**Batch: A-3 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of CRUD operations with Mongoose. |

**AIM:** **Demonstrate the use of Mongoose with CRUD operation.**

**Problem Definition:**

**1) Generate Database model**

**2) Create RESTful API**

**3) Demonstrate the Endpoints.**

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 2:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Mongoose CRUD Operations**

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides an easy-to-use interface for creating and managing MongoDB data through schemas and models, making it simpler to interact with MongoDB in a structured way. Here’s an overview of Mongoose CRUD (Create, Read, Update, Delete) operations:

* **Create**: To add new data, use Model.create() or new Model().save() methods.

javascript

Copy code

// Example: Adding a new user

const user = new User({ name: 'John Doe', email: 'john@example.com' });

user.save()

.then((savedUser) => console.log('User added:', savedUser))

.catch((error) => console.error(error));

* **Read**: To fetch data, use Model.find(), Model.findOne(), or Model.findById() methods.

javascript

Copy code

// Example: Finding all users

User.find()

.then((users) => console.log('Users:', users))

.catch((error) => console.error(error));

* **Update**: To modify existing data, use Model.updateOne(), Model.updateMany(), or Model.findByIdAndUpdate().

javascript

Copy code

// Example: Updating a user’s email

User.updateOne({ name: 'John Doe' }, { email: 'newemail@example.com' })

.then(() => console.log('User updated'))

.catch((error) => console.error(error));

* **Delete**: To remove data, use Model.deleteOne(), Model.deleteMany(), or Model.findByIdAndDelete().

javascript

Copy code

// Example: Deleting a user by ID

User.findByIdAndDelete(userId)

.then(() => console.log('User deleted'))

.catch((error) => console.error(error));

**RESTful API**

A RESTful API is an architectural style that allows communication between a client and a server over HTTP, using a standardized set of principles. In a RESTful API, each resource (e.g., users, products) is represented by a URL and manipulated through HTTP methods.

* **HTTP Methods**:
  + **GET**: Retrieves data (e.g., list of users).
  + **POST**: Creates a new resource (e.g., adding a new product).
  + **PUT** or **PATCH**: Updates an existing resource (e.g., modifying product details).
  + **DELETE**: Removes a resource (e.g., deleting a user).
* **Example of RESTful Endpoints**:
  + **GET /users**: Retrieves all users.
  + **POST /users**: Adds a new user.
  + **PUT /users/**

: Updates an existing user by ID.

* + **DELETE /users/**

: Deletes a user by ID.

* **Basic RESTful API with Express**:

javascript

Copy code

const express = require('express');

const app = express();

app.use(express.json());

// Example route for getting all users

app.get('/api/users', (req, res) => {

User.find()

.then((users) => res.json(users))

.catch((error) => res.status(500).json({ error: error.message }));

});

// Example route for adding a new user

app.post('/api/users', (req, res) => {

const newUser = new User(req.body);

newUser.save()

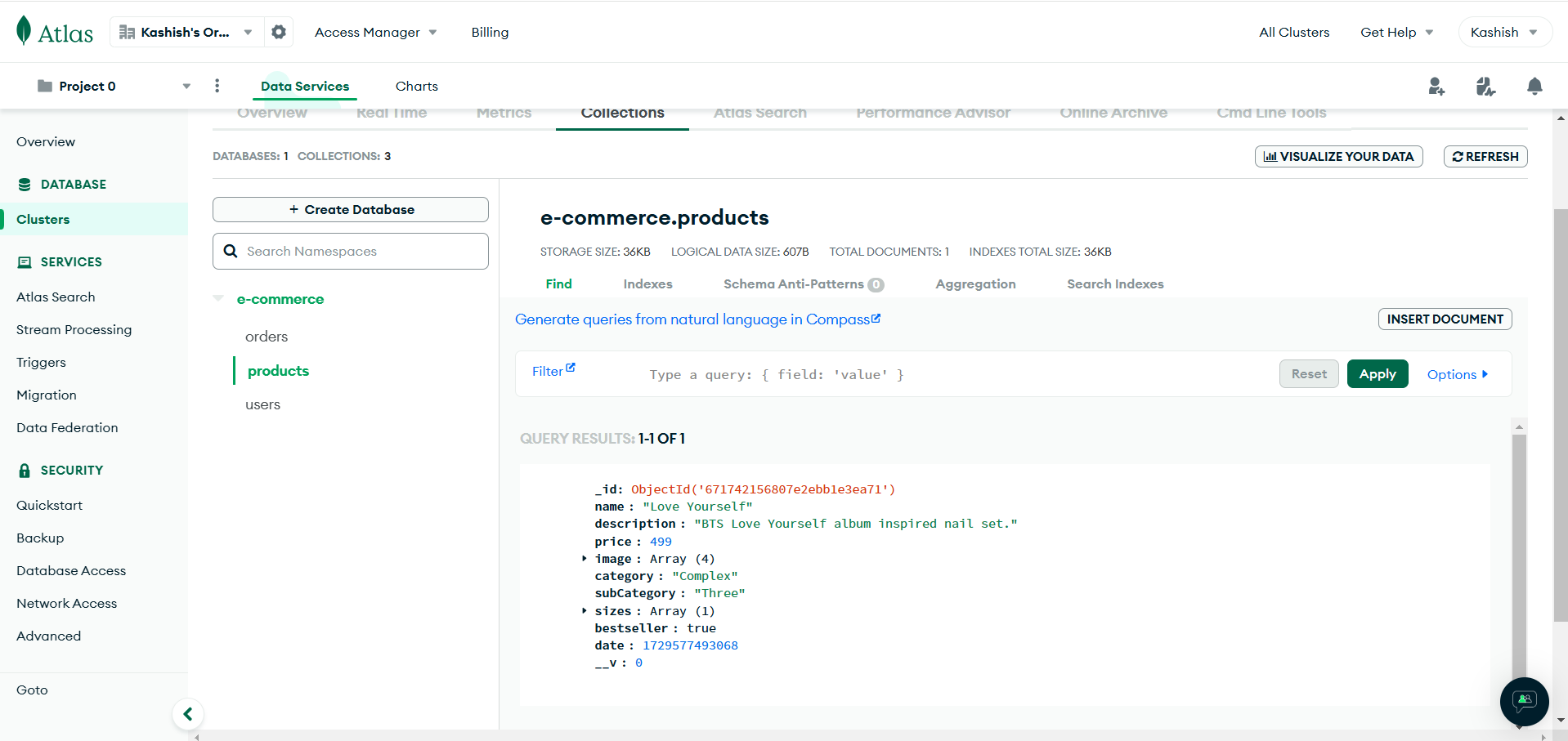
.then((user) => res.status(201).json(user))

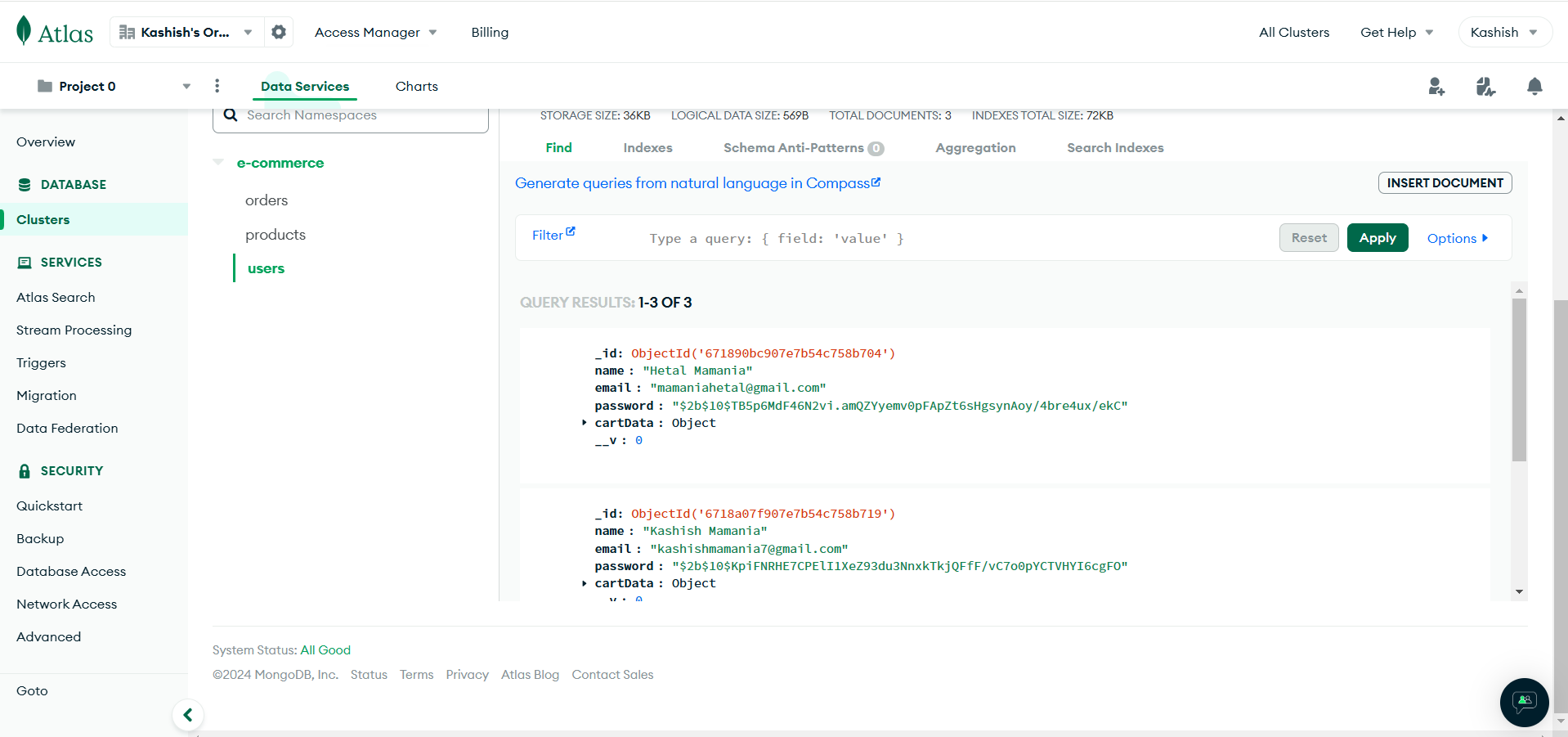
.catch((error) => res.status(500).json({ error: error.message }));

});

app.listen(3000, () => console.log('Server running on port 3000'));

**Methodology:**





**Conclusion:**