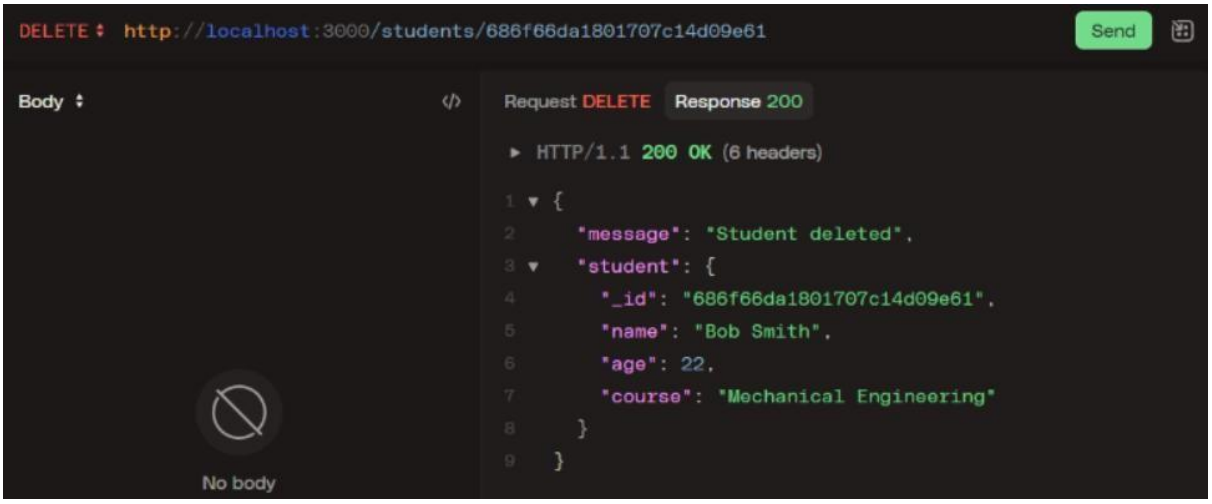


EXPERIMENT 5.3

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

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

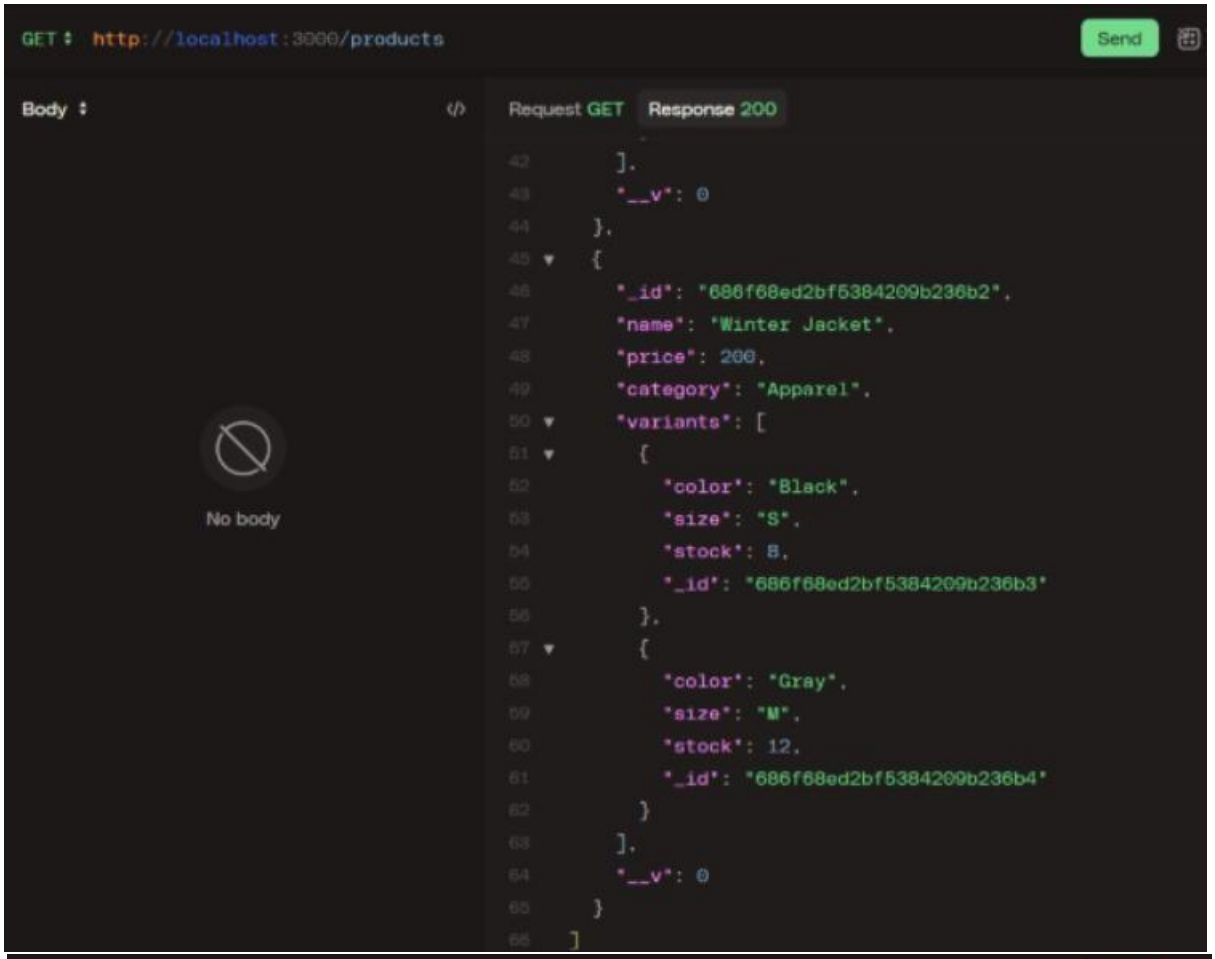
```
//   Import Dependencies const express
= require("express"); const mongoose =
require("mongoose");

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);
```

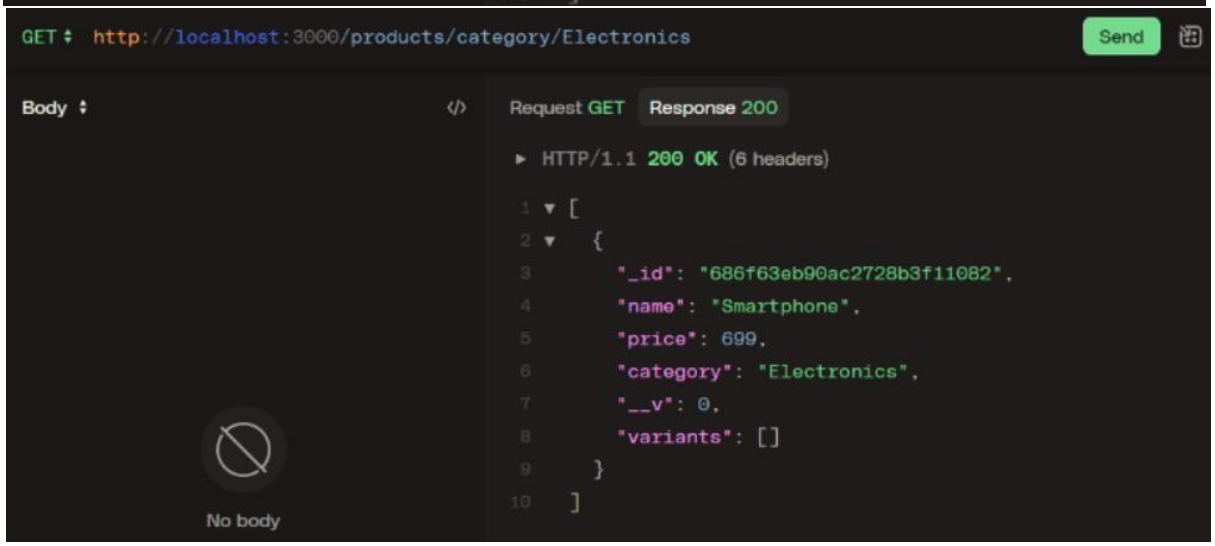
```
//   Insert Sample Products (Run Once) async function
insertSampleProducts() {   const count = await
Product.countDocuments();   if (count > 0) return console.log("
Sample data already exists.");

  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 }
      ]
    },
    {
```

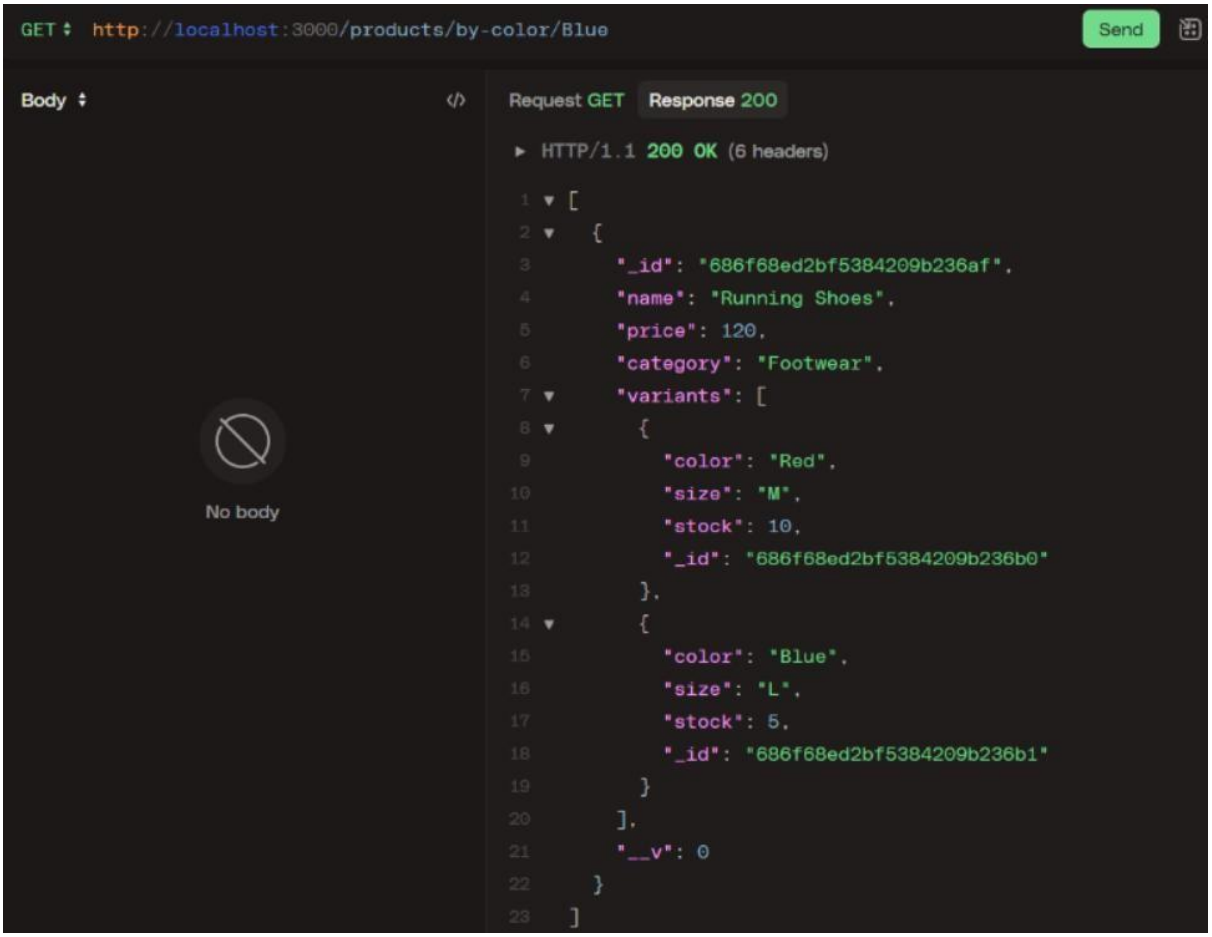
```
    name: "Wrist Watch",
price: 2499,      category:
"Accessories",      variants:
[
      { color: "Silver", size: "Standard", stock: 8 },
      { color: "Black", size: "Standard", stock: 15 }
    ]
  }
 ]);

 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
}
]