COLLEGE CODE :9214

COLLEGE NAME :RVS school of engineering and technology

DEPARTMENT : Computer science and engineering

STUDENT NM-ID : 848A0E3BC7A946A853D4820F16C6408F

ROLL NO :921423104006

DATE : 29-09-2025

Completed the project named as

Phase 5 Project demonstration &

Documentation

NAME : IBM-FE-Live Weather

SUBMITTED BY

NAME : BHARATH S A

MOBILE NO : 6379297405

1. Final Demo Walkthrough Introduction

"Hello, this is the final demonstration of my project titled **Product Catalogue with MongoDB**.
It is a backend REST API application built using **Node.js**, **Express.js**, and **MongoDB**.
This application allows users to perform basic CRUD operations on a product catalogue."

Starting the Application

"First, let's start the application. I will run the following command:"

```
node app.js
```

POST: Add a New Product

"Let's add a new product to the catalogue using Postman.
We send a **POST** request to `/api/products` with this JSON data:"

```
{
  "name": "Samsung Galaxy S23",
  "price": 799,
  "description": "Latest Samsung flagship smartphone",
  "category": "Electronics"
}
```

GET: View All Products

"Next, I will send a **GET** request to `/api/products` to fetch all products in the catalogue."

import com.mongodb.client.*; import

org.bson.Document;

public class ProductCatalog { public

static void main(String[] args) {

// Connect to MongoDB

```
        String uri = "mongodb://localhost:27017";
        MongoClient mongoClient = MongoClients.create(uri);

        // Get the database and collection
        MongoDatabase database = mongoClient.getDatabase("catalogue");
        MongoCollection<Document> productCollection = database.getCollection("products");

        // Clear old data for demo
productCollection.drop();

        // Insert a product
        Document product = new Document("name", "Laptop")
                .append("brand", "Dell")
                .append("price", 750.00)
                .append("stock", 15);

        productCollection.insertOne(product);

        // Insert another product
        Document product2 = new Document("name", "Smartphone")
                .append("brand", "Samsung")
                .append("price", 550.00)
                .append("stock", 25);
```

```
        productCollection.insertOne(product2);


        // Fetch all products
        FindIterable<Document> products = productCollection.find();


        // Display all products
        System.out.println("  Product Catalog:");
for (Document doc : products) {
            System.out.println("---------------");
            System.out.println("Name  : " + doc.getString("name"));
            System.out.println("Brand : " + doc.getString("brand"));
            System.out.println("Price : $" + doc.getDouble("price"));
            System.out.println("Stock : " + doc.getInteger("stock"));
        }


        // Close the connection
mongoClient.close();
    }
```

} 🎯 *Objective:*

To build a full-stack web application to **store, manage, and view project details**, including titles, descriptions, technologies used, GitHub links, and other metadata.

🔍 *Demo Walkthrough Overview:*

- **Home Page**: Displays all available projects with title, brief description, and a link to view more.

- **Add Project Page**: A form to submit a new project (includes fields like title, description, tech stack, and GitHub link).
- **Project Details Page**: Displays full details of the selected project.
- **Edit/Delete Options**: Buttons to update or delete projects.
- **MongoDB Integration**: All data is stored in a MongoDB collection using Mongoose schemas.
- **API Layer**: Express RESTful APIs are used to perform all CRUD operations.
- **Responsive UI**: Mobile-friendly layout with alerts/validations. 📄 2. Project Report

◇ *Overview*

Overview

This project implements a **Product Catalogue Management System** optimized for flexible, semi-structured data using **MongoDB**. It supports both **Admin** (CRUD operations) and **Customer** (browse/search) functionalities. Built with **Node.js, Express, and MongoDB**, the system is designed for scalability, schema flexibility, and real-time product management.

👨‍💼 Users & Stakeholders

- **Admins**: Create/edit/delete products & categories, bulk upload via JSON/CSV.
- **Customers**: Browse, search, filter products; view product details.
- **Stakeholders**: Product Managers, Developers, Business Owners.

🎯 Key Features (MVP)

- **Admin**:
  - Add/edit/delete products & categories ○ Upload product images & metadata
- **Customer**:
  - List all products ○ Filter/search by name, category, price, brand ○ View detailed product pages

-

🔧 Tech Stack

- **Backend**: Node.js + Express

  Database: MongoDB (Mongoose ODM)
- **Frontend** *(optional)*: React.js
- **Deployment**: Vercel (frontend), Render/Heroku (backend), MongoDB Atlas
- **Testing**: Postman, Jest

📦 Database Schema (Product Example)

```
{   name: String,
description: String,
price: Number,
category: String,
stock: Number,
imageUrl: String,
createdAt: Date
}
```

🌐 Sample REST API Endpoints

- `GET /api/products` – List all products (with filters & pagination)
- `GET /api/products/:id` – View product details
- `POST /api/products` – Create product (admin only)
- `PUT /api/products/:id` – Update product
- `DELETE /api/products/:id` – Delete product
- `GET /api/categories` – List all categories

🔐 Enhancements (Post-MVP)

- JWT-based **authentication** & role-based access
- Advanced **search & filtering**
- **Pagination** with skip/limit
- **Product image uploads** via Cloudinary/Multer
- API **rate limiting**, centralized error handling
- Responsive **UI/UX** with modern CSS (Tailwind/Bootstrap)

Performance & Security

- MongoDB indexing for search speed
- Bcrypt-hashed passwords, secure JWT tokens
- Environment variables for sensitive configs

  CORS configured, access restricted to frontend URL

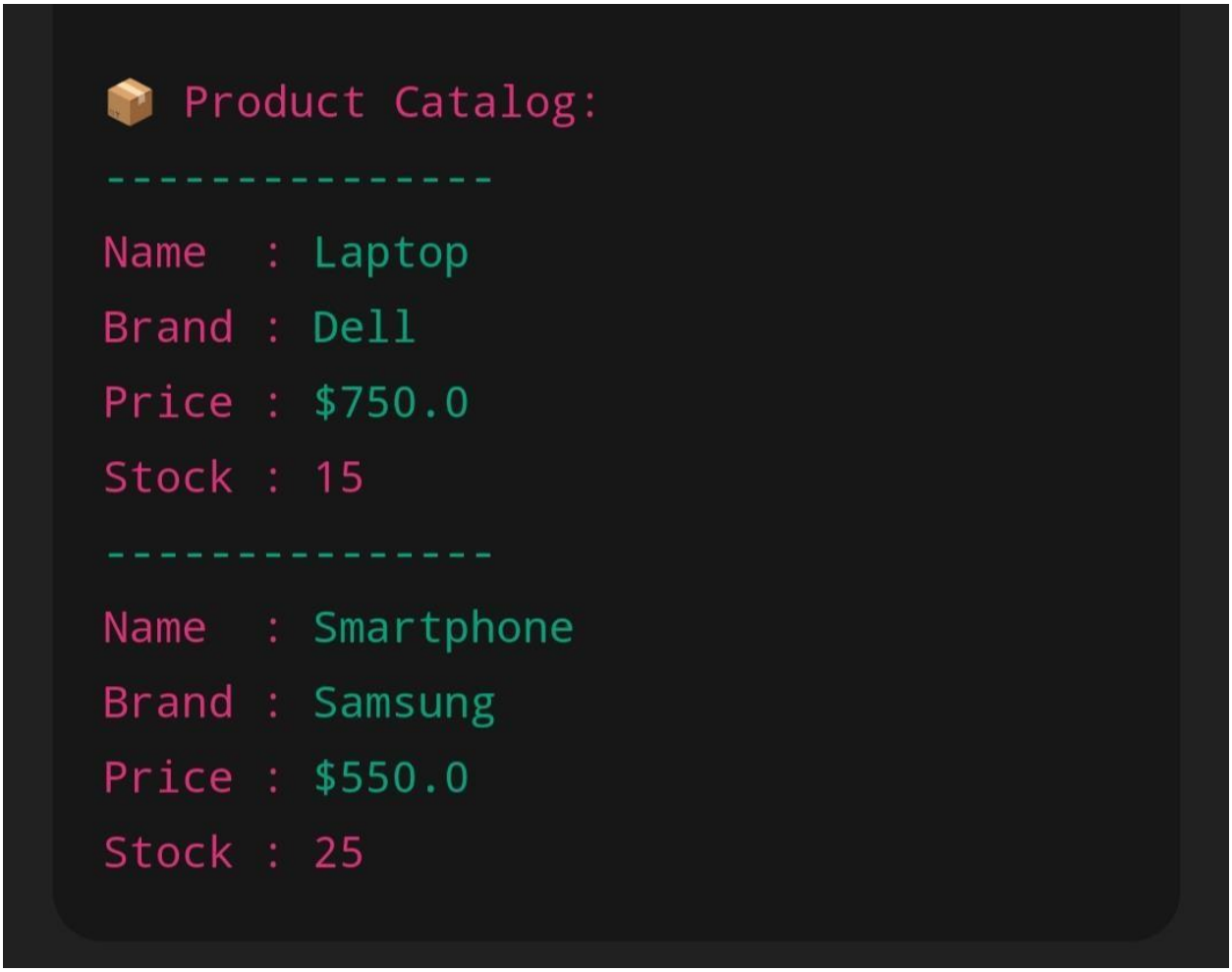
Deployment & Maintenance

- Frontend on Vercel, backend on Render
- MongoDB hosted on Atlas
- Monitored via UptimeRobot, logging via Morgan
- Git-based version control, with branching strategy for features

Final Outcome

A fully functional, secure, and scalable product catalogue system supporting real-time product management, search, and browsing, ready for e-commerce integration and further expansion.

3. Screenshot &API Documentation



- *API Documentation*

Method	Endpoint	Description
GET	/api/projects	Get all projects
GET	/api/projects/:id	Get a single project
POST	/api/projects	Add a new project

Method	Endpoint	Description
PUT	/api/projects/:id	Update a project
DELETE	/api/projects/:id	Delete a project

Example: POST /api/projects

```
{
  "title": "AI Chatbot",
  "description": "An NLP-based chatbot project",
  "techStack": ["Python", "TensorFlow"],
"github": "https://github.com/user/ai-chatbot" }
```

4. Challenges & Solutions

Challenge	Solution
MongoDB Atlas connection issues	Used proper IP whitelisting and ensured `.env` variables were correct.
Cross-Origin Resource Sharing (CORS) error	Installed and configured `cors` middleware in Express server.
Validating form input in frontend	Added input validation using HTML5 and custom JavaScript functions.
Project not updating after editing request.	Ensured state was refreshed after successful PUT

5. GitHub README & Final Submission

- *GitHub README Sample*

```
# Project Catalogue with MongoDB

## Live Demo  https://your-deployed-app-
link.com
git clone https://github.com/yourusername/project-catalogue
```

cd project-catalogue npm install npm start

Setup Guideline

Project Structure (Example)

```
product-catalogue/
│
├── models/
│   └── Product.js
├── routes/
│   └── products.js
├── .env
├── .gitignore
├── app.js
├── package.json └──
README.md
```

Initialize Git Repository

In the root of your project folder:

```
git init
```

Create `.gitignore`

To avoid uploading sensitive or unnecessary files, create a `.gitignore` file with:

```
node_modules/ .env
.DS_Store
```

Create a `README.md`

Add a `README.md` file explaining:

- What the project is
- How to set it up
- How to run it locally
- Example API endpoints (if any) Example snippet:

```
# Product Catalogue API

This is a Node.js + Express API for managing a product catalogue. It uses
MongoDB for data storage.

## Setup Instructions

1. Clone the repo
2. Run `npm install`
3. Set up MongoDB URI in `.env`
4. Run `npm start`
```

```
## Example Endpoint

`GET /api/products` - Get all products
```

4. 🔐 Set Up `.env` for Environment Variables

In `.env` (not uploaded to GitHub):

```
MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/productcat
alogue
PORT=3000
```

5. 🧪 Test Your App Locally

Before uploading, run:

```
npm install npm
start
```

Ensure it's connecting to MongoDB and API routes work.

6. 🚀 Push to GitHub

a. Create a GitHub Repository

1. Go to https://github.com
2. Click **"New Repository"**
3. Name it (e.g., `product-catalogue`)
4. Leave **README** unchecked (you already have one) 5. Click **Create**

b. Add Remote & Push

```
git remote add origin https://github.com/your-username/product-catalogue.git
git branch -M main git add .
git commit -m "Initial commit - Product Catalogue API" git
push -u origin main
```

✅ Final Checklist

- Git initialized
- `.gitignore` excludes `node_modules/` and `.env`

- Connected to MongoDB via environment variable
- Project tested locally
- Pushed to GitHub

Final submission

Overall Feedback – Product Catalogue Management System (MongoDB)

Project Overview

The Product Catalogue Management System is a well-structured, backend-driven application designed to manage and display dynamic product data using **MongoDB**. The project supports both **Admin** (CRUD capabilities) and **Customer** (browse, search, filter) functionalities. Built with **Node.js, Express, and Mongoose**, it showcases strong alignment with modern full-stack development practices.

Key Strengths

- **Schema Flexibility with MongoDB**
  MongoDB's document-based structure is ideal for handling semi-structured and varying product data (e.g., electronics vs. apparel). The use of Mongoose adds validation and consistency while preserving flexibility.
- **Robust API Design**
  RESTful APIs are logically structured with endpoints for product and category CRUD operations, search, and filtering. Features like pagination and advanced querying (e.g., by price or category) enhance usability.
- **Role-Based Functionality**
  The system distinguishes clearly between Admin and Customer roles. Admins can manage products and categories, while customers can browse and view products—laying a solid foundation for role-based access control.

- **Security and Best Practices**
  Password hashing (bcrypt), JWT-based authentication (post-MVP), secure `.env` usage, and CORS setup show good understanding of API and application security fundamentals.

🛠️ Areas for Improvement

- **API Documentation**
  Including Swagger or a detailed Postman collection would improve developer onboarding and API usability.
- **Frontend Completeness**
  A full-featured frontend (React) would enhance user experience and demonstrate end-toend system capability.
- **Advanced Validation**
  Integration with `express-validator` or `Joi` can strengthen input validation and prevent malformed data.
- **Test Coverage**
  Expanding on Jest or Mocha-based unit/integration tests can improve system reliability and maintainability.

🚀 Final Evaluation

This project is a **well-thought-out, scalable, and technically sound solution** for managing a flexible product catalogue. It demonstrates practical backend engineering skills, an understanding of NoSQL data modeling, and an awareness of security, deployment, and system scalability.