**Name :Nikhil**

**UID : 23BCC70030**

**Class : 23BCC-(A)**

**Experiment 4.3**

**Aim:**

To create an E-commerce Catalog using MongoDB that demonstrates the use of nested document structures to store products, categories, and variants in a single collection, enabling efficient retrieval and management of hierarchical data.

**Theory:**

* MongoDB:
  + A NoSQL document-based database that stores data as JSON-like documents (BSON).
  + Supports nested documents and arrays, which is ideal for hierarchical data like product catalogs.
* Nested Documents in E-commerce:
  + Products can have variants (size, color), category information, and pricing details embedded directly within the product document.
  + Benefits: Faster queries, less need for joins, better performance for hierarchical data.
* Example Structure:

{ "productName": "T-Shirt", "category": "Clothing", "variants": [ { "color": "Red", "size": "M", "price": 499 }, { "color": "Blue", "size": "L", "price": 549 } ], "inStock": true }

* Operations in Catalog System:
  + Add new product with nested variants
  + Retrieve all products or a specific product
  + Update product or nested variant
  + Delete product or variant

**Procedure:**

1. Install Node.js and MongoDB, then install npm packages: express, mongoose, body-parser.
2. Connect Node.js to MongoDB using Mongoose.
3. Define Product Schema with nested documents for variants.
4. Create controller functions for CRUD operations.
5. Define routes to interact with products via HTTP requests.
6. Test the system using Postman or browser.

**Code :**

// app.js

// ================== Import Packages ==================

const express = require('express');

const mongoose = require('mongoose');

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

// ================== Initialize App ==================

const app = express();

app.use(bodyParser.json());

// ================== MongoDB Connection ==================

mongoose.connect('mongodb://127.0.0.1:27017/ecommerceDB', {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => console.log("MongoDB Connected..."))

.catch(err => console.log(err));

// ================== Model (Nested Document Schema) ==================

const variantSchema = new mongoose.Schema({

color: String,

size: String,

price: Number,

stock: Number

});

const productSchema = new mongoose.Schema({

productName: { type: String, required: true },

category: String,

description: String,

variants: [variantSchema],

createdAt: { type: Date, default: Date.now }

});

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

// ================== Controller ==================

const productController = {

// Create new product

createProduct: async (req, res) => {

try {

const product = new Product(req.body);

const savedProduct = await product.save();

res.status(201).json(savedProduct);

} catch (err) {

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

}

},

// Get all products

getAllProducts: async (req, res) => {

try {

const products = await Product.find();

res.status(200).json(products);

} catch (err) {

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

}

},

// Get product by ID

getProductById: async (req, res) => {

try {

const product = await Product.findById(req.params.id);

if (!product) return res.status(404).json({ message: "Product not found" });

res.status(200).json(product);

} catch (err) {

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

}

},

// Update product

updateProduct: async (req, res) => {

try {

const updatedProduct = await Product.findByIdAndUpdate(

req.params.id,

req.body,

{ new: true }

);

if (!updatedProduct) return res.status(404).json({ message: "Product not found" });

res.status(200).json(updatedProduct);

} catch (err) {

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

}

},

// Delete product

deleteProduct: async (req, res) => {

try {

const deletedProduct = await Product.findByIdAndDelete(req.params.id);

if (!deletedProduct) return res.status(404).json({ message: "Product not found" });

res.status(200).json({ message: "Product deleted successfully" });

} catch (err) {

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

}

}

};

// ================== Routes ==================

app.post('/products', productController.createProduct);

app.get('/products', productController.getAllProducts);

app.get('/products/:id', productController.getProductById);

app.put('/products/:id', productController.updateProduct);

app.delete('/products/:id', productController.deleteProduct);

// ================== Start Server ==================

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running at http://localhost:${PORT}`);

});

**Learning Outcomes:**

1. Understand nested document structures in MongoDB for hierarchical data.
2. Learn to model products with variants and categories efficiently.
3. Gain experience performing CRUD operations with nested documents using Mongoose.
4. Understand the use of Node.js and Express.js for creating REST APIs.
5. Learn to structure controller and route logic in a single-file MVC-like pattern.
6. Understand advantages of NoSQL over relational databases for flexible data models.