Name	Akshit Sharma
Sec	622 - B
UID	23BCS10929

EXPERIMENT 5.1

Title

CRUD Operations for Product Database Using Mongoose

Objective

Learn how to implement basic Create, Read, Update, and Delete (CRUD) operations on a MongoDB collection using Mongoose in Node.js. This task helps you understand schema design, database connectivity, and handling data in a structured, real-world manner.

Task Description

Create a Node.js application that connects to a MongoDB database using Mongoose. Define a Product model with properties such as name, price, and category. Implement routes or functions to perform CRUD operations: add a new product, retrieve all products, update a product by its ID, and delete a product by its ID. Use appropriate Mongoose methods for each operation and ensure that all data validations and error handling are included.

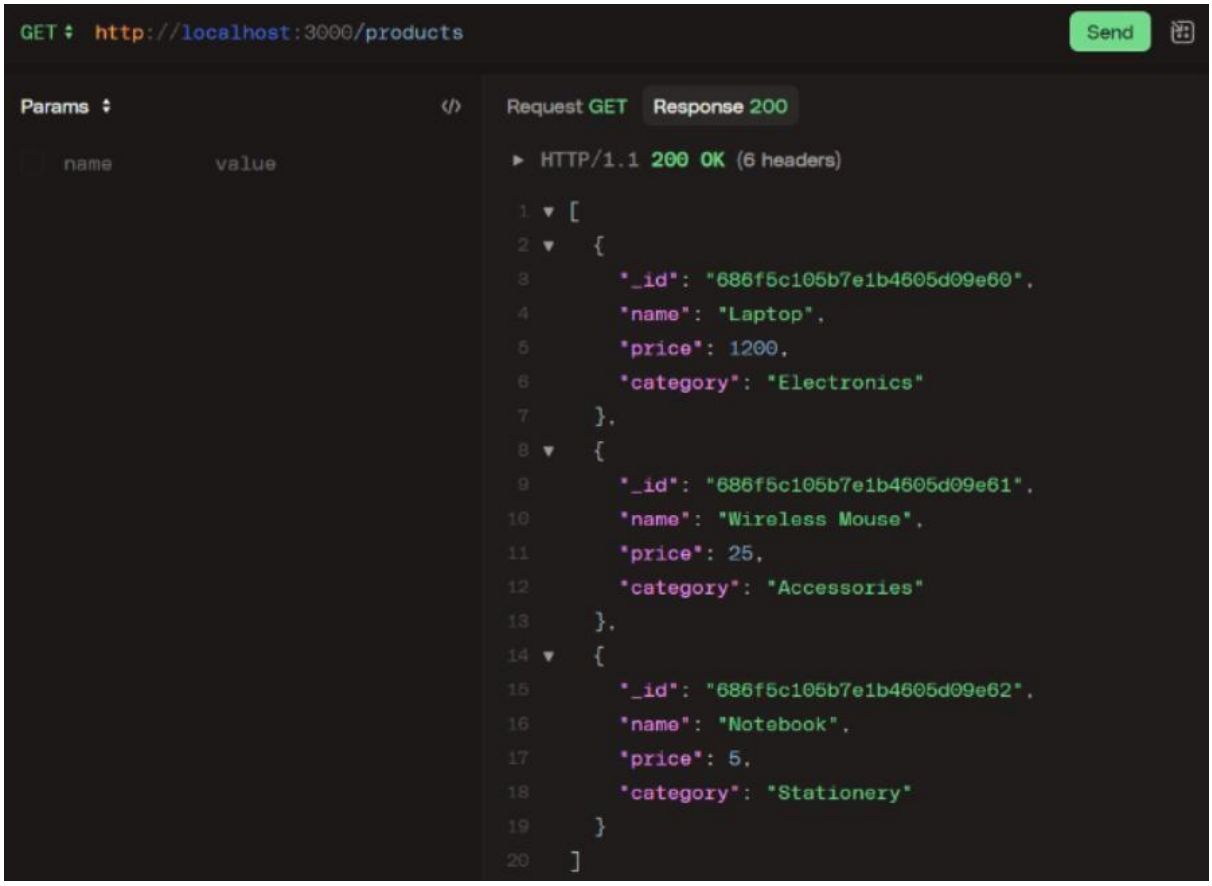
Code

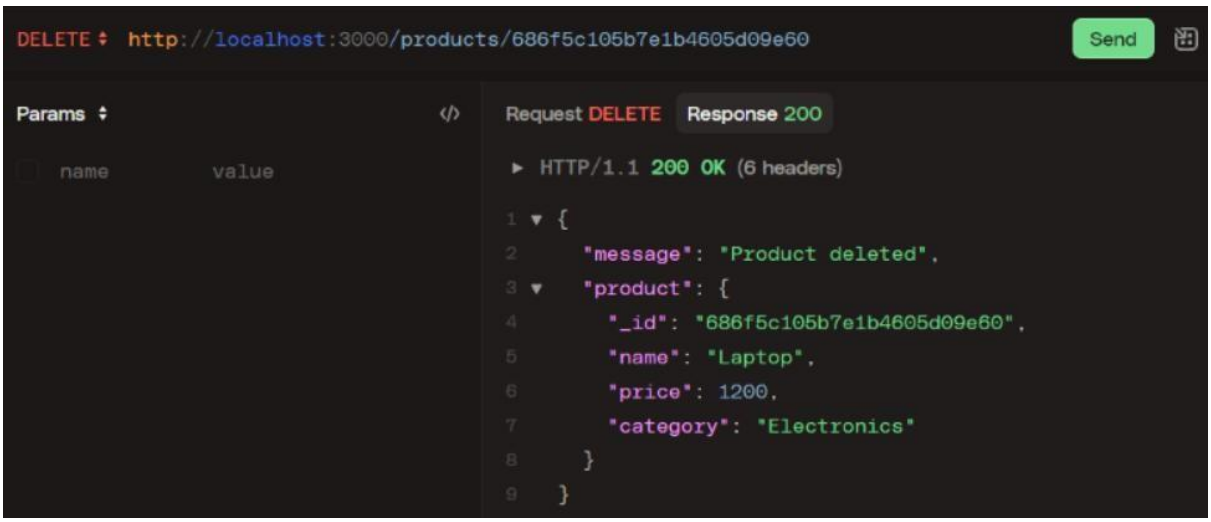
```
// Import dependencies const mongoose = require('mongoose'); // Connect to MongoDB mongoose.connect('mongodb://localhost:27017/productDB', {
```

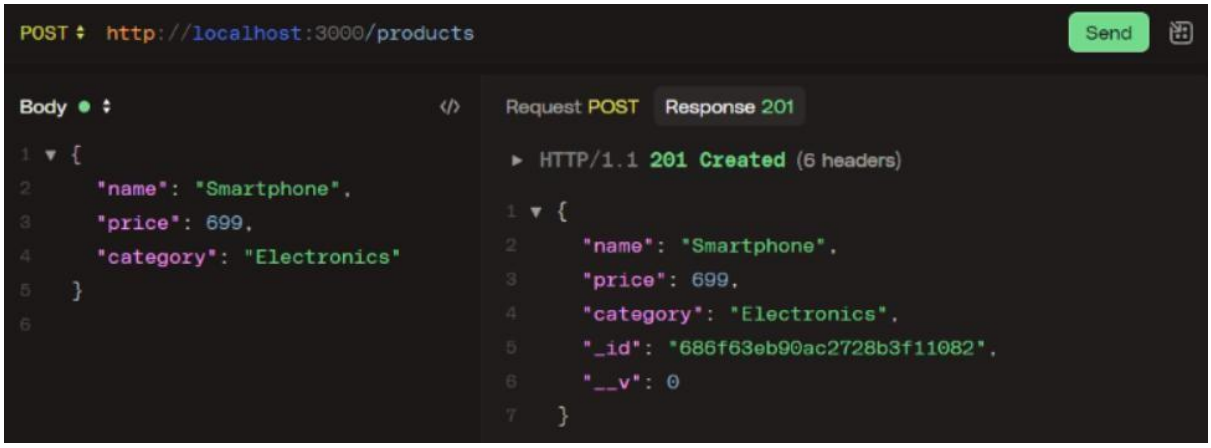
```
useNewUrlParser:                    true,
useUnifiedTopology: true
});

// Define Product schema const productSchema
= new mongoose.Schema({
  name: { type: String, required: true },
price: { type: Number, required: true },
category: { type: String, required: true }
});

// Create Product model const Product =
mongoose.model('Product', productSchema);

// CREATE: Add a new product async function
addProduct(name, price, category) {   const product =
new Product({ name, price, category });   await
product.save();   console.log('Product added:', product);
}

// READ: Retrieve all products async
function getAllProducts() {   const
products = await Product.find();
console.log('All Products:', products); }
```

```
// UPDATE: Update a product by ID async function updateProduct(id, updateData)
{   const product = await Product.findByIdAndUpdate(id, updateData, { new: true
});   console.log('Updated Product:', product);
}

// DELETE: Delete a product by ID async
function deleteProduct(id) {   await
Product.findByIdAndDelete(id);
console.log('Product deleted with ID:', id);
}

// Example usage (async () => {   await
addProduct('Laptop', 1200, 'Electronics');   await
getAllProducts();

  const products = await Product.find();   if
(products.length > 0) {     const productId =
products[0]._id;     await
updateProduct(productId, { price: 1300 });     await
deleteProduct(productId);
  }

  mongoose.connection.close();
})();
```

OUTPUT :


GET http://localhost:3000/products
Send
Params
name
value
Request GET
Response 200
HTTP/1.1 200 OK (6 headers)
[
{
"_id": "686f5c105b7e1b4605d09e60",
"name": "Laptop",
"price": 1200,
"category": "Electronics"
},
{
"_id": "686f5c105b7e1b4605d09e61",
"name": "Wireless Mouse",
"price": 25,
"category": "Accessories"
},
{
"_id": "686f5c105b7e1b4605d09e62",
"name": "Notebook",
"price": 5,
"category": "Stationery"
}
]


DELETE http://localhost:3000/products/686f5c105b7e1b4605d09e60
Send
Params
name
value
Request DELETE
Response 200
HTTP/1.1 200 OK (6 headers)
{
"message": "Product deleted",
"product": {
"_id": "686f5c105b7e1b4605d09e60",
"name": "Laptop",
"price": 1200,
"category": "Electronics"
}
}



EXPERIMENT 5.2

Title

Student Management System Using MongoDB and MVC Architecture

Objective

Learn how to design and build a Node.js application using the Model-View-Controller (MVC) architecture to manage student data stored in MongoDB. This task helps you understand how to separate concerns, structure backend logic clearly, and interact with a database using Mongoose.

Task Description

Create a student management system using Node.js, Express.js, and MongoDB (with Mongoose). Define a Student model with properties like name, age, and course. Implement a controller to handle CRUD operations (create, read, update, delete) on student data. Set up routes to connect client requests to the appropriate controller methods. Use Mongoose to handle all database interactions. Organize your codebase into separate folders for models, controllers, and routes to follow MVC principles clearly.

CODE:

// ====== app.js ====== const express =

require('express'); const mongoose =

require('mongoose'); const studentRoutes =

require('./routes/studentRoutes'); const app = express();

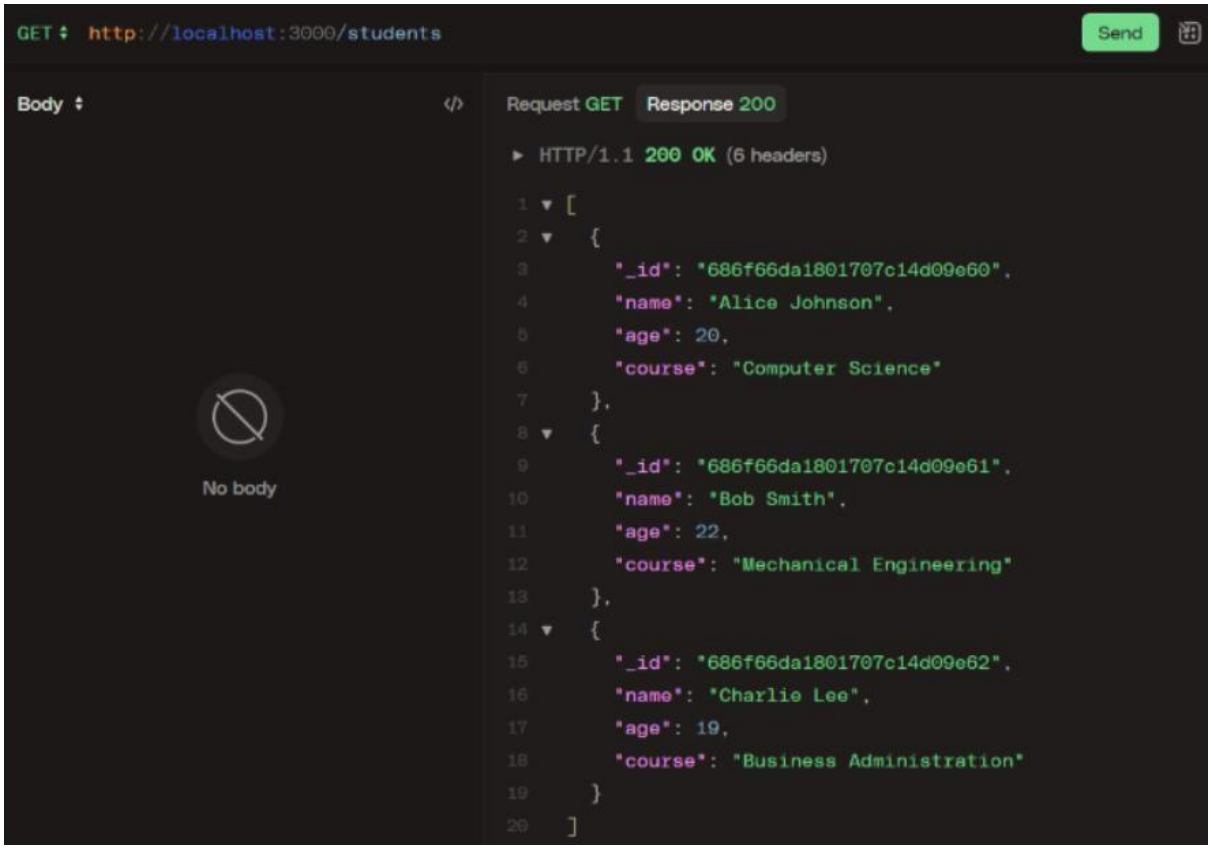
app.use(express.json());

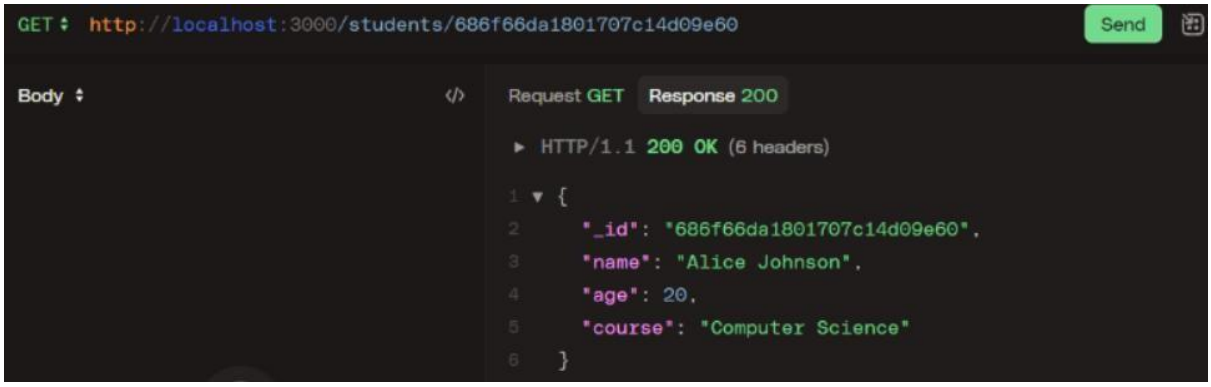
```
mongoose.connect('mongodb://localhost:27017/studentDB', {
useNewUrlParser: true,   useUnifiedTopology: true
});

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

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

// ====== models/Student.js ====== const
mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
age: { type: Number, required: true },
course: { type: String, required: true }
});

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



// ====== controllers/studentController.js ====== const
Student = require('../models/Student');

exports.createStudent = async (req, res) => {
```

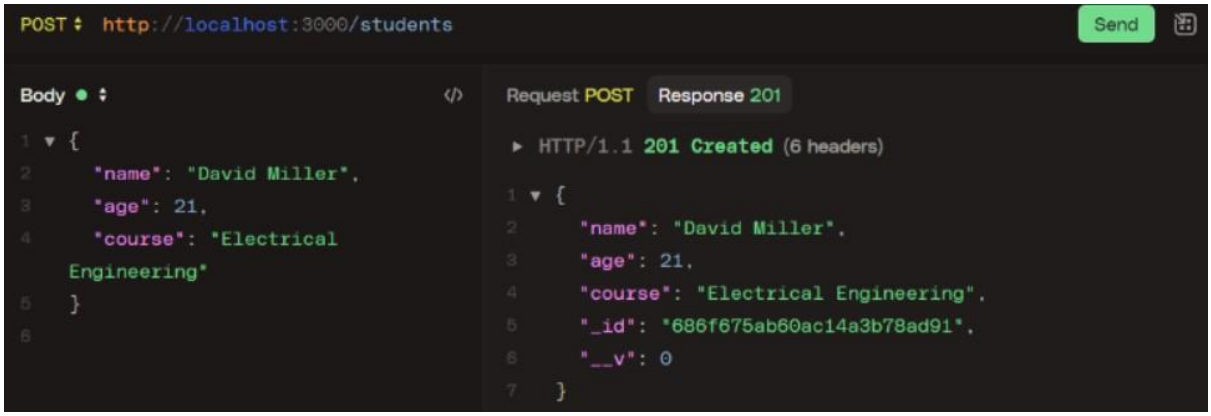
```
  try {
    const student = new Student(req.body);
await student.save();
res.status(201).json(student);
  } catch (err) {     res.status(400).json({ error:
err.message });
  }
};

exports.getAllStudents = async (req, res) => {
  try {
    const students = await Student.find();
res.json(students);   } catch (err) {
res.status(500).json({ error: err.message });
  }
};

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

exports.updateStudent = async (req, res) => {
```

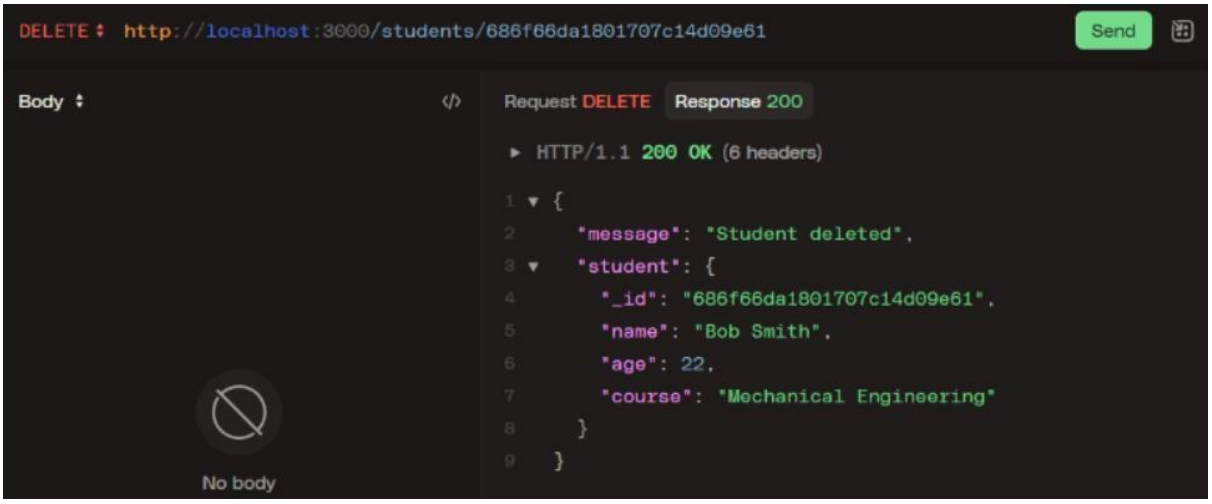
```
  try {
    const student = await Student.findByIdAndUpdate(req.params.id, req.body, { new: true });
if (!student) return res.status(404).json({ message: 'Student not found' });
res.json(student);   } catch (err) {     res.status(400).json({ error: err.message });
  }
};

exports.deleteStudent = async (req, res) => {
  try {
    const student = await Student.findByIdAndDelete(req.params.id);     if
(!student) return res.status(404).json({ message: 'Student not found' });
res.json({ message: 'Student deleted successfully' });
  } catch (err) {     res.status(500).json({ error:
err.message });
  }
};
```

OUTPUT :


GET http://localhost:3000/students
Send
Body
Request GET
Response 200
HTTP/1.1 200 OK (6 headers)
No body
[
{
"_id": "686f66da1801707c14d09e60",
"name": "Alice Johnson",
"age": 20,
"course": "Computer Science"
},
{
"_id": "686f66da1801707c14d09e61",
"name": "Bob Smith",
"age": 22,
"course": "Mechanical Engineering"
},
{
"_id": "686f66da1801707c14d09e62",
"name": "Charlie Lee",
"age": 19,
"course": "Business Administration"
}
]


GET http://localhost:3000/students/686f66da1801707c14d09e60
Send
Body
Request GET
Response 200
HTTP/1.1 200 OK (6 headers)
{
"_id": "686f66da1801707c14d09e60",
"name": "Alice Johnson",
"age": 20,
"course": "Computer Science"
}


POST http://localhost:3000/students
Send
Body
Request POST
Response 201
{
"name": "David Miller",
"age": 21,
"course": "Electrical Engineering"
}
HTTP/1.1 201 Created (6 headers)
{
"name": "David Miller",
"age": 21,
"course": "Electrical Engineering",
"_id": "686f675ab60ac14a3b78ad91",
"__v": 0
}



EXPERIMENT 5.3

Title

E-commerce Catalog with Nested Document Structure in MongoDB

Objective

Learn how to design and implement a MongoDB data model using nested documents to represent a real-world e-commerce catalog. This task strengthens your understanding of MongoDB's flexible document structure, schema design, and handling complex relationships inside a single collection.

Task Description

Create a MongoDB collection to represent an e-commerce catalog. Each product document should include fields such as name, price, category, and an array of nested variants. Each variant should include details like color, size, and stock. Insert a few sample product documents demonstrating different variants. Then, implement queries to retrieve all products, filter products by category, and project specific variant details. Use MongoDB shell queries or Mongoose methods to show how nested documents can be accessed and manipulated effectively.

CODE

```
// ==============================

// E-commerce Catalog with Nested Document Structure in MongoDB

// ==============================
```

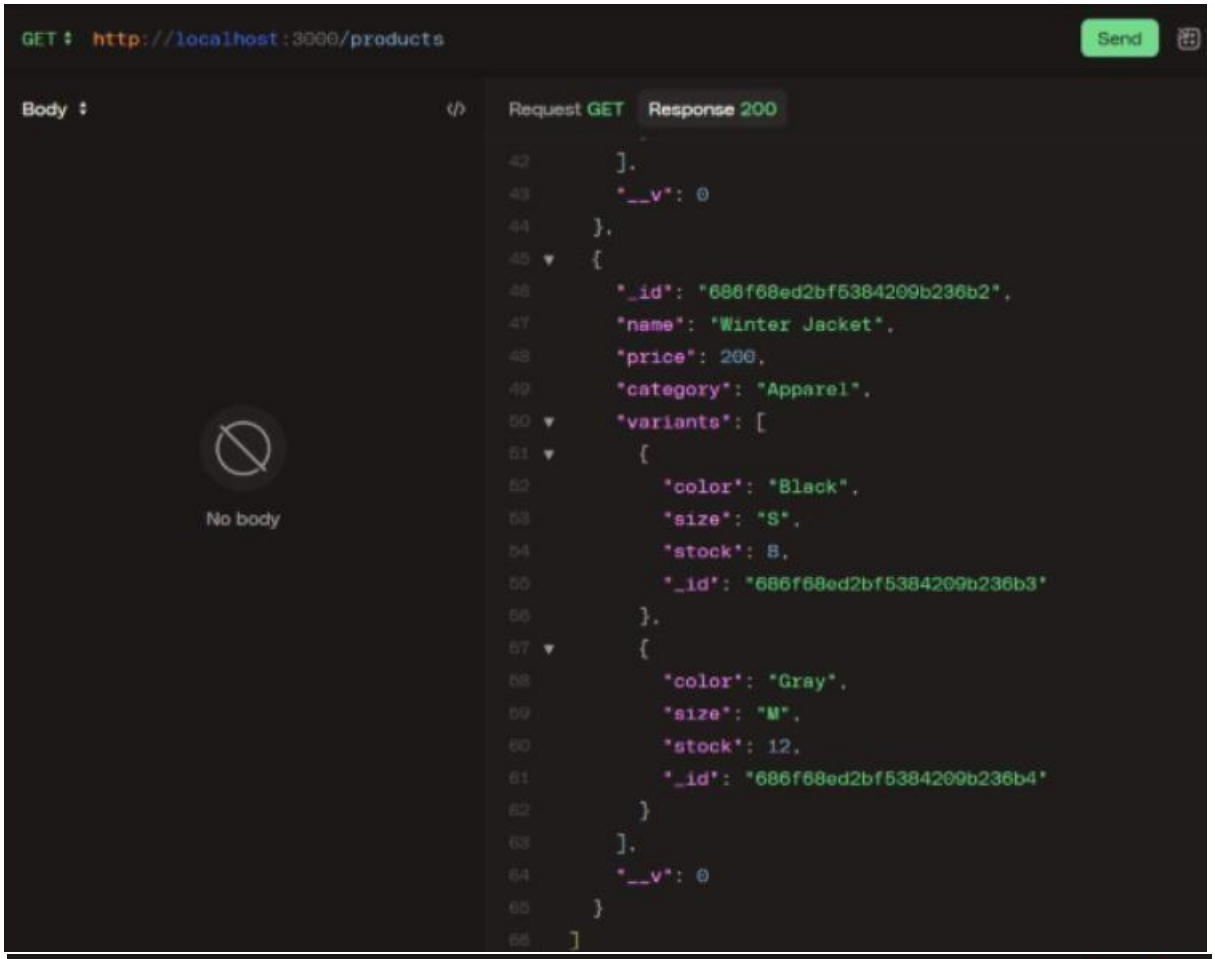
```
//   Import Dependencies const express
= require("express"); const mongoose =
require("mongoose");

const app = express(); app.use(express.json());

//   Connect to MongoDB mongoose
  .connect("mongodb://127.0.0.1:27017/ecommerce_catalog", {
useNewUrlParser: true,     useUnifiedTopology: true
  })
  .then(() => console.log("  Connected to MongoDB"))
  .catch((err) => console.error("   MongoDB Connection Error:", err));

//   Define Schemas and Models const
variantSchema = new mongoose.Schema({
  color: String,
size: String,
stock: Number
});

const productSchema = new mongoose.Schema({
name: String,   price: Number,
  category: String,
variants: [variantSchema]
});

const Product = mongoose.model("Product", productSchema);
```

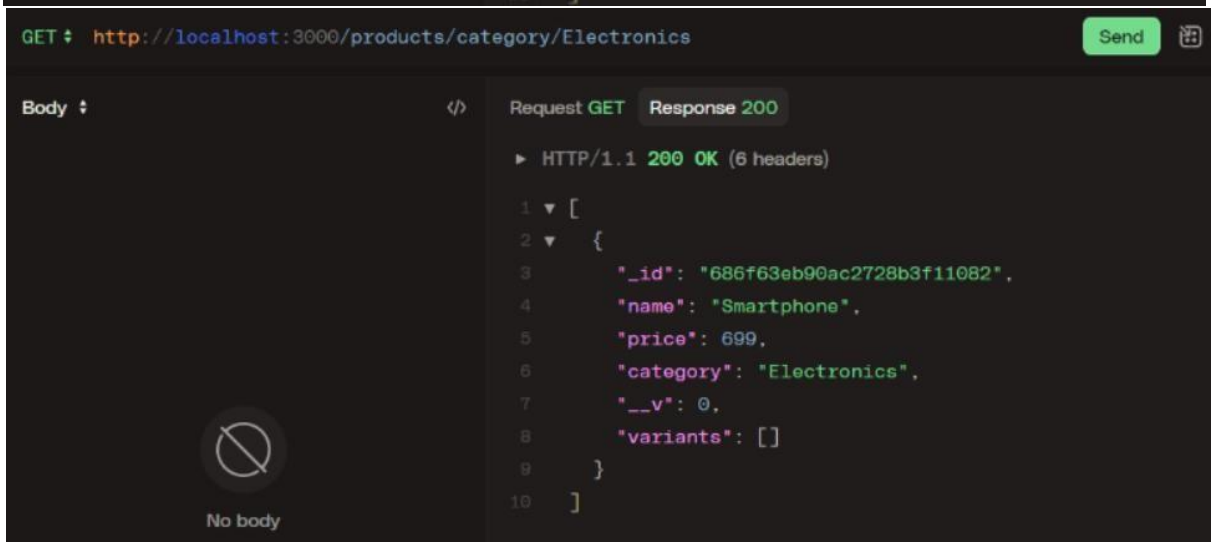
```
//   Insert Sample Products (Run Once) async function
insertSampleProducts() {   const count = await
Product.countDocuments();   if (count > 0) return console.log("
Sample data already exists.");

  await Product.insertMany([
    {
      name: "T-Shirt",
price: 499,       category:
"Clothing",       variants:
[
        { color: "Red", size: "M", stock: 25 },
        { color: "Blue", size: "L", stock: 10 },
        { color: "Black", size: "S", stock: 30 }
      ]
    },
    {
      name: "Sneakers",
price: 1999,       category:
"Footwear",       variants:
[
        { color: "White", size: "9", stock: 12 },
        { color: "Black", size: "8", stock: 5 }
      ]
    },
    {
```

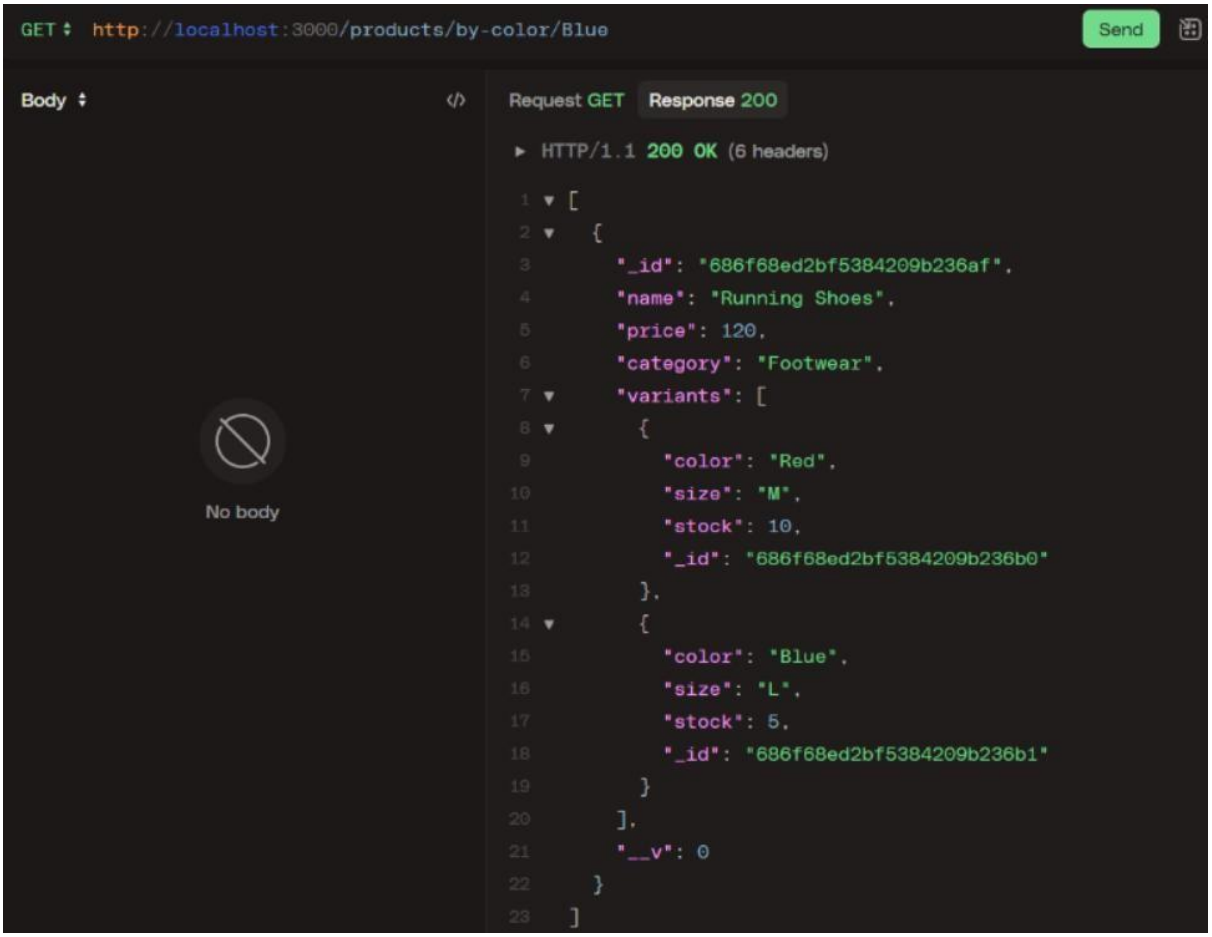
```
    name: "Wrist Watch",
price: 2499,      category:
"Accessories",      variants:
[
      { color: "Silver", size: "Standard", stock: 8 },
      { color: "Black", size: "Standard", stock: 15 }
    ]
  }
 ]);

 console.log("  Sample products inserted successfully!");
}

//   API Endpoints

// Get all products app.get("/products",
async (req, res) => {   const products =
await Product.find();
res.json(products);
});

// Filter products by category
app.get("/products/category/:category", async (req, res) => {
const category = req.params.category;

 const products = await Product.find({ category });
res.json(products);
});
```

```
// Get only product name and variants
app.get("/products/variants", async (req, res) => {   const
products = await Product.find({}, "name variants");
res.json(products);
});

// Find products with a specific variant color
app.get("/products/variant/color/:color", async (req, res) => {
const color = req.params.color;   const products = await
Product.find({ "variants.color": color });   res.json(products);
});

// Find specific variant details using $elemMatch
app.get("/products/:name/variant/:size", async (req, res) => {
const { name, size } = req.params;   const product = await
Product.find(
   { name },
   { variants: { $elemMatch: { size } } }
 );
 res.json(product);
});

//   Start Server
const PORT = 3000; app.listen(PORT, async () => {
console.log(`  Server running on http://localhost:${PORT}`);
await insertSampleProducts();
});
```

OUTPUT:


GET http://localhost:3000/products
Send
Body
Request GET
Response 200
No body
],
"__v": 0
},
{
"_id": "686f68ed2bf5384209b236b2",
"name": "Winter Jacket",
"price": 200,
"category": "Apparel",
"variants": [
{
"color": "Black",
"size": "S",
"stock": 8,
"_id": "686f68ed2bf5384209b236b3"
},
{
"color": "Gray",
"size": "M",
"stock": 12,
"_id": "686f68ed2bf5384209b236b4"
}
],
"__v": 0
}
]


GET http://localhost:3000/products/category/Electronics
Send
Body
Request GET
Response 200
HTTP/1.1 200 OK (6 headers)
[
{
"_id": "686f63eb90ac2728b3f11082",
"name": "Smartphone",
"price": 699,
"category": "Electronics",
"__v": 0,
"variants": []
}
]
No body


GET
http://localhost:3000/products/by-color/Blue
Send
Body
No body
Request GET
Response 200
HTTP/1.1 200 OK (6 headers)
[
{
"_id": "686f68ed2bf5384209b236af",
"name": "Running Shoes",
"price": 120,
"category": "Footwear",
"variants": [
{
"color": "Red",
"size": "M",
"stock": 10,
"_id": "686f68ed2bf5384209b236b0"
},
{
"color": "Blue",
"size": "L",
"stock": 5,
"_id": "686f68ed2bf5384209b236b1"
}
],
"__v": 0
}
]