**Project Documentation**

# Store Manager: Keep Track of Inventory

# 1. Introduction

* **Project Title:** Store Manager: Keep Track of Inventory .
* **Team ID** : NM2025TMID38465
* **Team Leader:** Mirthula S

Mirthulasenthikumar5188@Gmail.Com

* **Team Members:** Yavanthika R

yavanthiraj@Gmail.Com

* **Team Members:**  Yogeswari M

Pm5341593@Gmail.Com

* **Team Members:** Tharani R

Tharaniprasad.official@gmail.com

# 2. Project Overview

* Purpose: The **purpose** of this inventory management system is to provide store managers with a streamlined, efficient way to **monitor, manage, and optimize inventory operations**. It enables better decision-making, prevents stockouts or overstocking, and ensures a seamless sales process.
* **Features:**
* **Maintain Optimal Stock Levels:**
  + Keep track of available products in real-time.
  + Automatically update inventory on sale or restocking.
* **Streamline Sales Process:**
  + Allow products to be added to a cart.
  + Facilitate easy checkout while automatically updating inventory and generating sale records.
* **Improve Stock Visibility & Control:**
  + Highlight depleting stock levels with alerts to avoid out-of-stock situations.
  + Easily search for and manage inventory items.
* **Enhance Record Keeping:**
  + Store detailed sale records including date, products sold, and sale value for reporting and analysis.
* **Simplify Product Management:**
  + Add new products with essential details (name, image, price, tags, stock).
  + Ensure product catalog is always up to date.

# 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

# 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

# 5.Folder Structure

# client/

# │

# ├── node\_modules/ # Auto-generated dependencies

# ├── public/ # Static public files (HTML, images, etc.)

# ├── src/ # Source code directory

# │

# ├── .gitignore # Git ignored files

# ├── package.json # Project metadata and dependencies

# ├── package-lock.json # Locked versions of dependencies

# ├── tailwind.config.js # Tailwind CSS configuration

# ├── README.md # Project overview and instructions

src/

│

├── components/ # Reusable UI components grouped by domain

│ ├── Cart/

│ │ ├── Cart.jsx

│ │ └── CartItem.jsx

│ │

│ ├── Inventory/

│ │ ├── Inventory.jsx

│ │ └── Product.jsx

│ │

│ ├── Product/

│ │ ├── Product.jsx

│ │ ├── ProductCatalog.jsx

│ │ └── ProductList.jsx

│ │

│ ├── Sales/

│ │ ├── SaleRecord.jsx

│ │ └── Sales.jsx

│

│ ├── AddProduct.jsx # Component for adding new products

│ └── NavBar.jsx # Top navigation bar

│

├── context/ # React Contexts for global state (auth, cart, etc.)

├── hooks/ # Custom React Hooks

├── pages/ # Page-level components or routes

│

├── App.js # Main app component

├── App.css # App-level CSS

├── App.test.js # App test file (Jest)

├── index.js # Entry point of the React app

├── index.css # Global styles

├── Layout.jsx # Layout wrapper (optional)

├── reportWebVitals.js # Web Vitals reporting

├── setupTests.js # Test setup file

# 6. Running the Application

* **Frontend:**

cd client npm start • **Backend:**

cd server npm start

* **Access:** Visit http://localhost:3000

# 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

# 8. Authentication

* JWT-based authentication for secure login
* Middleware protects private routes

# 9. User Interface

### **1.Inventory Management**

* Maintains healthy stock levels.
* Automatically tracks quantity of each product.
* Ensures timely restocking to avoid shortages.

### **2. Stock Updates**

* Inventory decreases automatically when products are sold.
* Inventory increases when new stock is added.
* Real-time sync between sales and stock levels.

### **3. Cart Functionality**

* Products can be added to a cart for a particular sale.
* Quantity per product can be adjusted.
* Temporary storage before checkout.

### **4. Checkout at Cart**

* On checkout:
  + Cart is cleared.
  + Inventory updates (stock reduced based on sold products).
  + Sale record is stored for reporting.

### **5. Adding New Products**

* Admin/Storekeeper can add new products with:
  + Product name
  + Image URL
  + Price
  + Stock quantity
  + Tags (category, type, etc.)

### **6. Alert View for Depleting Stock**

* Products with stock below a threshold show in **red background**.
* Alert count/threshold can be configured (e.g., alert if stock < 5).
* Helps prioritize reordering.

### **7. Product Search Functionality**

* Search products in:
  + Inventory (current stock).
  + Product catalog (all available products).
* Search by name, tag, or category.

# 10. Testing

### **1. Manual Testing during Milestones**

* Testing will be performed at each development milestone (e.g., after completing Inventory Module, Cart Module, Checkout Module).
* Ensures early detection of bugs and smooth integration of features.
* Includes functional testing, UI/UX testing, and edge-case validation.

### **2. Tools**

* **Postman** → For testing APIs (adding products, updating stock, checkout, sales records).
* **Chrome DevTools** → For debugging frontend issues (UI responsiveness, console errors, network requests, performance checks).

1. **Screenshots**

**Product Catalog:**

A screenshot of a computer

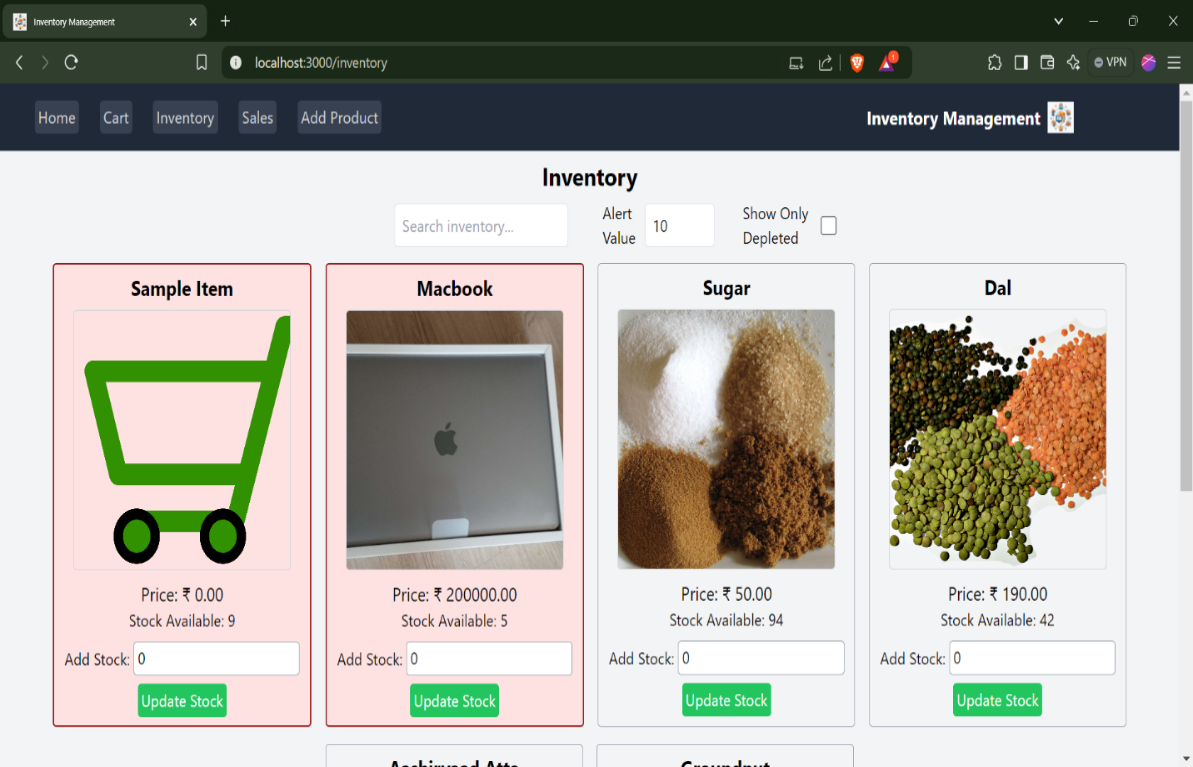
Description automatically generated

**Cart:**

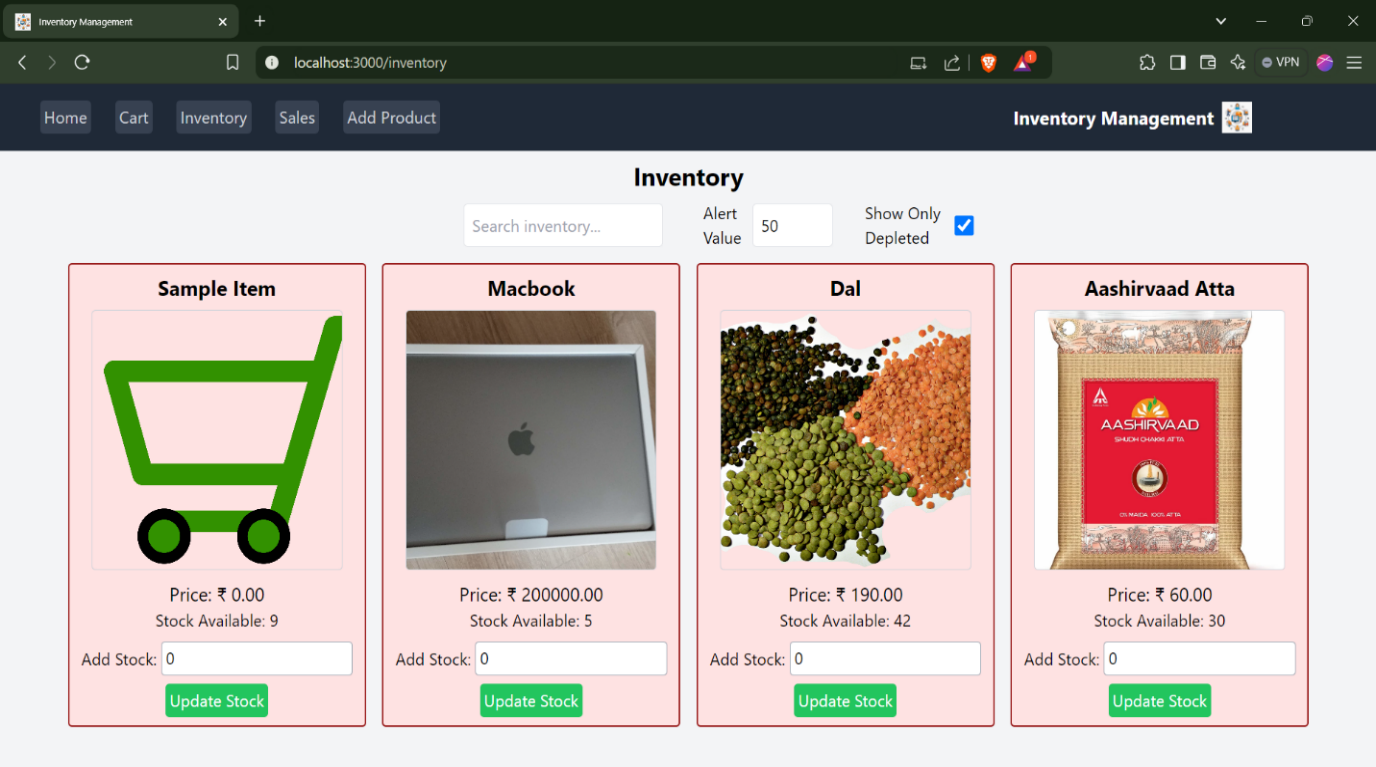
A screenshot of a computer

Description automatically generated

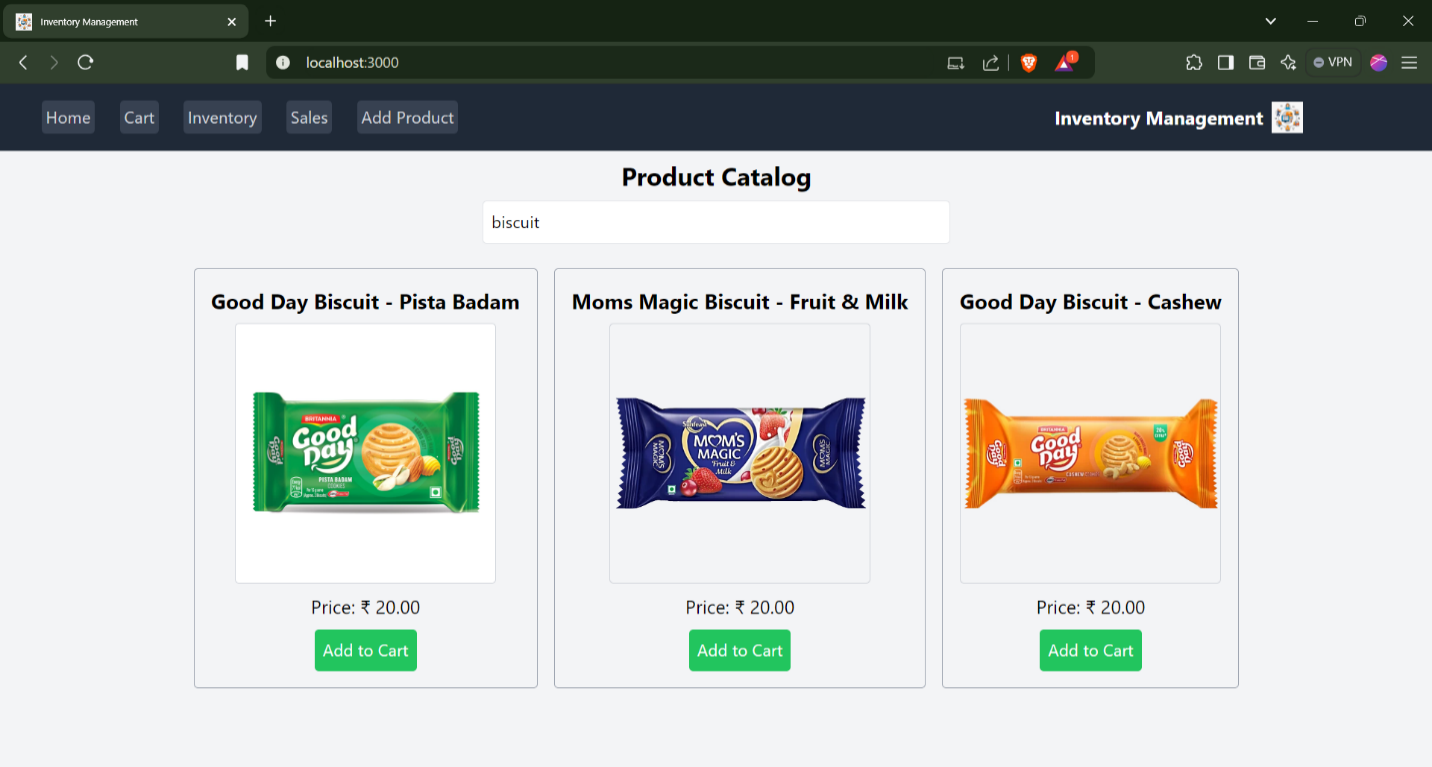
**Inventory:**

****

**Depleted Stock:**

****

**Search Functionality:**

****

**Sales:**

**A screenshot of a computer

Description automatically generated**

**Add Product:**

**A screenshot of a computer

Description automatically generated**

1. **Known Issues**

The current system has a few known issues that need attention. At times, stock updates may experience slight delays, requiring a manual page refresh for accurate inventory levels to display. The search functionality is case-sensitive, meaning that searching for “apple” and “Apple” produces different results, which may affect usability. Additionally, the stock depletion alert threshold is static and needs to be manually adjusted rather than dynamically adapting to usage trends. Another limitation is that cart data is not persistent—if the page is refreshed before checkout, items in the cart are cleared. Finally, sales reporting is limited, as it does not yet support advanced analytics such as filtering by date, category, or customer.

# 13.Future Enhancements

In the future, the system can be enhanced with several improvements to increase efficiency and usability. Dynamic stock alerts can be introduced, allowing the system to automatically adjust the threshold for low-stock warnings based on sales patterns. Cart persistence across sessions can be implemented so that users do not lose items on page refresh or logout. Advanced reporting and analytics features, such as sales trends, customer insights, and category-wise performance, can be added to provide deeper business intelligence. Integration with barcode scanners or QR codes can further simplify stock management and checkout processes. Additionally, role-based access control for admins, managers, and staff can strengthen security and streamline responsibilities. In the long term, integrating the system with accounting modules and enabling multi-store inventory tracking would significantly expand its scalability and usefulness.