**Title: Store Manager: Keep Track of Inventory**

**Category:** Frontend Development with Database Integration (MongoDB)

**1. Abstract**

The *Store Manager* application is an inventory management system designed to simplify stock handling for small and medium-sized stores. Built using **ReactJS** for the frontend and **MongoDB** for backend database management, it ensures smooth handling of products, stock levels, sales, and cart operations. The system is user-friendly, efficient, and scalable, providing store managers with real-time updates, low-stock alerts, and detailed sales records.

**2. Objectives**

* To develop a web-based inventory management system using **ReactJS**.
* To maintain accurate stock levels and prevent over-selling.
* To automate cart and checkout operations with inventory updates.
* To generate and store detailed sales records.
* To provide a responsive, interactive UI with **TailwindCSS** styling.
* To enable data persistence through **localStorage** and MongoDB integration.

**3. Key Features**

1. **Inventory Management:**  
   Add, update, and manage stock levels efficiently.
2. **Stock Updates:**  
   Automatic stock deduction upon checkout.
3. **Cart Management:**  
   Add/remove products, update quantities, and calculate total.
4. **Checkout System:**  
   Clears cart, updates inventory, and records sales.
5. **Add Products:**  
   Input product name, image URL, price, stock, and tags.
6. **Stock Alerts:**  
   Highlight low stock products in red with adjustable threshold.
7. **Search Functionality:**  
   Quickly locate products in the inventory.
8. **Sales Records:**  
   Store and display sales history with date, products, and value.

**4. Prerequisites**

* **Node.js & npm** – for running and managing React apps.
* **React.js** – for UI development.
* **MongoDB** – for database storage and management.
* **TailwindCSS** – for responsive UI styling.
* **Git & GitHub** – version control and collaboration.
* **Code Editor (VS Code preferred).**

**5. Methodology**

**Step 1: Project Setup**

* Initialize React app using create-react-app or Vite.
* Install dependencies: react, react-dom, tailwindcss, axios, react-router-dom.
* Configure TailwindCSS and folder structure (components/, context/, pages/).

**Step 2: State Management**

* Implement **Context API + Reducers** for Inventory, Cart, and Sales.
* Persist states with **localStorage**.

**Step 3: Component Development**

* **Inventory Component:** Displays products, stock, and search functionality.
* **Product Component:** Shows product details with add/update stock features.
* **Cart Component:** Handles cart operations, checkout, and stock deduction.
* **Sales Component:** Displays sales history with sorting.

**Step 4: Utilities**

* Create helper functions (currency formatter, sorting, filtering).
* Validate inputs to prevent negative stock/invalid cart actions.

**Step 5: Styling**

* Responsive layouts using TailwindCSS grid & flex utilities.
* Consistent theme and UI patterns across components.

**Step 6: Testing & Debugging**

* Component-level testing for rendering and functionality.
* State and reducer testing for inventory/cart updates.
* Debugging for responsive issues and edge cases.

**6. Tools & Technologies Used**

* **Frontend:** ReactJS, TailwindCSS
* **Backend/Database:** MongoDB
* **Version Control:** Git, GitHub
* **IDE:** Visual Studio Code
* **Package Manager:** npm
* **APIs & Libraries:** Axios, React Router

**7. Implementation Flow**

1. **User adds/updates products in inventory.**
2. **Stock displayed with search & alerts.**
3. **User selects items and adds to cart.**
4. **Checkout updates inventory & records sale.**
5. **Sale records stored with timestamp and details.**

**8. Testing**

* **Unit Testing:** Verified each component (Inventory, Cart, Sales).
* **Integration Testing:** Checked interaction between cart and inventory.
* **Database Testing:** Ensured correct storage/retrieval from MongoDB.
* **UI Testing:** Verified responsiveness across devices.

**9. Outcomes**

* Successfully developed a **responsive, user-friendly inventory system**.
* Automated inventory updates reduced manual errors.
* Generated reliable **sales reports for decision-making**.
* Achieved data persistence and smooth state management.

**10. Conclusion & Future Scope**

The *Store Manager* project effectively addresses common inventory challenges for small businesses. By combining **ReactJS** with **MongoDB**, it provides scalability and efficiency.

**Future Enhancements:**

* Role-based authentication (Admin, Staff).
* Barcode scanning for faster product entry.
* Data visualization (charts for sales trends).
* Cloud deployment for real-time multi-user access.

**Screenshots**



