Prince(23BCS11945)

Practice 1 - MongoDB

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 Implementation:

```
// Step 1: Initialize a Node.js project
// npm init -y
// npm install express mongoose body-parser

// Step 2: Import Dependencies
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

// Step 3: Initialize App and Middleware
const app = express();
app.use(bodyParser.json());

// Step 4: Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/productDB', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('MongoDB Connected'))
  .catch(err => console.log(err));

// Step 5: Define Product Schema
const productSchema = new mongoose.Schema({
    name: { type: String, required: true },
    price: { type: Number, required: true },
    category: { type: String, required: true }
});

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

// Step 6: CRUD Routes

// Create (POST)
app.post('/products', 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 });
    }
});

// Read (GET)
```

```
app.get('/products', async (req, res) => {
    try {
        const products = await Product.find();
        res.status(200).json(products);
    } catch (err) {
        res.status(500).json({ message: err.message });
    }
});

// Update (PUT)
app.put('/products/:id', async (req, res) => {
    try {
        const updatedProduct = await Product.findByIdAndUpdate(req.params.id, req.body, { new: true
        res.status(200).json(updatedProduct);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
});

// Delete (DELETE)
app.delete('/products/:id', async (req, res) => {
    try {
        const deletedProduct = await Product.findByIdAndDelete(req.params.id);
        res.status(200).json({ message: 'Product deleted', product: deletedProduct });
    } catch (err) {
        res.status(500).json({ message: err.message });
    }
});

// Step 7: Start Server
app.listen(3000, () => console.log('Server running on port 3000'));
```

Expected Output:

- GET /products → Returns all products
- POST /products → Adds a new product
- PUT /products/:id → Updates a product by ID
- DELETE /products/:id → Deletes a product by ID

Output: