedCount: 1 }
inventory> |
```

B) Create a set of RESTful endpoints using Node.js, Express, and Mongoose for handling student data operations. The endpoints should support:

Code:

```
const express = require('express'); const
mongoose = require('mongoose'); const
bodyParser = require('body-parser');
```

```
// Initialize Express app
const app = express();
```

```
// Set up body parser middleware
app.use(bodyParser.json());
```

```
// Connect to MongoDB using Mongoose
mongoose.connect("mongodb+srv://sarvadnyaawaghad10:picTrO7MwW0z30H3@cluster0.roxgctq.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0").then(() => app.listen(5000))
```


Display

```
.then(() => console.log('database connected  
listenig to 4100'))  
.catch((err) => console.log('foun error', err) )
```

```

// Define Student Schema const studentSchema =
new mongoose.Schema({
  name: String,
  age: Number,
  grade: String
});

// Create Student model
const Student = mongoose.model('Student', studentSchema);

// RESTful endpoints // Retrieve a list of
all students app.get('/students', async
(req, res) => { try {
  const students = await Student.find();
  res.json(students);
} catch (error) { res.status(500).json({ message:
  error.message });
}
});

// Retrieve details of an individual student by ID
app.get('/students/:id', async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json(student);
  } catch (error) { res.status(500).json({ message:
    error.message });
  }
});

// Add a new student to the database
app.post('/students', async (req, res) => {
  const student = new Student({
    name: req.body.name,
    age: req.body.age,
    grade: req.body.grade
  });
  try {

```

```

    const newStudent = await student.save();
    res.status(201).json(newStudent);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});

```

```

// Update details of an existing student by ID
app.put('/students/:id', async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) {
      return res.status(404).json({ message: 'Student not found' });
    }
    student.name = req.body.name; student.age
    = req.body.age; student.grade =
    req.body.grade; const updatedStudent =
    await student.save();
    res.json(updatedStudent);
  } catch (error) { res.status(400).json({ message:
    error.message });
  }
});

```

```

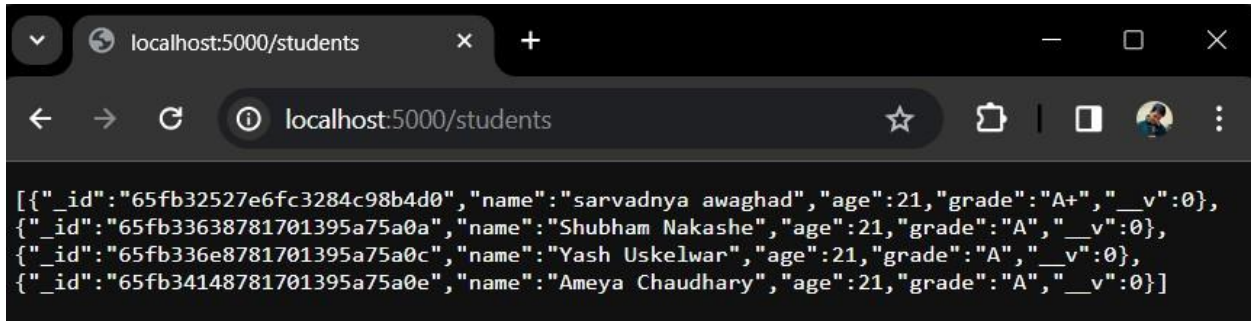
app.delete('/students/:id', async (req, res) => {
  try {
    const deletedStudent = await Student.findOneAndDelete({ _id: req.params.id });
    if (!deletedStudent) {
      return res.status(404).json({ message: 'Student not found' });
    }
    res.json({ message: 'Student deleted', deletedStudent });
  } catch (error) { res.status(500).json({ message:
    error.message });
  }
});

```

Run nodemon index.js to start server

```
PS D:\study material\SEM 6\webxlab\mongorestproject> nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server is running on port 3000
```

Retrieve a list of all students.



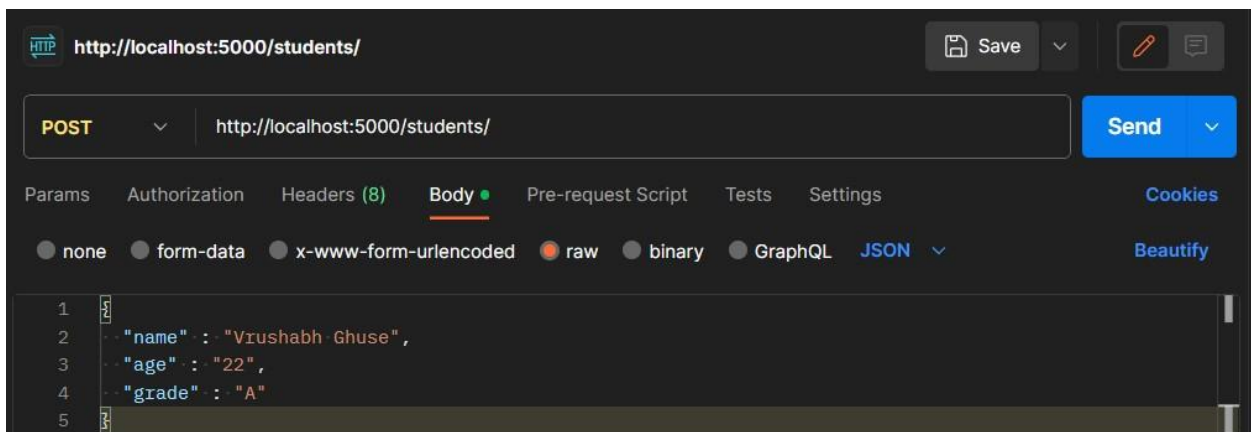
```
[{"_id":"65fb32527e6fc3284c98b4d0","name":"sarvadnya awaghad","age":21,"grade":"A+","__v":0},
{"_id":"65fb33638781701395a75a0a","name":"Shubham Nakashe","age":21,"grade":"A","__v":0},
{"_id":"65fb336e8781701395a75a0c","name":"Yash Uskelwar","age":21,"grade":"A","__v":0},
{"_id":"65fb34148781701395a75a0e","name":"Ameya Chaudhary","age":21,"grade":"A","__v":0}]
```

Retrieve details of an individual student by ID.



```
{"_id":"65fb32527e6fc3284c98b4d0","name":"sarvadnya awaghad","age":21,"grade":"A+","__v":0}
```

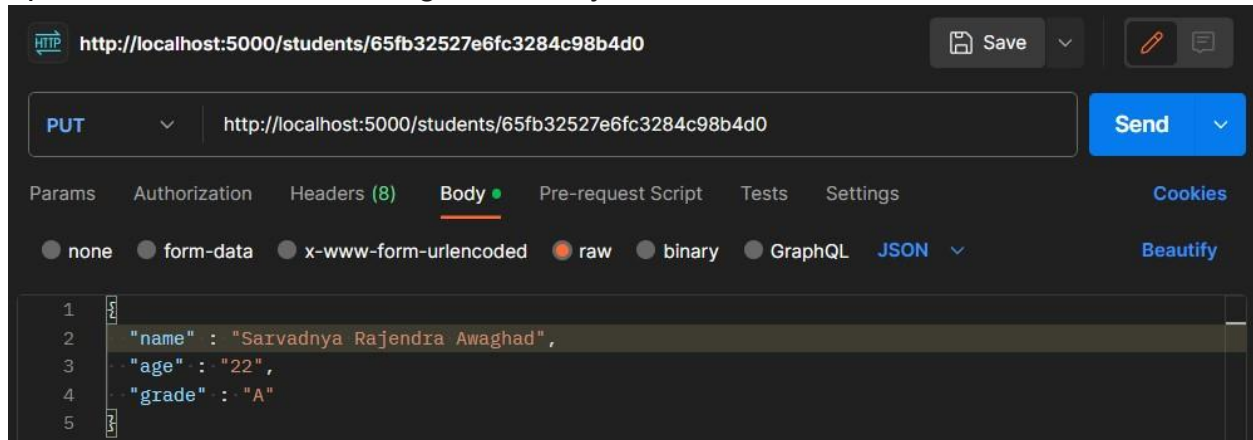
Add a new student to the database.



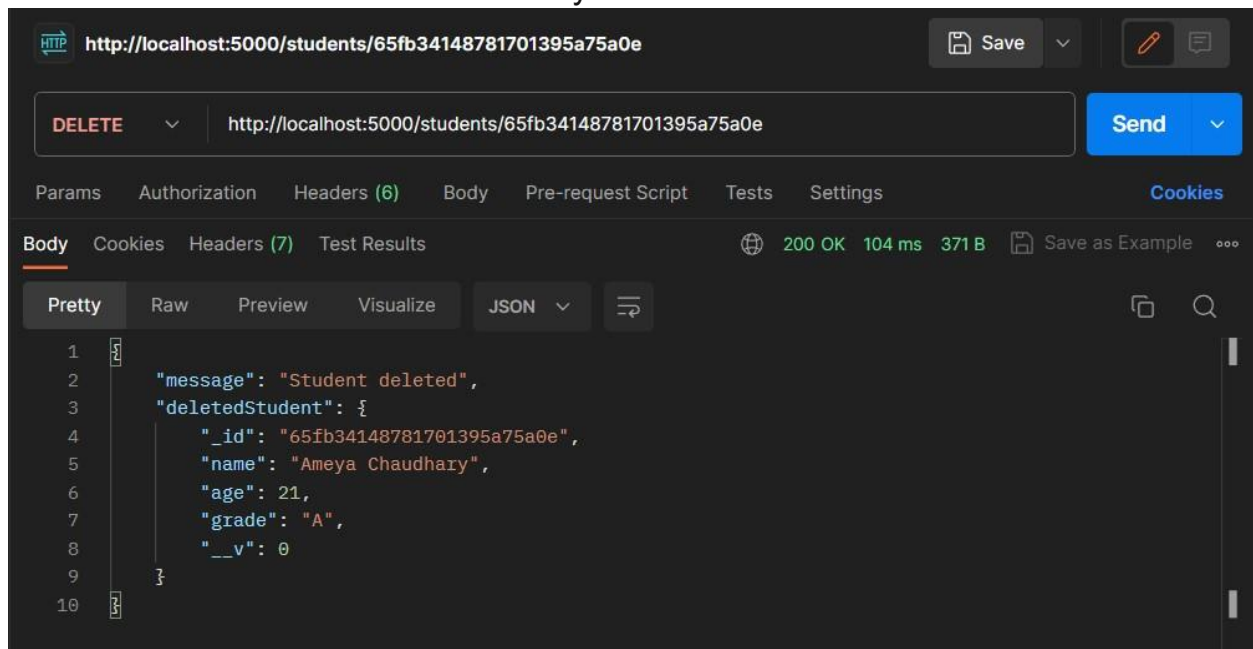
```
POST http://localhost:5000/students/

{"name": "Vrushabh Ghuse",
 "age": "22",
 "grade": "A"}
```

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.

Conclusion: From the above experiment, we have successfully implemented mongodb using shell and created database and also connected database using restful api and fetch and updated database using get, post, put delete and performed all problem statements successfully.