Store Manager: Keep Track of Inventory using React.js and Node.js

1. Introduction

Effective inventory management is a critical component of any retail or wholesale business. It ensures that the store has the right quantity of products in stock to meet customer demand while minimizing overstock or stockouts. Traditional inventory tracking methods are often manual and prone to errors, causing inefficiencies and financial losses.

This project provides a web-based inventory management system built with **React.js** on the frontend and **Node.js** on the backend. The system enables store managers to efficiently monitor stock levels, add new items, update quantities, and remove obsolete products through a user-friendly interface. Real-time synchronization between the frontend and backend allows for smooth, efficient management of inventory data.

---

2. Objectives

- Develop a responsive frontend using React.js to display inventory data and handle user interactions smoothly.
- Create a RESTful API using Node.js and Express.js to perform CRUD (Create, Read, Update, Delete) operations on inventory items.
- Store inventory data securely in a database (MongoDB recommended for ease of use with Node.js).
- Allow store managers to:
    - View current inventory with details such as product name, quantity, price, and description.
    - Add new inventory items with necessary details.
    - Update stock levels when new shipments arrive or items are sold.
    - Remove items that are discontinued or no longer sold.
- Ensure seamless communication between frontend and backend using HTTP requests (Axios or Fetch API).
- Provide clear error handling and feedback to users during interactions.

---

3. Prerequisites

Before starting this project, ensure you have the following installed and setup on your development machine:

- **Node.js** (version 14 or above recommended): Runtime environment for backend server.
- **npm** or **yarn**: Package managers to install dependencies.
- **MongoDB** (local or cloud, e.g., MongoDB Atlas): Database to store inventory data.
- Basic knowledge of **JavaScript**, **React.js**, and **Node.js/Express.js**.
- Code editor such as **Visual Studio Code**.
- Postman or similar API testing tool (optional but useful).

---

4. Project Structure

The project is divided into two main parts: **frontend** (React.js) and **backend** (Node.js with Express). Below is a typical folder structure:

```
inventory-management/
│
├── backend/                  # Node.js backend server
│   ├── controllers/          # Logic to handle API requests
│   ├── models/               # Database schemas (e.g., InventoryItem)
│   ├── routes/               # API endpoints
│   ├── app.js                # Express app setup
│   ├── server.js             # Server startup script
│   └── config/               # Database configuration
│
├── frontend/                 # React.js frontend app
│   ├── public/               # Static files (index.html, favicon, etc.)
│   ├── src/
│   │   ├── components/       # React components (InventoryList,
AddItemForm, etc.)
│   │   ├── services/         # API call functions
│   │   ├── App.js            # Main app component
│   │   ├── index.js          # React DOM render
│   └── package.json          # Frontend dependencies
│
├── README.md                 # Project documentation
└── .gitignore                # Git ignore rules
```

---

5. Project Flow

Step 1: User Interface (Frontend)

- The store manager opens the web app built with React.js.
- The homepage displays a list of all inventory items fetched from the backend API. Each item shows product name, quantity, price, and description.
- The user can interact with the UI to add a new product, update the quantity of an existing product, or delete an item.

Step 2: API Requests

- When the manager performs an action (e.g., adds an item), the React app sends an HTTP request (POST, PUT, DELETE, GET) to the Node.js backend API using Axios or Fetch.
- The API endpoints are designed to handle these requests securely and validate input data.

Step 3: Backend Processing

- The Node.js server receives the API request and forwards it to the appropriate controller function.
- The controller interacts with the MongoDB database via models (using Mongoose ORM) to read or modify inventory data.
- After the operation is successful, the backend sends a JSON response back to the frontend confirming the action or returning updated data.

Step 4: Frontend Updates

- Upon receiving the API response, the React app updates the UI dynamically to reflect the latest inventory state.
- For example, after adding a new item, the new product appears in the inventory list without needing a page refresh.
- Error messages or confirmations are shown if needed.

Step 5: Continuous Interaction

- The manager can repeat these operations anytime to keep inventory data current and accurate.

6.Project Setup and Configuration

A. Backend Setup (Node.js + Express + MongoDB)

1. **Create a backend folder**:
```
mkdir backend && cd backend
npm init -y
```
4. **Install required dependencies**:
```
npm install express mongoose cors dotenv
npm install nodemon --save-dev
```
7. **Setup file structure**:
    - `server.js`: Entry point
    - `app.js`: Main Express app config
    - `routes/inventoryRoutes.js`: API routing
    - `models/InventoryItem.js`: MongoDB schema
    - `controllers/inventoryController.js`: Logic for inventory actions
    - `.env`: Environment variables (MongoDB URI, port)
8. **Example MongoDB model** (`models/InventoryItem.js`):
```
const mongoose = require('mongoose');

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

```
    quantity: Number,
    price: Number,
    description: String
  });

  module.exports = mongoose.model('InventoryItem',
  InventoryItemSchema);
```

B. Frontend Setup (React.js)

1. **Create the React app**:

```
npx create-react-app frontend
cd frontend
```

4. **Install Axios for API requests**:

```
npm install axios
```

6. **File structure**:
   - `components/Inventory.js`: Displays list
   - `components/Addproduct.js`: Add item
   - `components/EditItemForm.js`: Edit existing item
   - `services/api.js`: Axios config and API functions
7. **Example API service** (`services/api.js`):

```
import axios from 'axios';

const API_BASE = 'http://localhost:5000/api/inventory';

export const fetchInventory = () => axios.get(API_BASE);
export const addItem = (item) => axios.post(API_BASE, item);
export const updateItem = (id, item) =>
axios.put(`${API_BASE}/${id}`, item);
export const deleteItem = (id) =>
axios.delete(`${API_BASE}/${id}`);
```

□ 7. Project Development

Features Developed

1. **View Inventory**:
   - Fetches and displays all inventory items from backend using `GET /api/inventory`.
2. **Add Inventory Item**:
   - Frontend form submits new item to backend with `POST /api/inventory`.
3. **Update Item**:
   - UI allows editing quantity or details using `PUT /api/inventory/:id`.
4. **Delete Item**:
   - Items can be removed using `DELETE /api/inventory/:id`.

Frontend Technologies

- **React.js** for UI
- **Axios** for HTTP requests

- **React Hooks** for state and effects

Backend Technologies

- **Node.js + Express** for server
- **Mongoose** for MongoDB integration
- **CORS + dotenv** for environment and cross-origin support

□ 8. Project Implementation and Execution

A. Running the Backend

1. Create `.env` file in `backend/` with:
2. `MONGO_URI=your_mongo_connection_string`
3. `PORT=5000`
4. Run the backend server:
5. `npm run dev`
6. Confirm server running at:
   `http://localhost:5000/api/inventory`

B. Running the Frontend

1. Start the React frontend:
2. `npm start`
3. The app opens at:
   `http://localhost:3000`