EXPERIMENT NO.4.1

# Aim

To implement **CRUD (Create, Read, Update, Delete)** operations on a product database using **Mongoose** in Node.js, demonstrating how MongoDB can be used for managing product-related information.

# Theory

* **CRUD Operations** are the fundamental operations used in database management:
  1. **Create** → Add new product records.
  2. **Read** → Retrieve product data from the database.
  3. **Update** → Modify existing product records.
  4. **Delete** → Remove product records.
* **Mongoose** is an **ODM (Object Data Modeling)** library for MongoDB in Node.js. It provides:
  1. Schema-based modeling of data.
  2. Query building, validation, and hooks.
  3. Easy handling of MongoDB documents in a structured way.
* **Product Database Example**:

Each product may contain fields such as:

* 1. name (String)
  2. price (Number)
  3. category (String)
  4. inStock (Boolean)

# Code

## Setup Project

mkdir product-crud cd product-crud npm init -y

npm install express mongoose body-parser

## index.js – Main Server File

const express = require("express"); const mongoose = require("mongoose");

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

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

// Connect to MongoDB mongoose.connect("mongodb://localhost:27017/productDB", {

useNewUrlParser: true, useUnifiedTopology: true,

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

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

// Schema Definition

const productSchema = new mongoose.Schema({ name: String,

price: Number, category: String, inStock: Boolean

});

// Model

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

// CRUD APIs

// Create (POST)

app.post("/products", async (req, res) => { try {

const product = new Product(req.body); await product.save(); res.status(201).send(product);

} catch (err) { res.status(400).send(err);

}

});

// Read All (GET)

app.get("/products", async (req, res) => { try {

const products = await Product.find(); res.send(products);

} catch (err) { res.status(500).send(err);

}

});

// Read One by ID (GET) app.get("/products/:id", async (req, res) => {

try {

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

if (!product) return res.status(404).send("Product not found"); res.send(product);

} catch (err) {

res.status(500).send(err);

}

});

// Update (PUT)

app.put("/products/:id", async (req, res) => { try {

const product = await Product.findByIdAndUpdate( req.params.id,

req.body,

{ new: true, runValidators: true }

);

if (!product) return res.status(404).send("Product not found"); res.send(product);

} catch (err) { res.status(400).send(err);

}

});

// Delete (DELETE)

app.delete("/products/:id", async (req, res) => { try {

const product = await Product.findByIdAndDelete(req.params.id); if (!product) return res.status(404).send("Product not found"); res.send({ message: "Product deleted successfully" });

} catch (err) { res.status(500).send(err);

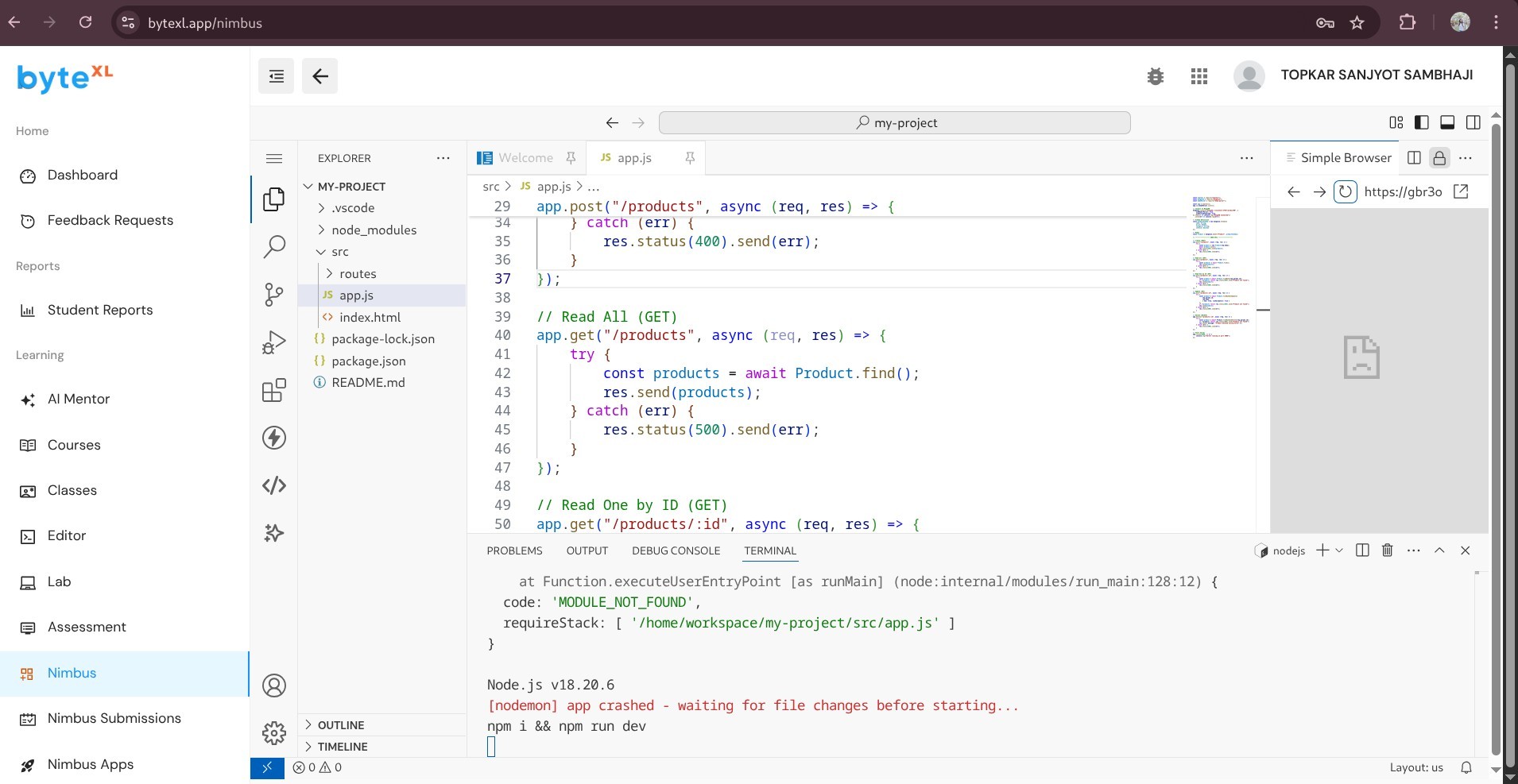
}

});

// Start Server app.listen(3000, () => {

console.log("Server running on port 3000");

});



# Learning Outcomes

After completing this experiment, students will be able to:

1. Understand the concept of **CRUD operations** in databases.
2. Implement a **MongoDB schema** using Mongoose.
3. Build a simple **RESTful API** in Express.js for product management.
4. Perform database operations like **insert, fetch, update, and delete** records.
5. Gain practical exposure to **backend development** with Node.js, Express, and MongoDB.