Name: Kaushik Kotian Roll no.:30 Div : D15B Batch : B

WEBX Lab Experiment - 4

Aim : To understand and implement MongoDB and RESTful API by:

a. Building a RESTful API using MongoDB.
b. Implementation of all CRUD operations in MongoDB

Theory:

1. MongoDB, a leading NoSQL database management system, is renowned for its scalability, flexibility, and high performance. It stores data in BSON (Binary JSON) format, facilitating dynamic schemas and seamless integration with modern web applications.

2. MongoDB vs SQL:

a. MongoDB and SQL databases represent two distinct approaches to storing and managing data, each with its own set of characteristics and use cases. Here, we'll delve into the key differences between MongoDB, a NoSQL database, and traditional SQL databases:

b. Data Model:

i. SQL (Structured Query Language) Databases:
SQL databases are relational databases, where data is organized into tables with predefined schemas. Each table consists of rows and columns, and relationships between tables are established using foreign key constraints.

ii. MongoDB:
MongoDB is a NoSQL database, which means it employs a non-relational data model. Data in MongoDB is stored as flexible, schema-less documents in BSON format (Binary JSON), akin to JSON objects. These documents are organized into collections, and there is no need for predefined schemas. This flexibility allows for dynamic and hierarchical data structures, making MongoDB suitable for handling unstructured or semi-structured data.

c. Scalability:

i. SQL Databases:

Traditional SQL databases typically scale vertically by adding more resources (CPU, RAM) to a single server. This approach has limits in terms of scalability and may result in performance bottlenecks as data volumes grow.

ii. MongoDB:

MongoDB is designed to scale horizontally across clusters of commodity servers. It employs sharding, a technique that partitions data across multiple servers, allowing for seamless expansion and improved scalability as data volumes increase. This makes MongoDB well-suited for handling large-scale, distributed applications.

d. Query Language:

i. SQL Databases:

SQL databases use the structured query language (SQL) for querying and manipulating data. SQL provides a standardized syntax for performing operations such as SELECT, INSERT, UPDATE, and DELETE.

ii. MongoDB:

MongoDB uses a query language that is based on JavaScript and JSON-like syntax. It offers a rich set of query operators and aggregation pipelines for querying and manipulating data. While MongoDB's query language is more flexible in some regards, it may require a learning curve for developers accustomed to SQL.

e. Transactions and Atomicity:

i. SQL Databases:

SQL databases support ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring that database operations are atomic and consistent. Transactions allow multiple database operations to be grouped together as a single unit of work, ensuring data integrity.

ii. MongoDB:

MongoDB introduced multi-document transactions in recent versions, providing support for ACID transactions across multiple documents within a single replica set or sharded cluster. This enhances data integrity and consistency, making MongoDB more suitable for transactional workloads.

f. Use Cases:

i. SQL Databases:
SQL databases are well-suited for applications with structured data and complex relationships, such as traditional transactional systems, financial applications, and enterprise resource planning (ERP) systems.

ii. MongoDB:
MongoDB is particularly useful for applications requiring flexibility, scalability, and real-time analytics, such as content management systems, e-commerce platforms, social media analytics, and IoT (Internet of Things) applications.

3. In summary, MongoDB and SQL databases differ in their data models, scalability approaches, query languages, transaction support, and use cases. Choosing between them depends on the specific requirements and characteristics of the application being developed.

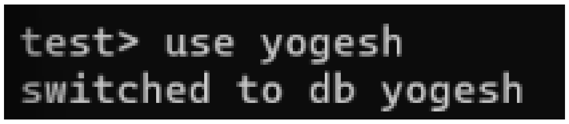
Implementation:

TASK 1: Implementing all CRUD operations for MongoDB

1. **Create Database**

   Syntax: use <database_name>

   This command is used to switch to the specified database or create a new database if it doesn't exist.

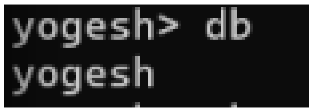


```
test> use yogesh
switched to db yogesh
```

2. **To check currently selected database**

   Syntax: db

   This command returns the name of the currently selected database.

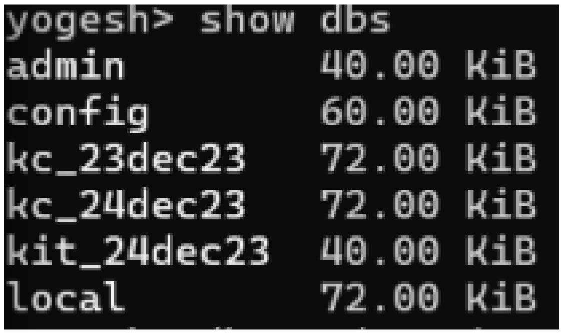


```
yogesh> db
yogesh
```

3. **To check database list**

   Syntax: show dbs

   This command lists all the databases present on the MongoDB server.

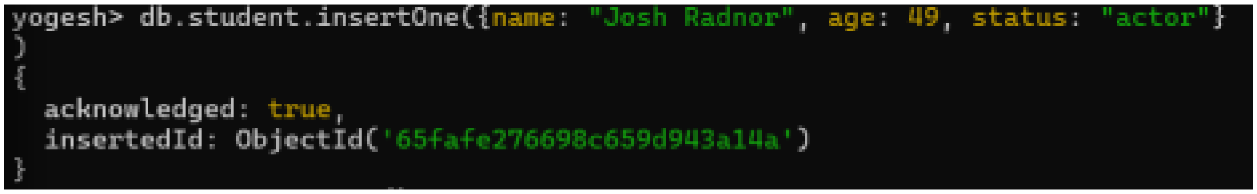


```
yogesh> show dbs
admin          40.00 KiB
config         60.00 KiB
kc_23dec23     72.00 KiB
kc_24dec23     72.00 KiB
kit_24dec23    40.00 KiB
local          72.00 KiB
```

4. **To insert a document in database**

   Syntax: db.collection.insertOne({document})

   Inserts a single document into the specified collection within the current database.

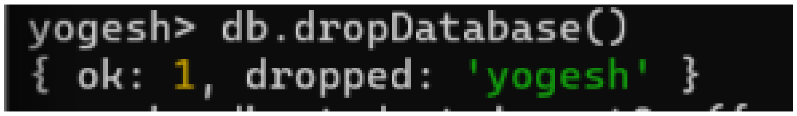


```
yogesh> db.student.insertOne({name: "Josh Radnor", age: 49, status: "actor"}
)
{
  acknowledged: true,
  insertedId: ObjectId('65fafe276698c659d943a14a')
}
```

5. To drop a database

Syntax: db.dropDatabase()

Deletes the currently selected database along with all its associated collections.

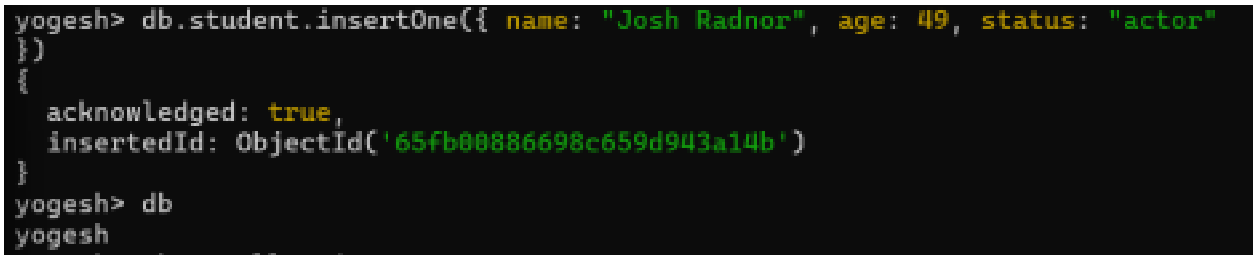


```
yogesh> db.dropDatabase()
{ ok: 1, dropped: 'yogesh' }
```

6. To create a collection and display the current database name

Syntax: db.createCollection("<collection_name>")

Creates a new collection with the specified name within the current database and returns the database name.

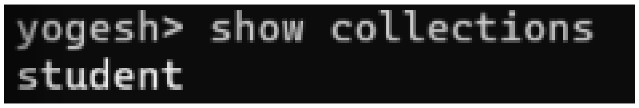


```
yogesh> db.student.insertOne({ name: "Josh Radnor", age: 49, status: "actor"
})
{
  acknowledged: true,
  insertedId: ObjectId('65fb00886698c659d943a14b')
}
yogesh> db
yogesh
```

7. To show existing collections

Syntax: show collections

Lists all the collections present in the current database.

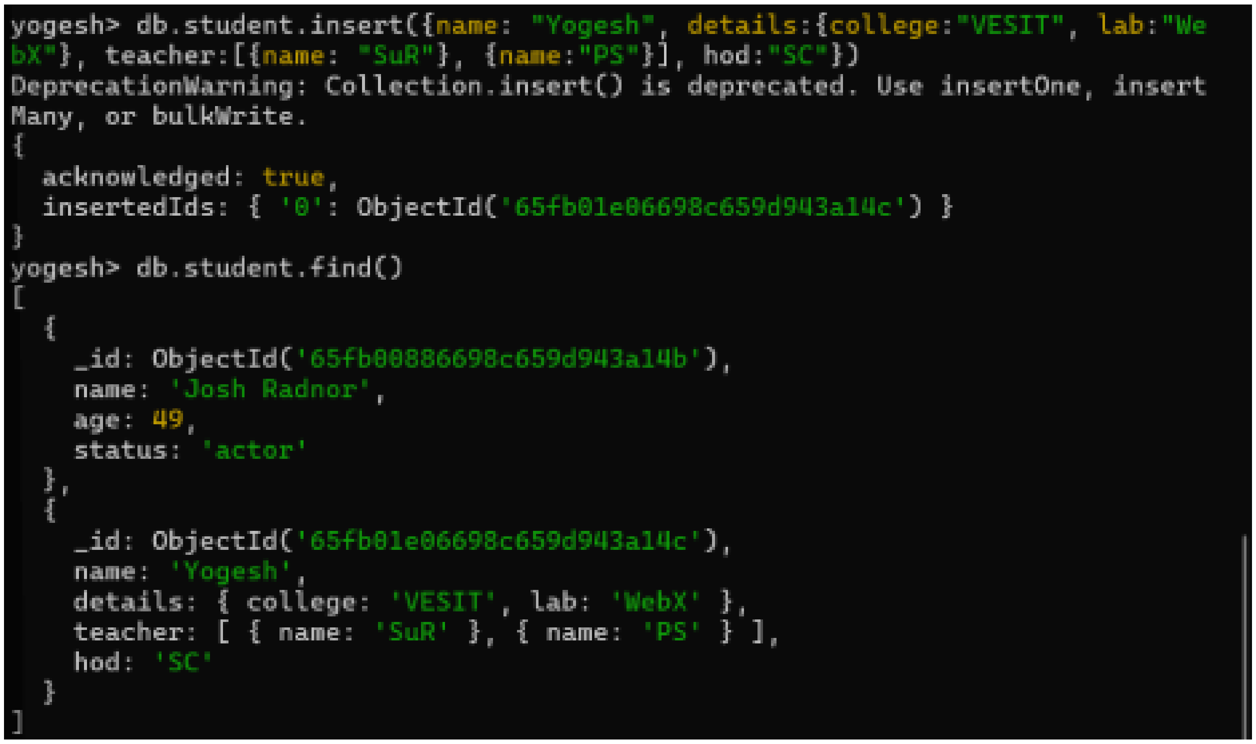


```
yogesh> show collections
student
```

8. To insert a document and check the inserted document

Syntax: db.collection.insertOne({document}) followed by db.collection.findOne()

Inserts a single document into the specified collection and then retrieves and displays the first document in the collection.

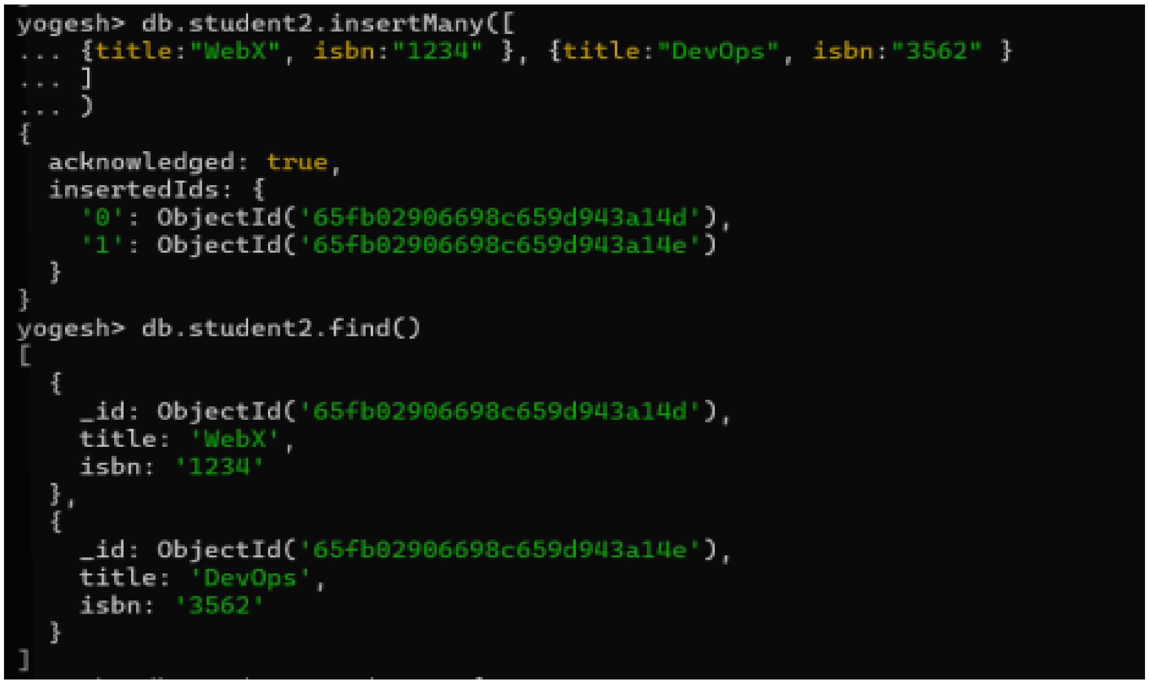


```
yogesh> db.student.insert({name: "Yogesh", details:{college:"VESIT", lab:"We
bX"}, teacher:[{name: "SuR"}, {name:"PS"}], hod:"SC"})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insert
Many, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('65fb01e06698c659d943a14c') }
}
yogesh> db.student.find()
[
  {
    _id: ObjectId('65fb00886698c659d943a14b'),
    name: 'Josh Radnor',
    age: 49,
    status: 'actor'
  },
  {
    _id: ObjectId('65fb01e06698c659d943a14c'),
    name: 'Yogesh',
    details: { college: 'VESIT', lab: 'WebX' },
    teacher: [ { name: 'SuR' }, { name: 'PS' } ],
    hod: 'SC'
  }
]
```

9. To insert many parameters

Syntax: db.collection.insertMany([{document1}, {document2}, ...])

Inserts multiple documents into the specified collection in a single operation.

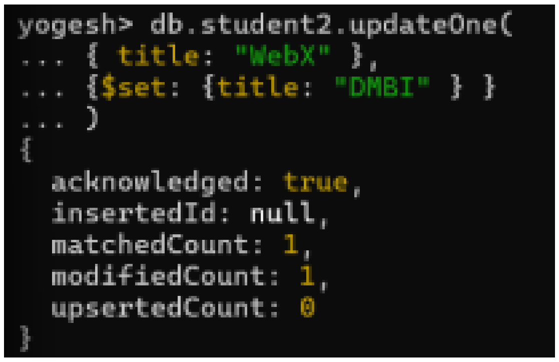


```
yogesh> db.student2.insertMany([
... {title:"WebX", isbn:"1234" }, {title:"DevOps", isbn:"3562" }
... ]
... )
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65fb02906698c659d943a14d'),
    '1': ObjectId('65fb02906698c659d943a14e')
  }
}
yogesh> db.student2.find()
[
  {
    _id: ObjectId('65fb02906698c659d943a14d'),
    title: 'WebX',
    isbn: '1234'
  },
  {
    _id: ObjectId('65fb02906698c659d943a14e'),
    title: 'DevOps',
    isbn: '3562'
  }
]
```

10.To update

Syntax: db.collection.updateOne(<filter>, <update>, <options>)

Updates the first document that matches the specified filter with the given update.

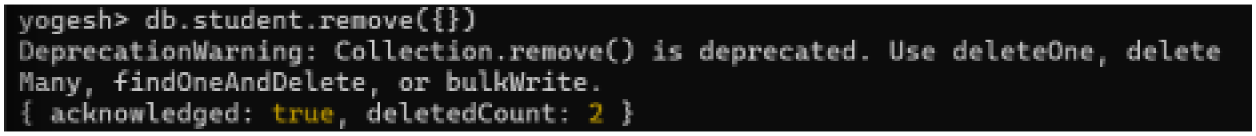


```
yogesh> db.student2.updateOne(
... { title: "WebX" },
... {$set: {title: "DMBI" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

11. To remove a collection

Syntax: db.collection.drop()

Deletes the specified collection from the current database.

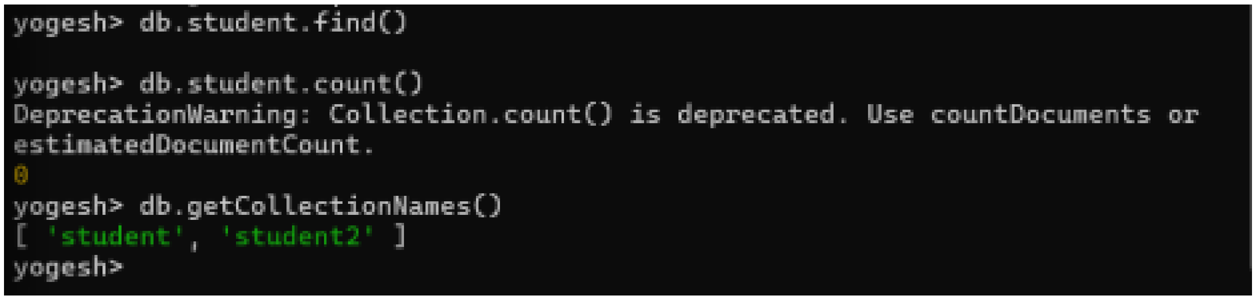


```
yogesh> db.student.remove({})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, delete
Many, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 2 }
```

12.Other common operations on a collection

Syntax: Various operations like find(), updateMany(), deleteOne(), etc.

These operations are used to perform CRUD operations and other manipulations on documents within a collection.



```
yogesh> db.student.find()

yogesh> db.student.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or
estimatedDocumentCount.
0
yogesh> db.getCollectionNames()
[ 'student', 'student2' ]
yogesh>
```

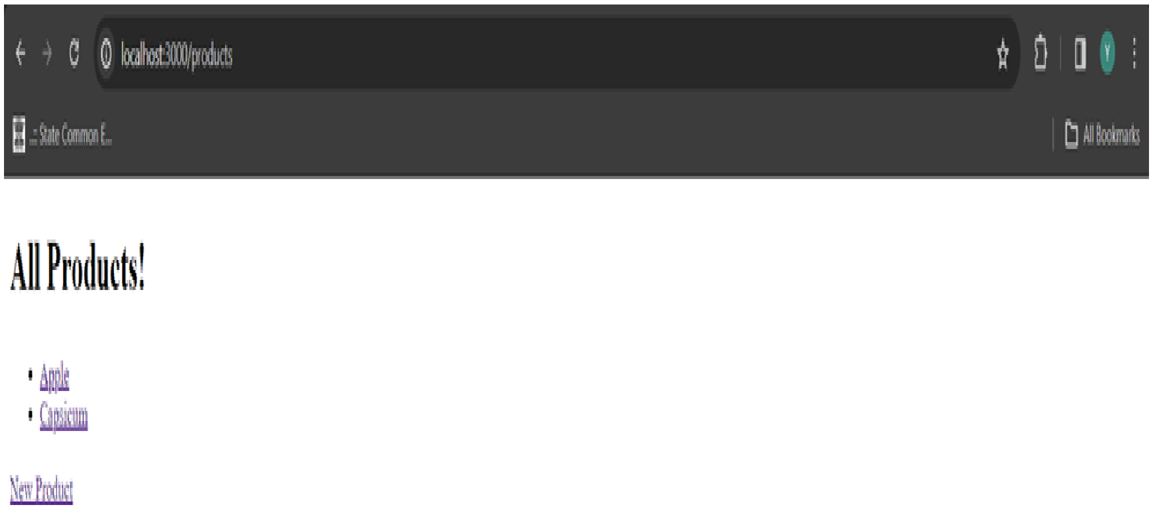
TASK2: Building a RESTful API using MongoDB.

index.js:

```
    .then(() => {
        console.log("MONGO CONNECTION OPEN!!!")
    })
    .catch(err => {
        console.log("OH NO MONGO CONNECTION ERROR!!!!")
        console.log(err)
    })

app.set('views', path.join(
____dirname, 'views')); app.set('view engine',
'ejs');


app.use(express.urlencoded({ extended: true }));
app.use(methodOverride('_method'))


const categories = ['fruit', 'vegetable', 'dairy'];


    app.get('/products', async (req, res) =>
        { const { category } = req.query;

    if (category) {

        const products = await Product.find({ category })
        res.render('products/index', { products, category })

    } else {

        const products = await Product.find({})
        res.render('products/index', { products, category: 'All'
})

    }

})


    app.get('/products/new', (req, res) => {
        res.render('products/new', { categories })

})


app.post('/products', async (req, res) => {

    const newProduct = new Product(req.body);
    await newProduct.save();
    res.redirect(`/products/${newProduct._id}`
    )

})

    app.get('/products/:id', async (req, res)
        => { const { id } = req.params;

    const product = await Product.findById(id)
    res.render('products/show', { product })

})

    app.get('/products/:id/edit', async (req, res) =>
        { const { id } = req.params;

    const product = await Product.findById(id);
    res.render('products/edit', { product, categories })

})
```

```
    app.put('/products/:id', async (req, res)
        => { const { id } = req.params;
    const product = await
Product.findByIdAndUpdate(id, req.body, {
runValidators: true, new: true });

    res.redirect(`/products/${product._id}`);

})
    app.delete('/products/:id', async (req, res) =>
        { const { id } = req.params;

    const deletedProduct = await Product.findByIdAndDelete(id);
    res.redirect('/products');

})

app.listen(3000, () => {
    console.log("APP IS LISTENING ON PORT 3000!")

})
```

a. **Previously added products**

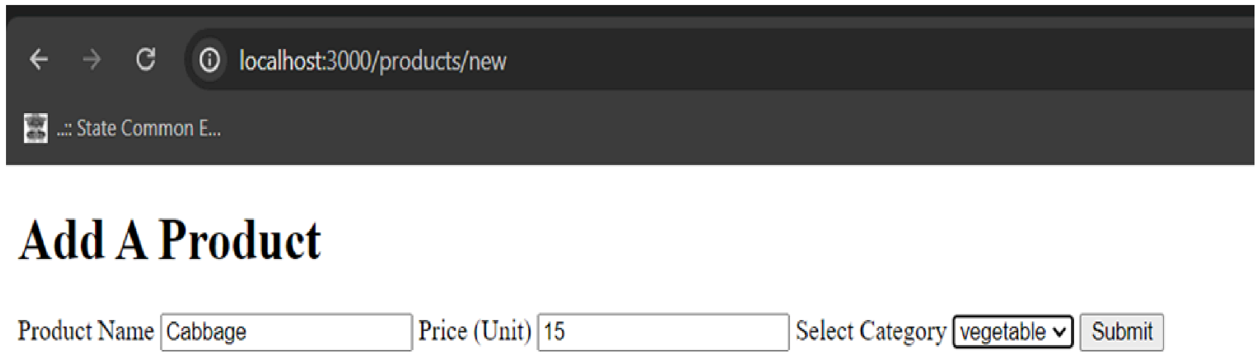


All Products!

- Apple
- Capsicum

New Product

b. **Adding a new product**

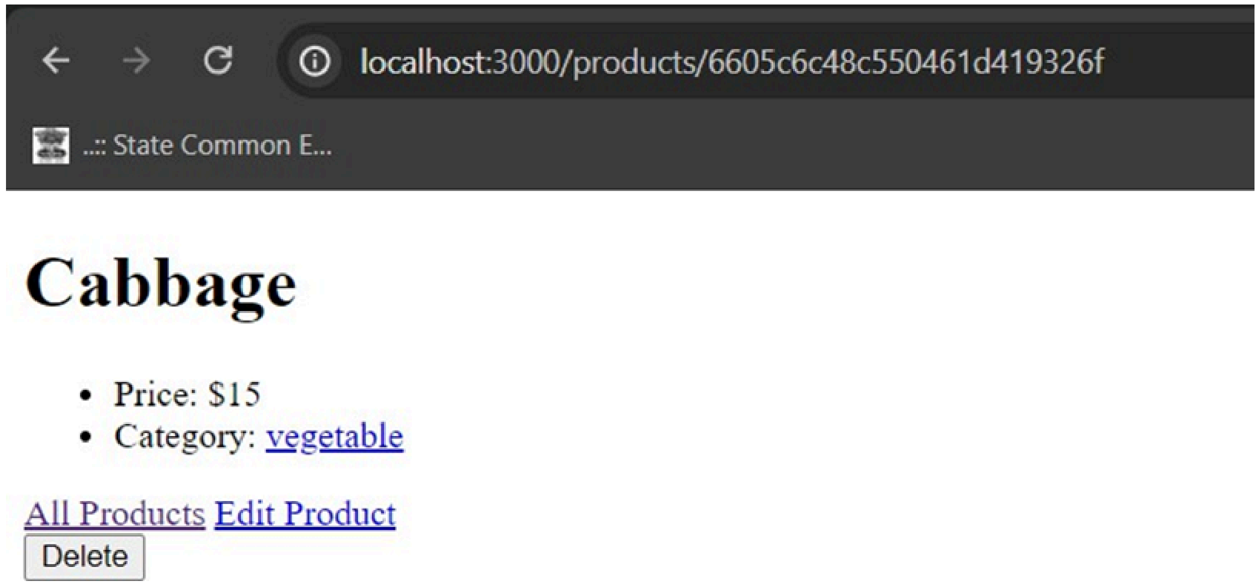


localhost:3000/products/new

..:: State Common E...

Add A Product

Product Name: Cabbage Price (Unit): 15 Select Category: vegetable Submit



localhost:3000/products/6605c6c48c550461d419326f

..:: State Common E...

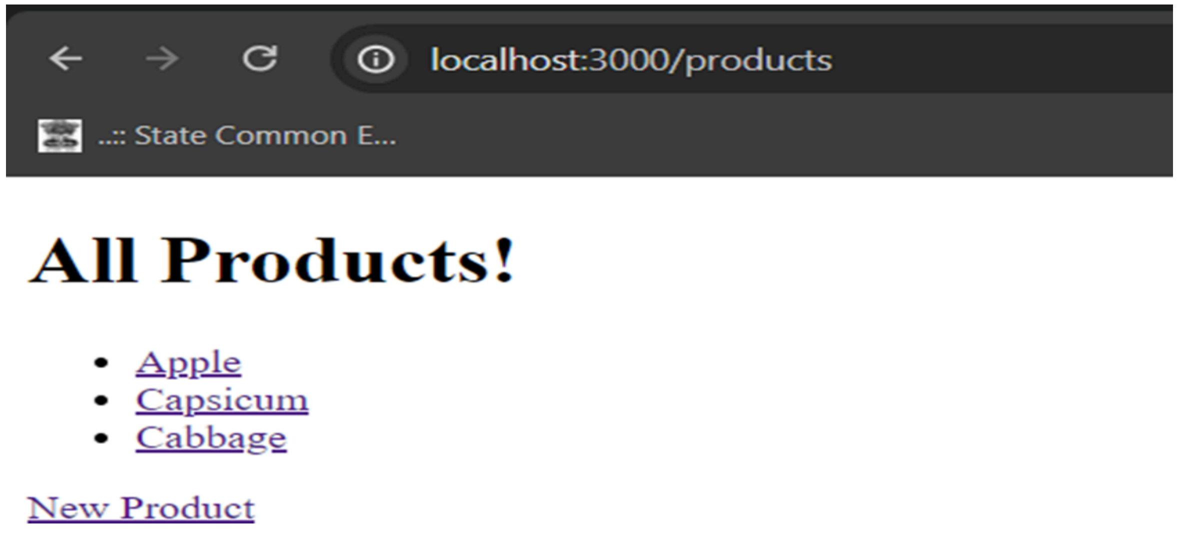
Cabbage

- Price: $15
- Category: vegetable

All Products Edit Product

Delete

c. **List of all products after adding a new item**



localhost:3000/products

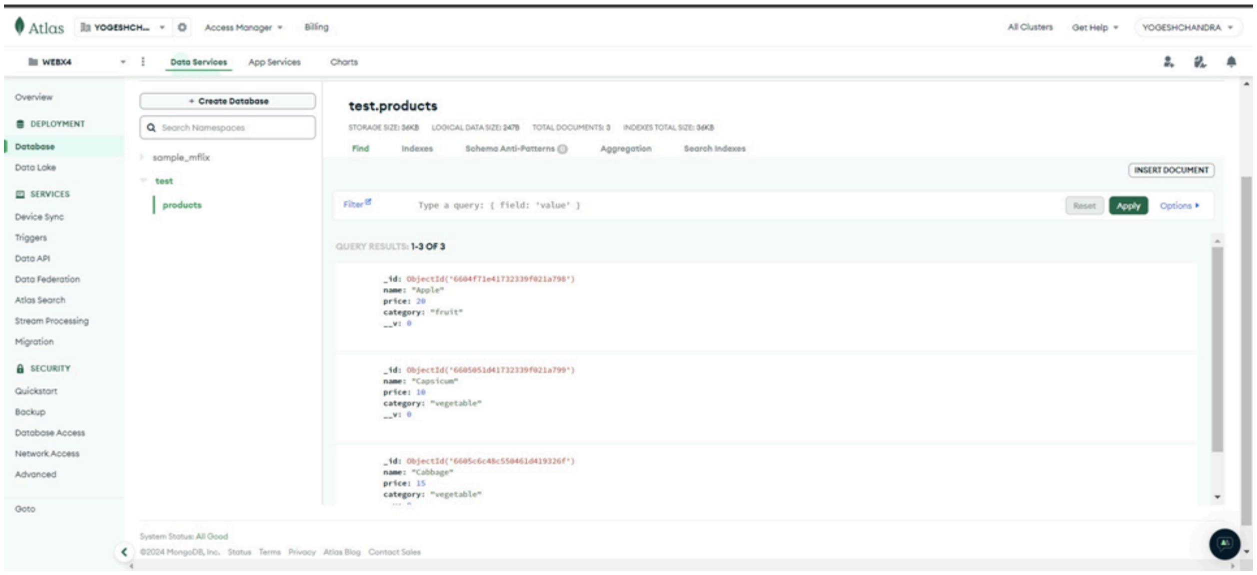
..:: State Common E...

All Products!

- Apple
- Capsicum
- Cabbage

New Product

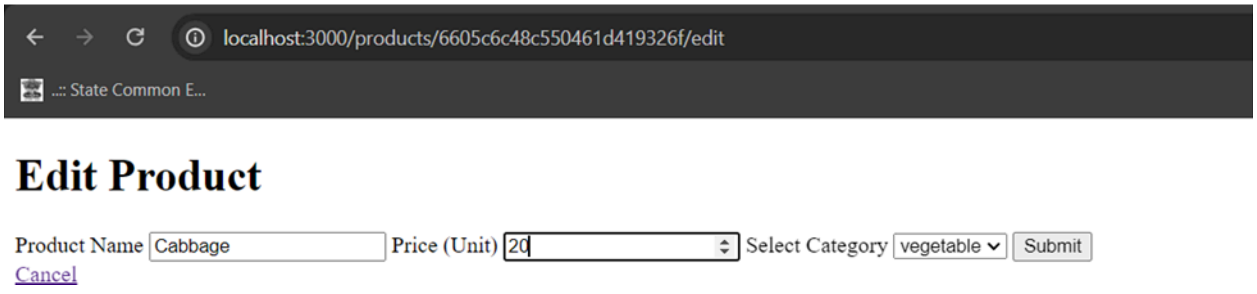
d. Changes reflected in MongoDB Atlas after adding a new product



QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('6604f71e41732339f021a798')
name: "Apple"
price: 20
category: "fruit"
__v: 0

_id: ObjectId('6605051d41732339f021a799')
name: "Capsicum"
price: 10
category: "vegetable"
__v: 0

_id: ObjectId('6605c6c48c550461d419326f')
name: "Cabbage"
price: 15
category: "vegetable"
```

e. Editing the recently added product

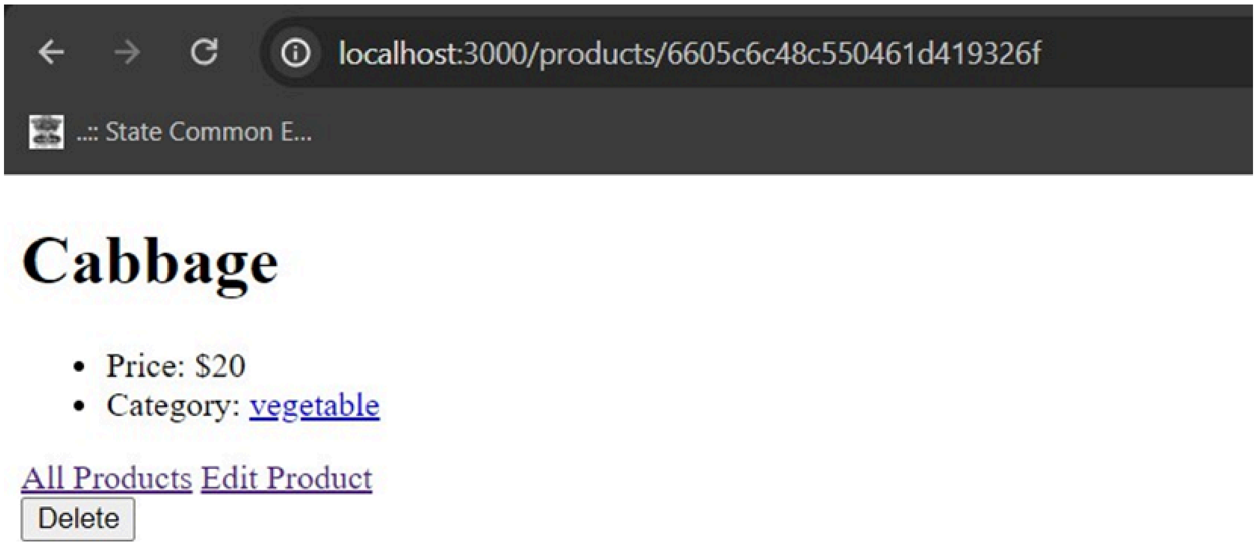


localhost:3000/products/6605c6c48c550461d419326f/edit

Edit Product

Product Name: Cabbage · Price (Unit): 20 · Select Category: vegetable · Submit

Cancel



localhost:3000/products/6605c6c48c550461d419326f

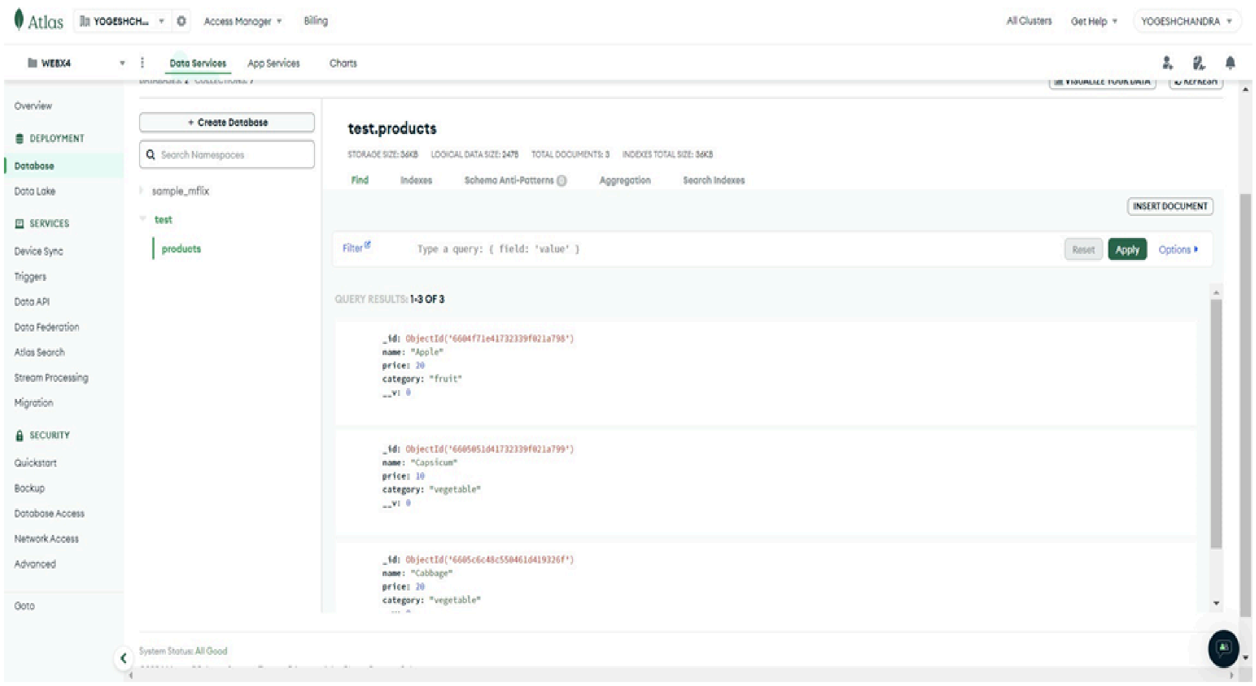
Cabbage

- Price: $20
- Category: vegetable

All Products Edit Product

Delete

f. Changes reflected in MongoDB Atlas after editing the added product



test.products

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 247B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 36KB

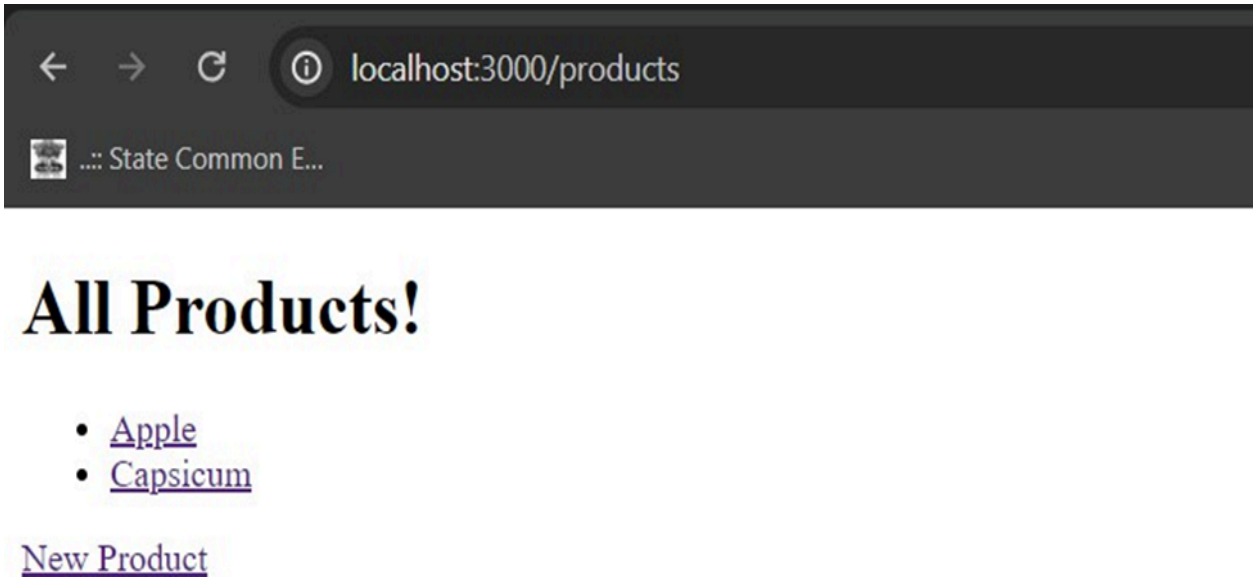
QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('6604f71e41732339f021a798')
name: "Apple"
price: 20
category: "fruit"
__v: 0
```

```
_id: ObjectId('6605051d41732339f021a799')
name: "Capsicum"
price: 10
category: "vegetable"
__v: 0
```

```
_id: ObjectId('6605c6c48c550461d419326f')
name: "Cabbage"
price: 20
category: "vegetable"
```

g. Deleting the added product

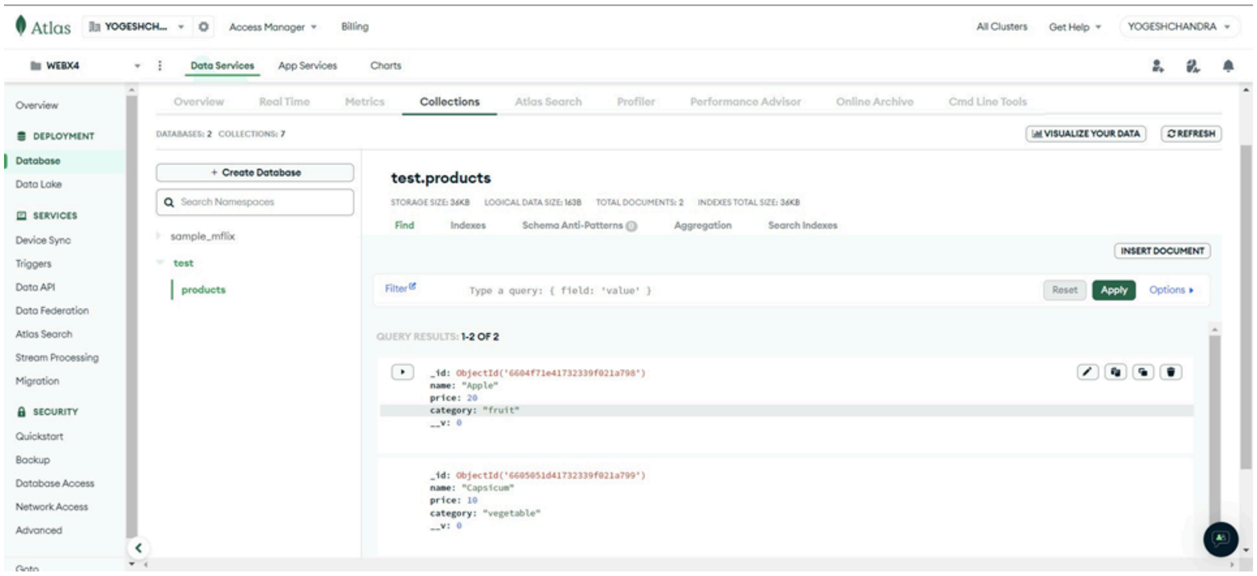


All Products!

- Apple
- Capsicum

New Product

h. Changes reflected in MongoDB Atlas after deleting the added product



QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('6604f71e41732339f021a798')
name: "Apple"
price: 20
category: "fruit"
__v: 0

_id: ObjectId('6605051d41732339f021a799')
name: "Capsicum"
price: 10
category: "vegetable"
__v: 0
```

Conclusion : In this experiment, the creation of a RESTful API with MongoDB showcased the robust capabilities of modern web development. Implementing CRUD operations provided a comprehensive understanding of data manipulation. Leveraging MongoDB's flexibility and scalability alongside RESTful architecture highlights the efficient management and accessibility of data in