**INTRODUCTION :**

• PROJECT TITLE:

STORE MANAGER : KEEP TRACK OF INVENTORY

• Team ID:

NM2025TMID37138

• Team Leader:

RAKSHITHA S – [rakshithaseenu16@gmail.com](mailto:rakshithaseenu16@gmail.com)

• Team Members:

– SHARMILI N – [natarajansharmili@gmail.com](mailto:natarajansharmili@gmail.com)

– MAHESWARI K – [Maheswarikrishnamoorthy@gmail.com](mailto:Maheswarikrishnamoorthy@gmail.com)

**🛒 Project Overview: Store Manager – Inventory Tracking System**

**🎯 Objective:**

To develop a user-friendly inventory management system that enables store managers to efficiently monitor stock levels, manage product data, and streamline sales and procurement processes.

**🧩 Key Features:**

* **Inventory Dashboard**: Real-time view of all products, quantities, and stock status.
* **Stock Updates**: Automatic updates when products are sold or restocked.
* **Product Management**: Add, edit, or remove products with details like name, price, image, tags, and quantity.
* **Cart & Checkout System**: Add items to cart, process sales, and update inventory accordingly.
* **Low Stock Alerts**: Visual indicators and notifications for items running low.
* **Sales Records**: Log each transaction with timestamp, product details, and total value.
* **Search Functionality**: Quickly locate products using name, category, or tags.
* **Reporting Module**: Generate reports on sales trends, inventory levels, and reorder needs.

**🛠️ Technologies (Suggested Stack):**

* **Frontend**: HTML, CSS, JavaScript (React or Vue.js)
* **Backend**: Node.js, Django, or Flask
* **Database**: MySQL or MongoDB
* **Optional**: Integration with POS systems or accounting software

**👥 Target Users:**

* Store managers
* Inventory clerks
* Sales staff

**📈 Benefits:**

* Reduces manual errors in stock tracking
* Improves decision-making with data-driven insights
* Enhances customer satisfaction by preventing stockouts
* Saves time through automation of routine task

**🏗️ Architecture of Store Manager – Inventory Tracking System:**

**🔹 1. Client Layer (Frontend)**

* **Purpose**: Interface for users (store managers, clerks) to interact with the system
* **Technologies**: HTML, CSS, JavaScript (React, Vue.js, or Angular)
* **Components**:
  + Dashboard for inventory overview
  + Product management forms
  + Cart and checkout interface
  + Alerts and reporting views

**🔹 2. Application Layer (Backend)**

* **Purpose**: Handles business logic, processes requests, and enforces rules
* **Technologies**: Node.js, Django, Flask, or ASP.NET
* **Responsibilities**:
  + Authenticate users and manage roles
  + Process inventory updates and transactions
  + Trigger alerts for low stock or expiry
  + Generate reports and analytics

**🔹 3. Data Layer (Database)**

* **Purpose**: Stores persistent data securely and efficiently
* **Technologies**: MySQL, PostgreSQL, or MongoDB
* **Entities**:
  + **Products**: ID, name, category, price, quantity, tags
  + **Users**: ID, name, role, credentials
  + **Sales**: ID, timestamp, items sold, total
  + **Alerts**: Product ID, type, threshold
  + **Transactions**: Type (sale, restock), quantity, date

**🔹 4. Integration Layer (Optional)**

* **Purpose**: Connects with external systems for enhanced functionality
* **Examples**:
  + POS systems
  + Barcode scanners
  + Accounting software
  + Cloud storage or ERP tools

**🔄 Data Flow Example**

User → Frontend → Backend → Database → Backend → Frontend → User

* A store clerk logs in → adds items to cart → checks out → backend updates inventory → frontend refreshes dashboard

**☁️ Deployment Architecture**

* **Cloud-Based**: Hosted on AWS, Azure, or Firebase for scalability and remote access
* **On-Premise**: For local stores with internal servers and limited internet dependency

Absolutely! Here's a complete setup guide for your **Store Manager: Keep Track of Inventory** system using **Node.js** and **MongoDB**:

**⚙️ Setup Instructions for Store Manager – Inventory Tracking System**

**✅ Prerequisites**

Before you begin, make sure the following are installed:

* **Node.js**: [Download here](https://nodejs.org/)
* **MongoDB**:
  + For local development: [Install MongoDB locally](https://www.mongodb.com/try/download/community)
  + Or use [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) for cloud-based storage

**📁 Step-by-Step Setup**

**1. Initialize the Project**

mkdir store-manager

cd store-manager

npm init -y

**2. Install Required Packages**

npm install express mongoose dotenv body-parser cors

**3. Create Project Structure**

store-manager/

│

├── server.js

├── .env

├── models/

│ └── Product.js

├── routes/

│ └── productRoutes.js

├── controllers/

│ └── productController.js

**4. Configure Environment Variables**

Create a .env file:

PORT=3000

MONGO\_URI=mongodb://localhost:27017/store\_manager

Replace MONGO\_URI with your MongoDB Atlas URI if using cloud storage.

**5. Connect to MongoDB**

In server.js:

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

app.use(cors());

app.use(express.json());

mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

}).then(() => console.log('MongoDB connected'))

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

app.listen(process.env.PORT, () => {

console.log(`Server running on port ${process.env.PORT}`);

});

**6. Define a Product Model**

In models/Product.js:

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

name: String,

category: String,

price: Number,

quantity: Number,

tags: [String]

});

module.exports = mongoose.model('Product', productSchema);

**7. Create Routes and Controllers**

Set up basic CRUD operations for inventory in routes/productRoutes.js and controllers/productController.js.

**🚀 Run the Server**

node server.js

Visit http://localhost:3000 to start interacting with your inventory system.

Perfect! Here's a refined and complete folder structure for your **Store Manager: Keep Track of Inventory** project, using a **React frontend** and **Node.js + MongoDB backend**. This structure promotes modularity, scalability, and clean separation of concerns:

**📁 Folder Structure: SB-Works/**

SB-Works/

│

├── client/ # React Frontend

│ ├── public/ # Static assets and index.html

│ ├── src/

│ │ ├── components/ # Reusable UI components (Navbar, ProductCard, etc.)

│ │ ├── pages/ # Page-level components (Dashboard, Inventory, Reports)

│ │ ├── services/ # API calls to backend (axios configs)

│ │ ├── context/ # React Context for global state (cart, user, inventory)

│ │ ├── App.js # Main app component

│ │ └── index.js # Entry point

│ └── package.json # Frontend dependencies

│

├── server/ # Node.js Backend

│ ├── controllers/ # Business logic (productController.js, userController.js)

│ ├── models/ # Mongoose schemas (Product.js, User.js, Sale.js)

│ ├── routes/ # API endpoints (productRoutes.js, userRoutes.js)

│ ├── middleware/ # Auth, error handling, logging

│ ├── config/ # DB connection, environment setup

│ ├── utils/ # Helper functions (e.g., alert triggers, report generators)

│ ├── server.js # Entry point for backend

│ └── .env # Environment variables

│

├── README.md # Project documentation

└── package.json # Root-level dependencies (if using concurrently)

**🧠 Tips for Development**

* Use concurrently to run both frontend and backend together during development.
* Keep your API routes RESTful and modular.
* Store sensitive keys in .env and never commit them to Git.
* Use React Context or Redux for managing global state like inventory and cart.

**🚀 Running the Application**

**🖥️ Frontend (React)**

cd SB-Works/client

npm install # Install dependencies

npm start # Launch frontend on http://localhost:3000

This will start the React development server and open the app in your browser.

**🖧 Backend (Node.js + MongoDB)**

cd SB-Works/server

npm install # Install backend dependencies

npm start # Start Express server (default port: 5000 or as defined in .env)

Make sure MongoDB is running locally or your .env file points to a live MongoDB Atlas URI.

**🌐 Access the App**

* **Frontend**: <http://localhost:3000>
* **Backend API**: <http://localhost:5000/api> (or your configured port)

**🧪 Quick Checklist**

* ✅ MongoDB is running
* ✅ .env file is configured correctly
* ✅ Both frontend and backend servers are running
* ✅ CORS is enabled in backend to allow frontend requests

Here’s a clean and developer-friendly **API Documentation** outline for your **Store Manager: Keep Track of Inventory** system. This will help you define and expose endpoints for managing products, inventory, and sales.

**📘 API Documentation – Store Manager Inventory System**

**🌐 Base URL**

http://localhost:5000/api

**🔐 Authentication**

(Optional if you want role-based access)

* POST /auth/login – Login with credentials
* POST /auth/register – Register a new user

**📦 Product Management**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /products | Retrieve all products |
| GET | /products/:id | Get details of a specific product |
| POST | /products | Add a new product |
| PUT | /products/:id | Update product details |
| DELETE | /products/:id | Delete a product |

**Sample Product Object:**

{

"name": "Wireless Mouse",

"category": "Electronics",

"price": 499,

"quantity": 25,

"tags": ["accessory", "tech"]

}

**🛒 Inventory Operations**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /inventory/restock/:id | Add stock to a product |
| POST | /inventory/deplete/:id | Reduce stock (e.g., after sale) |
| GET | /inventory/low-stock | List products below threshold |

**💰 Sales Management**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /sales | Retrieve all sales records |
| POST | /sales | Create a new sale |

**Sample Sale Object:**

{

"items": [

{ "productId": "abc123", "quantity": 2 },

{ "productId": "def456", "quantity": 1 }

],

"total": 1497,

"timestamp": "2025-09-13T21:48:00Z"

}

**📊 Reporting**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /reports/daily | Daily sales and inventory summary |
| GET | /reports/monthly | Monthly performance report |

**⚠️ Error Handling**

All endpoints return appropriate HTTP status codes:

* 200 OK – Success
* 400 Bad Request – Invalid input
* 404 Not Found – Resource not found
* 500 Internal Server Error – Server-side issue