Frontend Development With React.js

Project Documentation

STORE MANAGER: KEEP TRACK OF INVENTORY

1. **Introduction**

**Team Id** : NM2025TMID36656

**Team Leader** : Madhan kumar.D

**Mail Id** :mad0214han@gmail.com

**Team Members**  :

|  |  |
| --- | --- |
| **Name** | **Mail Id** |
| Kathir kumaran.S | Kathirkumaran2006@gmail.com |
| Mahesh.L | Mahesh.l20155@gmail.com |
| Krishnaraj.M | krishnabhavanrj@gmail.com |

1. **Project Overview**

**.)** Purpose: SB Works connects clients and freelancers through project postings, bidding, and real-time communication.

**.)** Features: Project posting and bidding – Secure chat system – Feedback and review system – Admin control panel

**.)** Scope of the Project

This project focuses on the core responsibilities of a store manager related to inventory. The scope includes:

* The entire lifecycle of an item within the store, from procurement and receiving to sale and reporting.
* An overview of the tools and technologies commonly used for inventory management.
* The reporting and analytical functions that help a manager make data-driven decisions.

The project does **not** delve into the complexities of the `supply chain beyond the store's receiving dock or the detailed technical specifications of specific hardware (e.g., barcode scanners) unless they are directly relevant to the manager's role.

3. **Architecture**

Frontend: React.js with Bootstrap and Material UI

Backend: Node.js and Express.js managing server logic and API endpoints

Database: MongoDB stores user data, project information, applications, and chat messages

The Business Logic Tier (ApplicationServer)

This is the "middle" layer where all the core business rules and data processing occur. It acts as a bridge between the user interface and the database.

* Components:
  + Inventory Management Module**:**

Handles all core inventory functions, such as adding new items, updating stock levels, and managing returns.

* + Sales Processing Module**:**

Receives data from the POS system and processes the transactions, updating inventory accordingly.

* + Reporting and Analytics Module:

Processes raw inventory and sales data to generate meaningful reports for the store manager (e.g., low-stock reports, best-seller lists, shrinkage analysis).

* + Alerts and Notifications Engine:

Sends automated alerts for low stock, expiring products, or other critical events.

* + Function**:**

This layer is the "brain" of the system. It ensures data integrity by applying all the rules (e.g., a negative stock value is not allowed) before interacting with the database.

.

**4. Setup Instructions**

• Prerequisites: Node.js

MongoDB

Git

React.js

Express.js

Mongoose

Visual Studio Code

• Installation Steps:

# Clone the repository git clone

# Install client dependencies cd client npm install

# Install server dependencies cd ../server npm install

Initial Inventory Data Entry:

Before you can start tracking, you need to input your existing inventory.

Import Products

* + If you have your product data in a spreadsheet (e.g., Excel or CSV), use the import feature.
  + Go to Products > Import and upload your file. The system will guide you through mapping your spreadsheet columns to the system's fields (e.g., SKU, Name, Price, Quantity).

Manual Product Entry (Optional)

* + For new products or if you have a small number of items, you can add them manually.
  + Go to Products > Add New Product and fill in the details for each item.

Physical Stock Count

* + Even after importing data, it's a best practice to perform an initial physical count to ensure accuracy.
  + Use the "Stock Count" feature to record the physical quantity of each item on your shelves. This will sync the system's data with your real-world inventory.

**5. Folder Structure**

SB-Works/

|-- client/ # React frontend |\_\_components/

L\_\_ pages/

|\_\_ server/ # Node.js backend

|\_\_routes/

|\_\_ models/

|\_\_ controllers/

A good folder structure is essential for organizing documentation, especially when it's part of a larger project. It helps keep files neat, makes them easy to find, and ensures consistency. For a project like "Store Manager Inventory Tracking," you can use a clear, hierarchical structure.

Here's a suggested folder structure that you can create within your project's main directory.

Explanation of Folders:

Documentation**/**: This is the main folder for all your written documentation.

Inventory\_Tracking\_Documentation.docx: This is your primary Word document, containing all the sections you've been working on (Project Overview, Architecture, Setup Instructions, etc.). Keeping it at the top level of this folder makes it easy to find.

/images/: A sub-folder specifically for all the images, screenshots, diagrams, and flowcharts that are inserted into your main documentation. Giving them a dedicated space keeps your main folder clean.

* + /resources/: A folder for any supplementary documents that are referenced in your main documentation. This could include a sample spreadsheet for data import, templates, or any other related files.

**6. Running the Application**

• Frontend: cd client

npm start

• Backend: cd server

npm start

• Access: Visit <http://localhost:3000>

Once your inventory management application is set up, running it is typically a straightforward process that involves daily, weekly, and periodic tasks to keep your inventory accurate. For a store manager, the key is to integrate the application into your regular workflow.

Common Running Applications

Many modern inventory applications are mobile-friendly and cloud-based, giving managers flexibility. Some popular examples include:

* Zoho Inventory: Offers a mobile app and web interface for comprehensive inventory and order management. It also provides features like barcode scanning and real-time alerts.
* Sortly: Known for its ease of use and visual inventory tracking with photos. It's great for small businesses that need a simple yet powerful way to track stock.
* Square Inventory: Integrates seamlessly with the Square POS system, making it an excellent choice for businesses already using Square for payments.

**7. API Documentation**

• User**:**

**– /api/user/register**

**– / api/user/login**

• Projects:

– /api/projects/create

– /api/projects/:id • Applications: /api/apply

• Chats:

– /api/chat/send

– /api/chat/:userId

API documentation is crucial for an inventory management system because it explains how different software components can communicate with each other. This is especially important for a store manager's system, as it needs to integrate with a Point of Sale (POS) system, a mobile app for stock counts, and potentially an e-commerce platform.

A standard API documentation should be structured to provide a clear, easy-to-use reference for developers. It typically includes an overview, authentication details, a list of endpoints, and example requests and responses.

1. API Overview

This section provides a high-level summary of the API's purpose and functionality.

* API Name: Store Manager Inventory API
* Description: A RESTful API that allows for

the programmatic management of a store's inventory. It provides endpoints for tracking products, monitoring stock levels, and generating inventory reports.

**8. Authentication**

• JWT-based authentication for secure login

• Middleware protects private routes

**9. User Interface**

• Landing Page

• Freelancer Dashboard

• Admin Panel

• Project Details Page

To user interference in an inventory tracking system, you need to describe how a user (like a store manager or employee) interacts with the system's interface and the actions they can perform. This section is often part of a larger document that explains the system's design and functionality.

Dashboard:

This is the first screen the user sees after logging in. It provides a quick overview of key metrics, such as:

* Current stock levels
* Low-stock alerts
* Sales performance over a selected period
* Recent inventory movements (e.g., new shipments received, recent sales)

Navigation Menu:

This is how the user moves through the system. A well-designed menu will have logical categories that mirror the store manager's workflow, such as:

* Products: To view, add, or edit product details.
* Inventory: To perform stock counts, manage shipments, and track transfers.
* Sales: To view sales history and reports.

**10. Testing**

• Manual testing during milestones

• Tools: Postman, Chrome Dev Tools

Testing a store manager's inventory tracking system is a crucial step to ensure it works correctly and reliably. The testing process should be systematic, covering all aspects of the system from basic functionality to security.

1.Functional Testing

Functional testing checks if the system performs all the tasks it's designed to do. This is a top priority.

Positive Testing: Check that the system works as expected with valid inputs

.

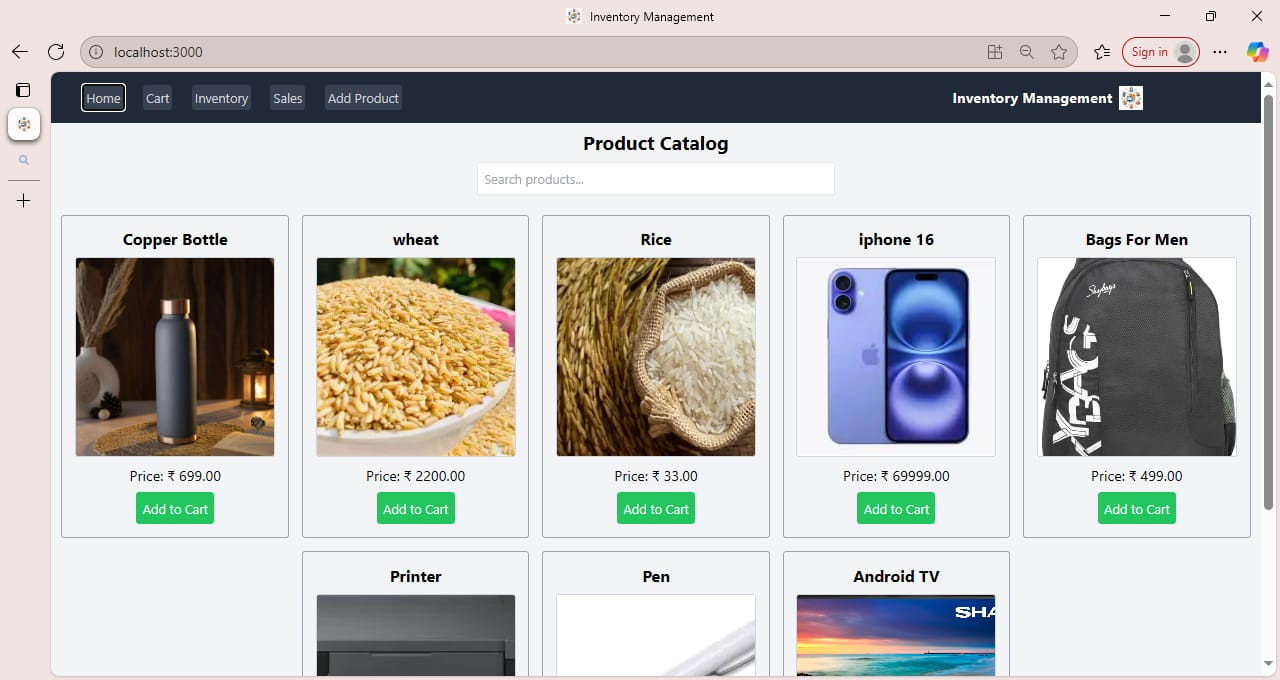
* + Can you successfully add a new product?
  + Does a sale at the Point of Sale (POS) accurately reduce the stock count?
  + Does a new shipment successfully increase thestock count?

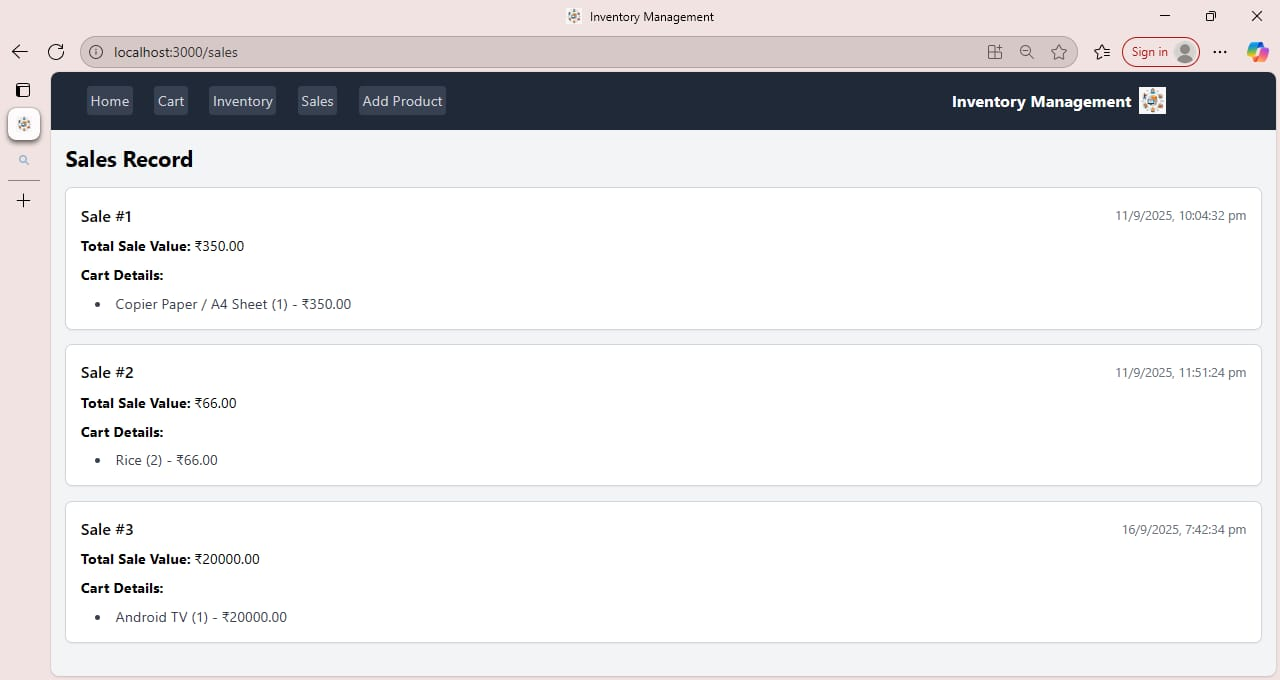
1. Negative Testing**:**

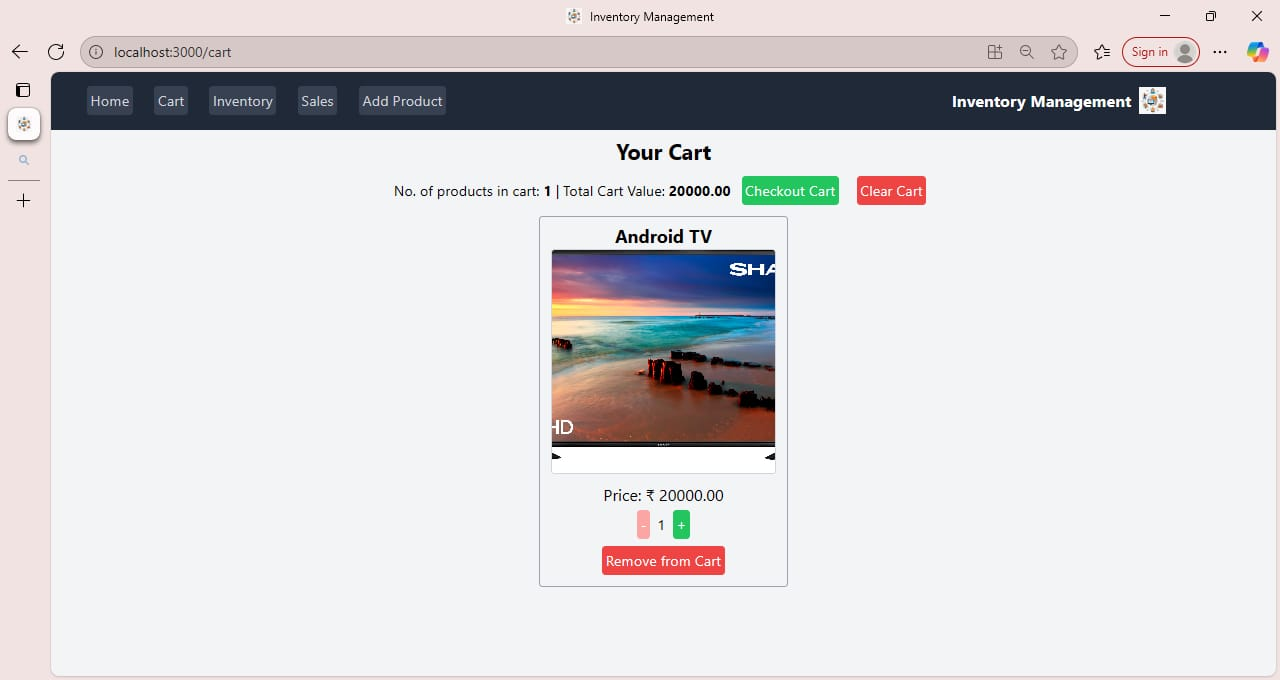
Check how the system handles incorrect or unexpected inputs**.**

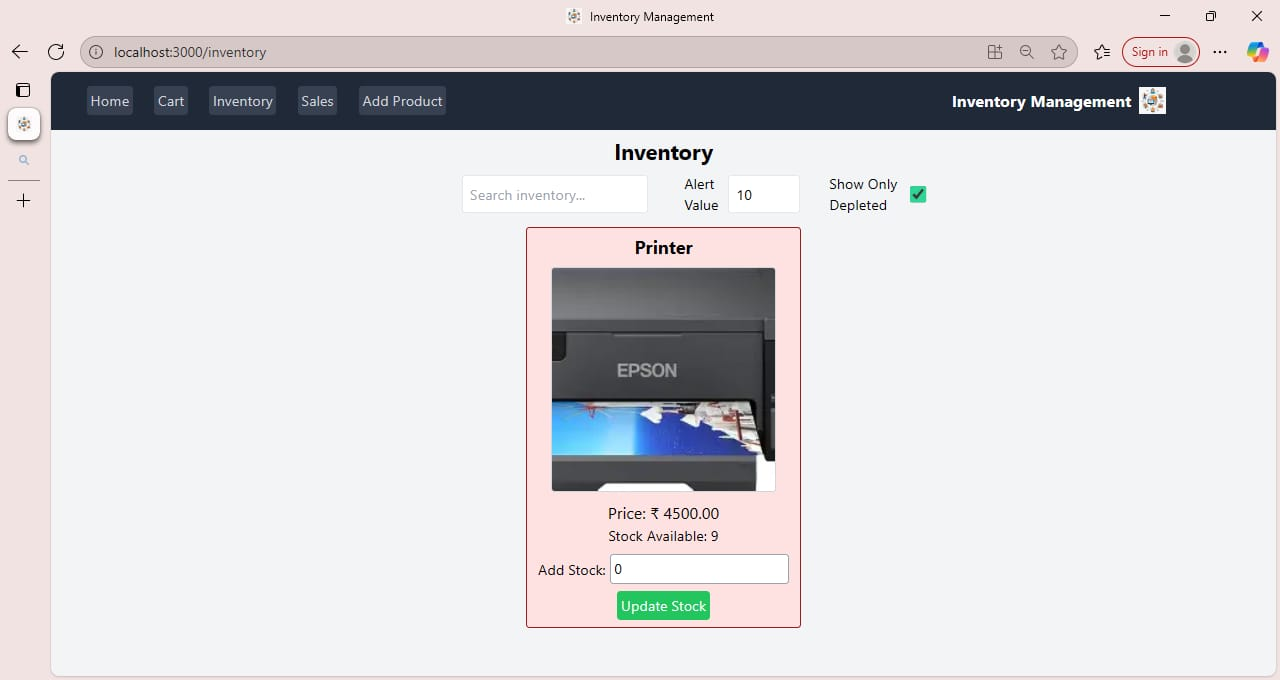
* What happens if you try to enter a negative stock quantity?
* Can you try to sell an item that is out of stock?

**11. Screenshots or Demo**

****







**12. Known Issues**

1. Data entry errors

Manual input of product details can lead to inconsistent or incorrect records if validation is not thorough.

2. Limited reporting features

Current reports provide only basic stock summaries and do not support detailed analysis like sales trends or reorder forecasts.

3. No real-time synchronization

Updates to inventory may not be reflected immediately across multiple devices or user accounts.

4. User access control is basic

All users may have the same access rights, which could lead to accidental modifications or unauthorized changes.

5. No automated alerts

There is no system to notify the manager when stock is low, expired, or due for reorder.

6. Backup and data recovery not integrated

In case of data corruption or accidental deletion, restoring data could be difficult without manual intervention.

Allow scheduled backups and easy recovery to prevent data loss and ensure continuity.

**13. Future Enhancements**

1. Implement barcode scanning

Use barcode or QR code integration to speed up product entries and reduce manual errors.

2. Add role-based access control

Create different user roles (Admin, Staff, Viewer) to control permissions and protect sensitive information.

3. Introduce real-time inventory tracking

Enable automatic updates across devices and users for accurate, up-to-date stock information.

4. Enable notifications and alerts

Add automated alerts for low stock levels, expiry dates, and reorder thresholds to help managers stay proactive.

5. Develop advanced reporting tools

Provide sales insights, purchase history, and demand forecasting to support better decision-making.

6. Integrate backup and restore features

Allow scheduled backups and easy recovery to prevent data loss and ensure continuity.

END