Name - Gaurang A Raorane Div - D15A Roll no - 49 Batch - C

**Experiment - 4**

**Aim:-** To study CRUD operations in MongoDB

**Theory:-**

A) Describe some of the features of MongoDB?

MongoDB, a NoSQL database, offers several features that make it popular among developers:

* **Document-Oriented:** MongoDB stores data in flexible, JSON-like documents, making it easy to represent complex hierarchical relationships.
* **Schema-less:** Unlike traditional SQL databases, MongoDB doesn't require a predefined schema, allowing for dynamic and flexible data modeling.
* **Scalability:** MongoDB can scale horizontally across multiple servers with ease, allowing for high availability and performance.
* **High Performance:** It provides fast read and write operations due to its memory-mapped storage engine and support for indexing.
* **Rich Query Language:** MongoDB supports a powerful query language with features like secondary indexes, sorting, and aggregation pipelines.
* **Replication and Failover:** MongoDB offers automatic data replication and failover, ensuring data durability and high availability.
* **Geospatial Indexing:** It supports geospatial queries, making it suitable for location-based applications.
* **Built-in Sharding:** MongoDB includes built-in sharding capabilities for distributing data across multiple servers, enabling horizontal scaling.

B) What are Documents and Collections in MongoDB?

* **Documents:** In MongoDB, a document is a JSON-like data structure that stores data in key-value pairs. Documents are analogous to rows in relational databases but offer more flexibility, as they can have nested structures.
* **Collections:** A collection is a grouping of MongoDB documents. It is analogous to a table in relational databases. Collections in MongoDB don't enforce a schema, allowing documents within a collection to have varying structures.

### C) When to use MongoDB?

MongoDB is suitable for various use cases, including:

* **Document Storage:** Applications with complex, hierarchical data structures benefit from MongoDB's document-oriented storage model.
* **Big Data:** MongoDB's scalability and performance make it well-suited for handling large volumes of data.
* **Real-Time Analytics:** Its ability to handle high-throughput read and write operations makes it suitable for real-time analytics and logging applications.
* **Content Management Systems:** MongoDB's flexible schema and ease of scaling make it ideal for content management systems and web applications.

D) What is Sharding in MongoDB?

Sharding in MongoDB is a technique for distributing data across multiple servers to improve scalability and performance. It involves partitioning data into smaller chunks called shards and distributing these chunks across different servers. Sharding allows MongoDB to handle large volumes of data by distributing the workload across multiple machines.

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

Mongoose is an Object Data Modeling (ODM) library for MongoDB in Node.js. It simplifies interactions with MongoDB by providing a schema-based solution and abstraction over raw MongoDB queries. In building a RESTful API with MongoDB, Mongoose plays a crucial role in defining data models, enforcing schemas, performing validation, and simplifying CRUD operations. It abstracts away the complexity of interacting with MongoDB directly and provides a more structured approach to building APIs, which results in faster development and easier maintenance.

F) List features of REST architectural style.

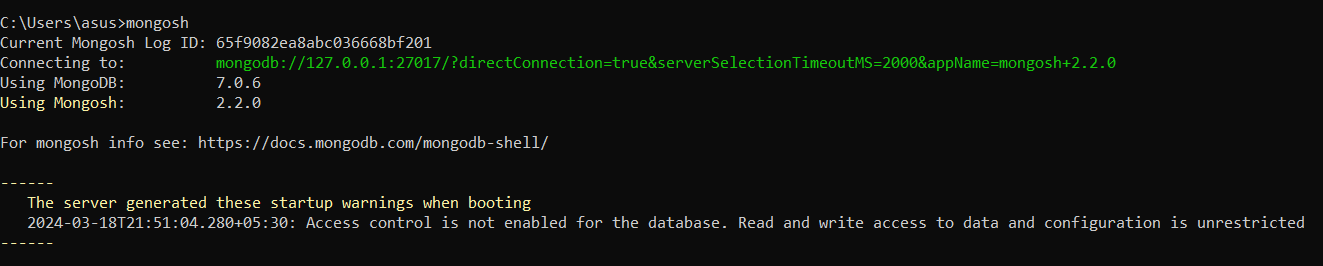
* **Statelessness:** Each request from a client to the server must contain all the information necessary to understand and fulfill the request, making the server stateless and scalable.
* **Uniform Interface:** RESTful APIs expose resources using standard HTTP methods (GET, POST, PUT, DELETE) and use URIs (Uniform Resource Identifiers) to identify resources.
* **Client-Server Architecture:** REST separates the client and server, allowing them to evolve independently. Clients are not concerned with the server's implementation details, and servers are not concerned with the client's state.
* **Cacheability:** Responses from a server can be marked as cacheable or non-cacheable, improving performance and scalability.
* **Layered System:** REST allows for a hierarchical system of layers, where each layer provides a specific functionality without knowing the details of other layers. This enhances scalability and simplifies system architecture.

G) What are the advantages of using APIs?

* **Modularity:** APIs allow for modular development, where different components of an application can be developed, tested, and deployed independently.
* **Interoperability:** APIs enable communication and data exchange between different systems and platforms, promoting interoperability and integration.
* **Scalability:** By providing a standardized interface, APIs facilitate scaling applications horizontally and vertically, accommodating growing user bases and increasing data volumes.
* **Flexibility:** APIs provide flexibility by allowing developers to access and manipulate data and functionality in various ways, depending on their requirements.
* **Ecosystem Growth:** APIs encourage the growth of an ecosystem around a platform or service by enabling third-party developers to build upon existing functionality, leading to innovation and expansion of capabilities.

**Output:-**

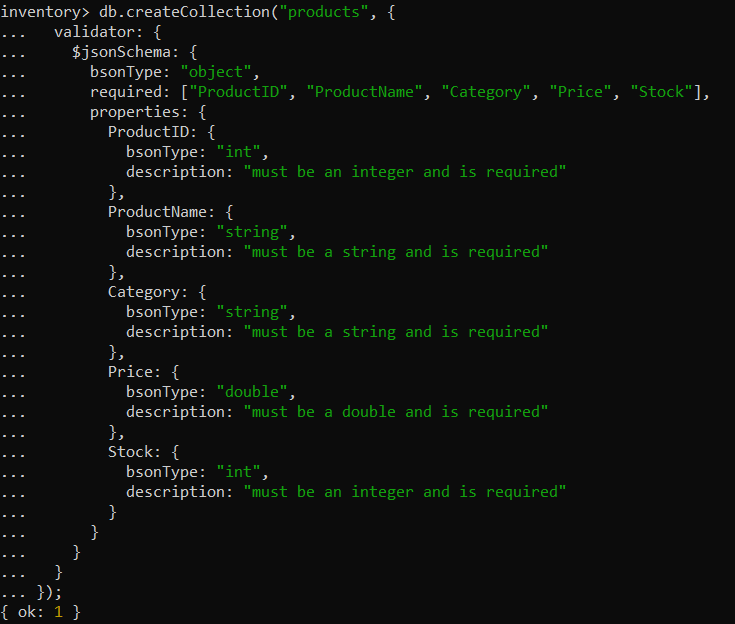
MongoShell is set up and initialized.



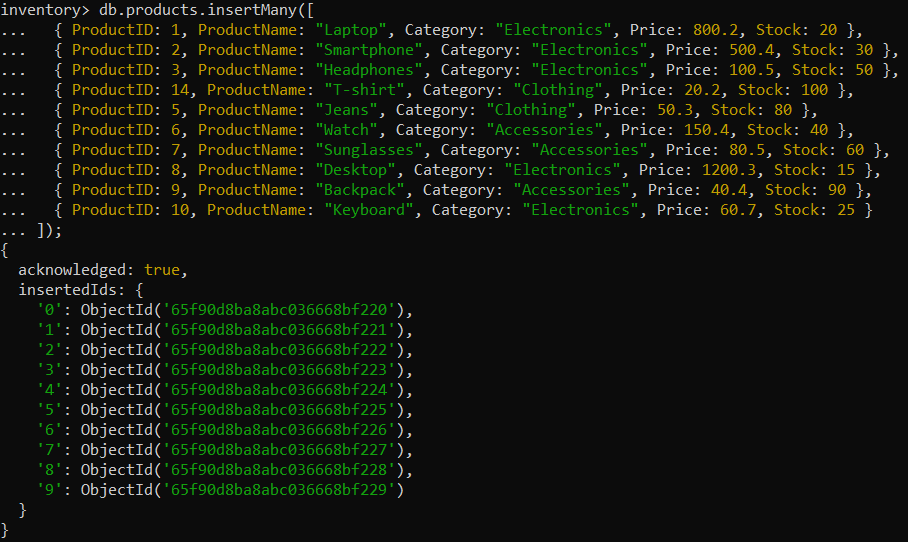
Create a database named "inventory".



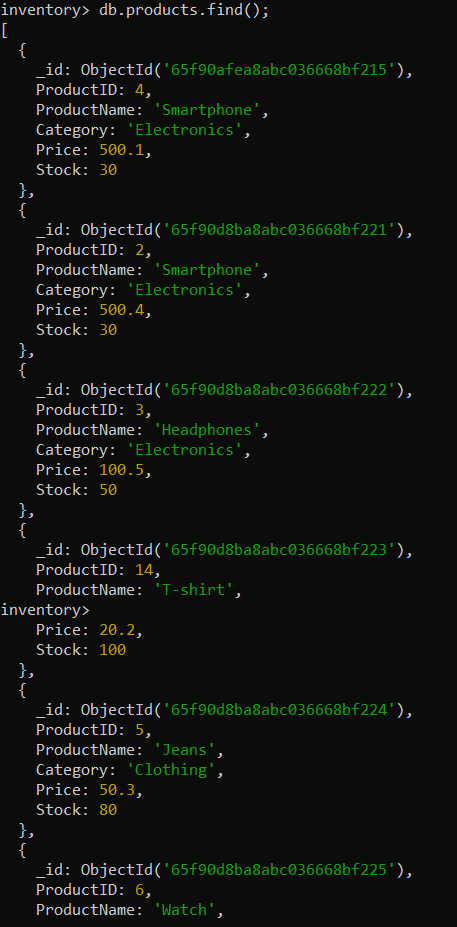
Create a collection named "products" with the fields: (ProductID, ProductName, Category, Price, Stock).



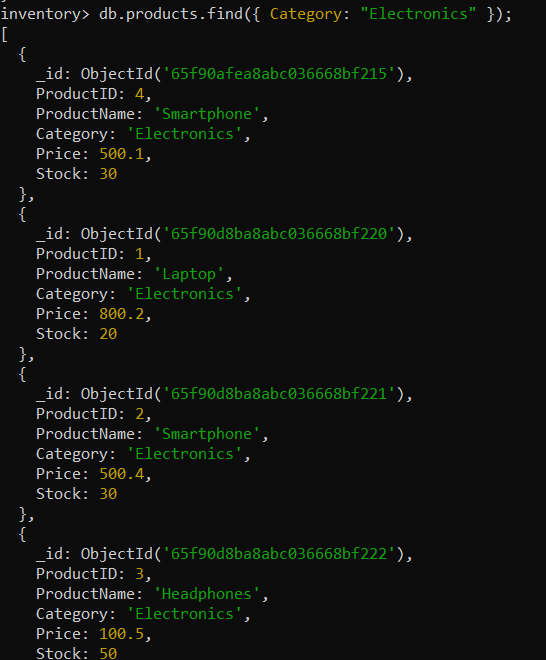
Insert 10 documents into the "products" collection.



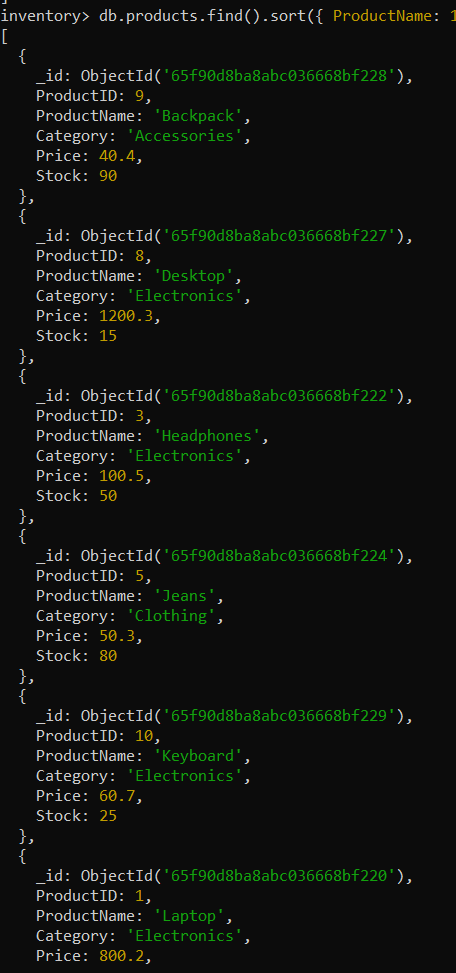
Display all the documents in the "products" collection.



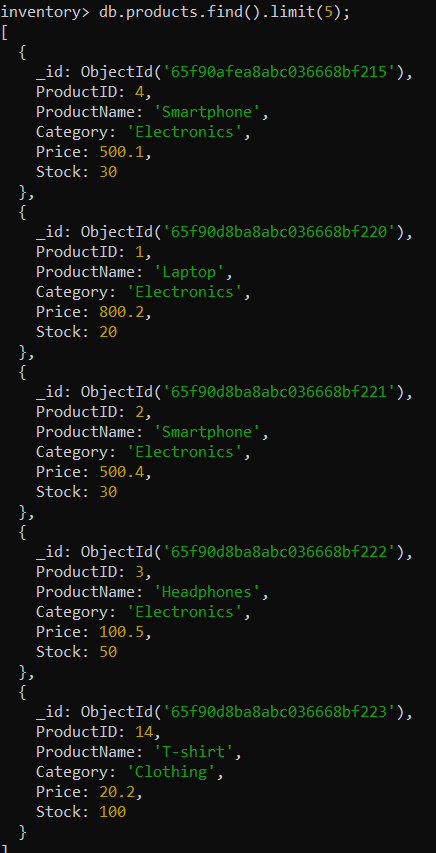
Display all the products in the "Electronics" category.



Display all the products in ascending order of their names.



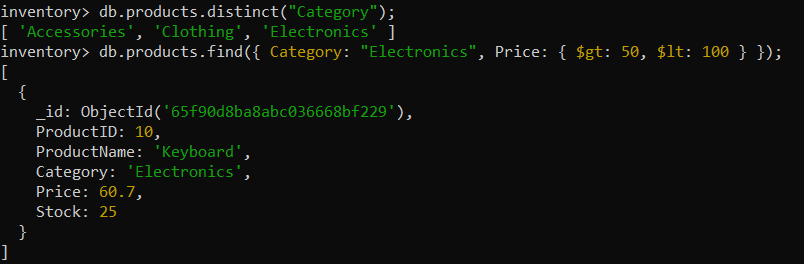
Display the details of the first 5 products.



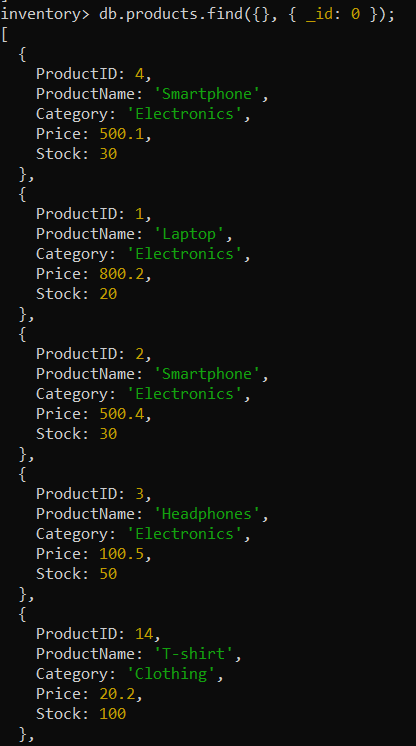
Display the categories of products with a specific name.



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



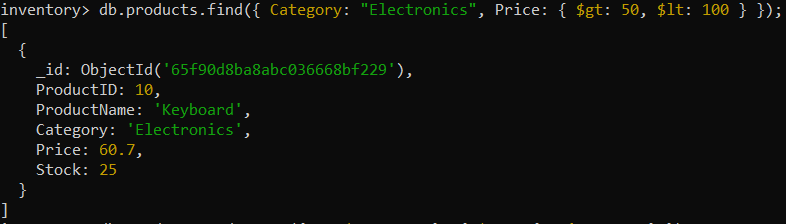
Display all the products without showing the "\_id" field.



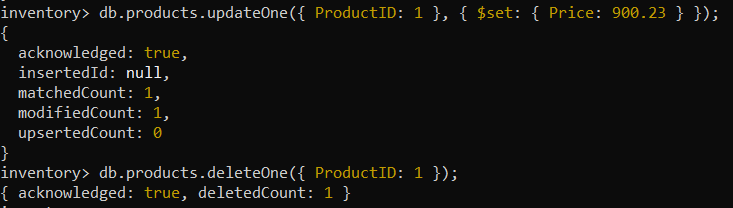
Display all the distinct categories of products.



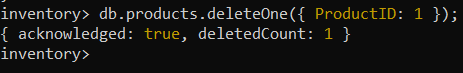
Display products in the "Electronics" category with prices greater than 50 but less than 100.



Change the price of a product.



Delete a particular product entry.



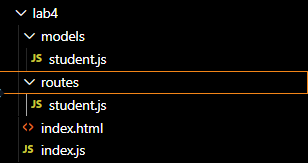
B) Create a set of RESTful endpoints using Node.js, Express, and Mongoose for handling student data operations.

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.

File structure:-



Input:-

**models/student.js**

const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({

name: String,

age: Number,

grade: String

});

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

module.exports = Student;

**routes/student.js**

const express = require('express');

const router = express.Router();

const Student = require('../models/student');

router.get('/', async (req, res) => {

try {

const students = await Student.find();

res.json(students);

} catch (err) {

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

}

});

router.get('/:id', getStudent, (req, res) => {

res.json(res.student);

});

router.post('/', 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 (err) {

res.status(400).json({ message: err.message });

}

});

router.patch('/:id', getStudent, async (req, res) => {

if (req.body.name != null) {

res.student.name = req.body.name;

}

if (req.body.age != null) {

res.student.age = req.body.age;

}

if (req.body.grade != null) {

res.student.grade = req.body.grade;

}

try {

const updatedStudent = await res.student.save();

res.json(updatedStudent);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

router.delete('/:id', getStudent, async (req, res) => {

try {

await res.student.remove();

res.json({ message: 'Student deleted' });

} catch (err) {

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

}

});

async function getStudent(req, res, next) {

let student;

try {

student = await Student.findById(req.params.id);

if (student == null) {

return res.status(404).json({ message: 'Student not found' });

}

} catch (err) {

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

}

res.student = student;

next();

}

module.exports = router;

**index.js**

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const cors = require('cors');

const studentRoutes = require('./routes/student');

const app = express();

app.use(cors());

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

mongoose.connect('mongodb://localhost:27017/studentDB', {

useNewUrlParser: true,

useUnifiedTopology: true

});

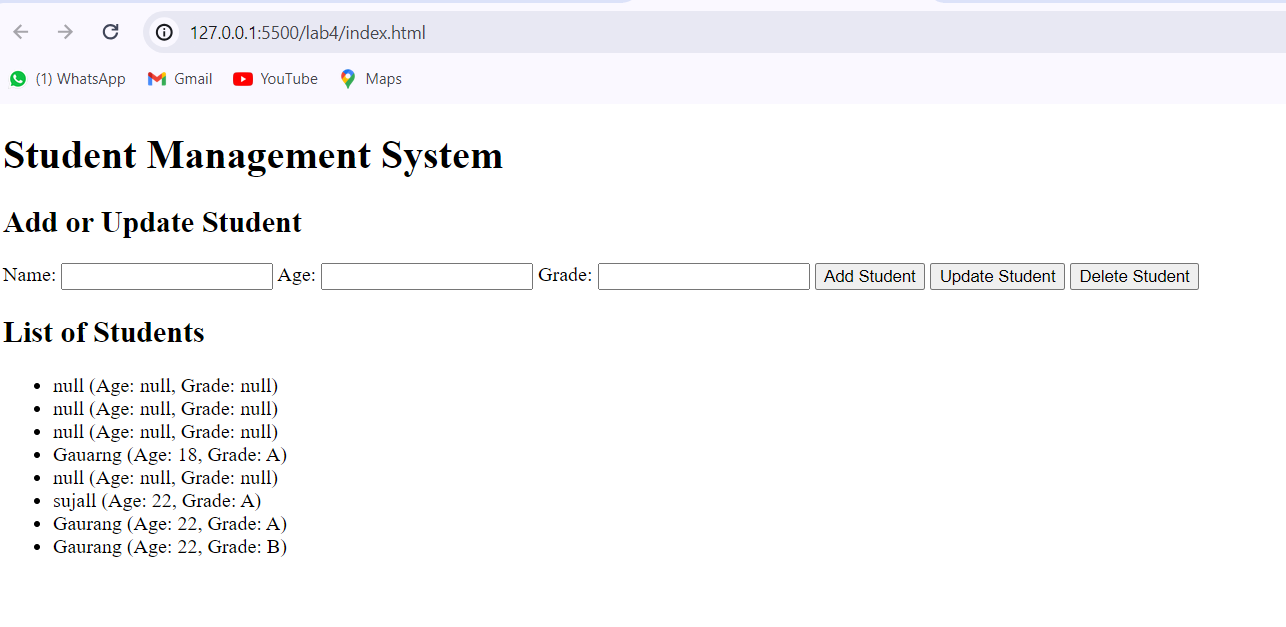
app.use(bodyParser.json());

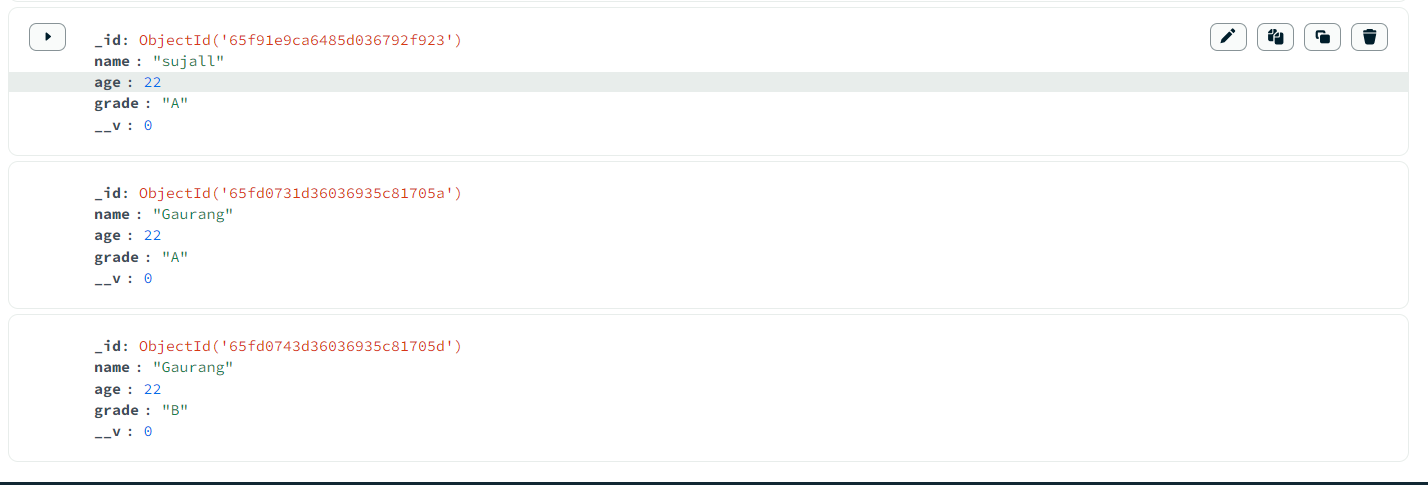
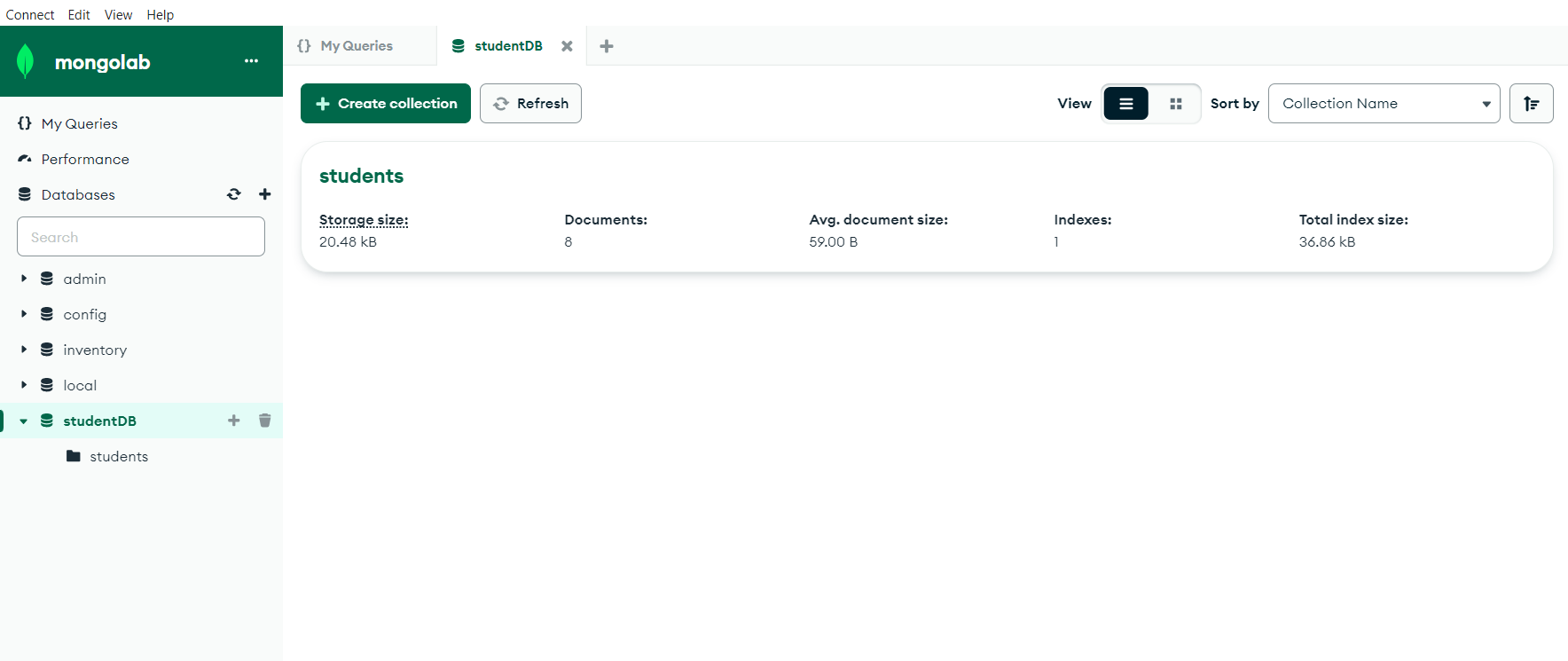
app.use('/students', studentRoutes);

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

**Output:-**   




Conclusion:- We studied about the CRUD operations in MongoDB and built RESTAPI for simple student database system.