Project Documentation

Store Manager: Keep Track of Inventory

# Introduction

## Project Title: Store Manager: Keep Track of Inventory

* + **Team ID :- NM2025TMID38908**
  + **Team Leader:**

|  |  |  |
| --- | --- | --- |
| Name | Mail id | Work |
| ANANDAN M | [anandgamer218@gmail.com](mailto:anandgamer218@gmail.com) | CODING CREATER |

## Team Members:

|  |  |  |
| --- | --- | --- |
| Name | Mail id | Work |
| DHANASEELAN T | [dhaseelant979@gmail.com](mailto:dhaseelant979@gmail.com) | OPERATOR |
| AADESH JAIN P | [ajmobileservice1@gmail.com](mailto:ajmobileservice1@gmail.com) | DOCUMENT CREATER |
| KEERTHANA M | [Keerthanarevathi2305@gmail.com](mailto:Keerthanarevathi2305@gmail.com) | DEMO VIDEO AND LINK GENRATOR |

# Project Overview

* + **Purpose:** The Store Manager project is designed to help businesses efficiently monitor, manage, and update their inventory in real time.

## Key Features:

## ****Dashboard Overview****

## \* Displays total products, low-stock items, recent activity, and notifications.

* **Product Management**

\* Add, update, or delete product details (name, category, price, quantity, etc.).

\* Assign product images and barcodes.

* **Inventory Tracking**

\* Monitor current stock levels in real-time.

\* Automatically update inventory after sales or restocking.

* **Low Stock Alerts**

\* Receive alerts/notifications when product quantity falls below the set threshold.

* **Search and Filter**

\* Quickly find products by name, category, SKU, or supplier.

\* Filter inventory by stock level, price range, etc.

# Architecture

# The system architecture of Store Manager: Keep Track of Inventory is structured to provide a reliable, scalable, and user-friendly solution for managing store inventory. It follows a modular three-tier architecture, which is divided into the following components:

**1. Presentation Layer (User Interface)**

**This layer is responsible for user interaction. It provides a clean and intuitive interface that allows store managers and staff to:**

* **Add, update, or remove inventory items**
* **View stock levels and low stock alerts**
* **Generate inventory reports**
* **The UI can be developed using web technologies (like HTML, CSS, JavaScript) or mobile frameworks for enhanced accessibility.**

2. Business Logic Layer (Application Layer)

The core functionality of the system resides in this layer. It processes user inputs, applies business rules, and handles operations such as:

* Inventory validation and updates
* Stock level monitoring and automatic alert generation
* Managing supplier details and restocking logic
* Report generation and analytics

This layer acts as a bridge between the UI and the database, ensuring smooth data flow and process control.

3. Data Access Layer (Database)

This layer handles data storage and retrieval operations. It maintains a structured and secure database containing:

* Product details (ID, name, category, quantity, price)
* Transaction history (sales, restocks, returns)
* Supplier and order information

A relational database (e.g., MySQL, PostgreSQL) or a NoSQL database (e.g., MongoDB) can be used, depending on system requirements.

4. Optional Enhancements and Integrations

To make the system more robust and future-ready, the following enhancements can be integrated:

* Barcode/RFID scanning for quick item lookup and updates
* Cloud storage for remote access and multi-location support
* API integration with POS or ERP systems
* AI-based demand forecasting for better stock planning

# Setup Instructions

# Before setting up the Store Manager: Keep Track of Inventory system, ensure that all necessary prerequisites such as a compatible operating system, Node.js, a database system (like MySQL or MongoDB), and any required package managers (e.g., npm or yarn) are installed. To begin, clone the repository from the project’s source control platform using Git. Next, set up the backend by installing all dependencies and configuring the database connection. Once completed, run the backend server to ensure it's operational. After that, proceed to set up the frontend, install the required packages, and run the development server to connect with the backend. Optionally, you may seed the initial data into the database to populate sample products, categories, and suppliers for testing purposes. Once everything is running, you can access the app via your browser at the specified local or hosted URL. Finally, for deployment, you can build the project for production, which compiles and optimizes both frontend and backend code for a live environment. This structured setup ensures a smooth and efficient development and deployment process.

## Prerequisites:

1. Clone the Repository

#### **Setup Backend**

#### Setup Frontend

#### Optional: Seed Initial Data

#### Access the App

#### Build for Production (Optional)

## Installation Steps:

## To set up the development environment for the Store Manager: Keep Track of Inventory system, follow these installation steps. First, ensure you have a suitable code editor—we recommend Visual Studio Code (VS Code), which you can download and install from here

## . Next, you need to install Node.js, which is essential for running both frontend and backend components of the application. You can download Node.js from the official website

## , and it's recommended to install the LTS (Long-Term Support) version for stability. Once both VS Code and Node.js are installed, you can proceed to clone the repository, install project dependencies, and follow the setup instructions to get the application up and running.

# VS Code: Install from here

# Node.js: Install from here

# Folder Structure

# ├── frontend/ # Frontend-related files (React)

# │ ├── node\_modules/ # Dependencies installed by npm

# │ ├── public/ # Static files (index.html, favicon, etc.)

# │ ├── src/ # All React components and code

# │ │ ├── App.js # Main React component that fetches data

# │ │ ├── index.js # Entry point of the React application

# │ │ ├── App.css # Styling for App component (optional)

# │ │ └── components/ # Any sub-components you create

# │ ├── package.json # Contains metadata and dependencies for the frontend

# │ ├── package-lock.json # Lock file for the frontend dependencies

# │ └── README.md # Optional: Description of the frontend setup

# ├── .gitignore # Lists files/folders to ignore for Git

# └── README.md # Optional: Project overview or instructionsRunning the Application

## Frontend:

cd client npm start • **Backend:**

cd server npm start

* + **Access:** Visit [http://localhost:3000](http://localhost:3000/)

To run the Store Manager: Keep Track of Inventory system locally, begin by starting both the frontend and backend servers. First, navigate to the frontend directory by running cd client in your terminal, then start the development server using the command npm start. This will launch the user interface in your default browser. Next, open a new terminal window and navigate to the backend directory using cd server, and start the backend server with npm start. Once both servers are running, you can access the application by visiting http://localhost:3000

in your web browser. This local environment allows you to interact with the full functionality of the application in development mode.To run the Store Manager: Keep Track of Inventory system locally, begin by starting both the frontend and backend servers. First, navigate to the frontend directory by running cd client in your terminal, then start the development server using the command npm start. This will launch the user interface in your default browser. Next, open a new terminal window and navigate to the backend directory using cd server, and start the backend server with npm start. Once both servers are running, you can access the application by visiting http://localhost:3000

in your web browser. This local environment allows you to interact with the full functionality of the application in development mode.

# API Documentation

## User:

* + - /api/user/register
    - /api/user/login

## Projects:

* /api/projects/create
* /api/projects/:id • **Applications:** /api/apply
  + **Purpose:** The Store Manager project is designed to help businesses efficiently monitor, manage, and update their inventory in real time.

## Features:

## ****Dashboard Overview****

## \* Displays total products, low-stock items, recent activity, and notifications.

* + **Chats:**
* /api/chat/send
* /api/chat/:userId

\* Automatically update inventory after sales or restocking.

To run the Store Manager: Keep Track of Inventory system locally, begin by starting both the frontend and backend servers. First, navigate to the frontend directory by running cd client in your terminal, then start the development server using the command npm start. This will launch the user interface in your default browser. Next, open a new terminal window and navigate to the backend directory using cd server, and start the backend server with npm start. Once both servers are running, you can access the application by visiting http://localhost:3000

in your web browser. This local environment allows you to interact with the full functionality of the application in development mode.

# User Interface

# The user interface of the Store Manager: Keep Track of Inventory system is designed to be intuitive, responsive, and easy to navigate for both administrators and end users. It includes several key pages and UI components, each serving a distinct purpose in the inventory and sales workflow:

# 1. Home Page

# The Home Page serves as the landing screen for users and displays the full product catalog in a clean, grid-style layout. Each product card typically includes an image, name, category, price, and an "Add to Cart" button. At the top of the page, a search bar enables users to quickly filter products by name, category, or keyword, improving discoverability. This page is primarily focused on customer interaction and product visibility.

# 2. Cart Page

# The Cart Page allows users to view and manage the items they have selected for purchase. It displays a list of products added to the cart, including product names, prices, quantities, and subtotals. Users can update quantities or remove items as needed. At the bottom of the page, a total price calculation is shown along with a checkout button, which can be linked to a payment process or order confirmation flow. This page plays a critical role in the purchasing process.

# 3. Inventory Page

# The Inventory Page is primarily used by administrators or store managers. It provides a comprehensive table view of all products in stock, including details such as SKU, product name, category, quantity, price, and supplier (if applicable). From this page, users can quickly identify low-stock items, make adjustments to stock levels, or delete discontinued products. Filtering, sorting, and pagination options help manage large inventories efficiently. This page may also include buttons to export inventory data or trigger restock orders.

# 4. Sales Page

# The Sales Page presents a detailed summary of all transactions, including the date, products sold, total revenue, and payment status. It often includes charts or visualizations to help administrators track sales performance over time (e.g., daily, weekly, monthly). Each transaction entry may also include customer details, invoice numbers, and the status of delivery or fulfillment. This page helps store managers analyze performance trends and make data-driven business decisions.

# Testing

# Throughout the development of the Store Manager: Keep Track of Inventory system, a combination of manual and tool-assisted testing was conducted to ensure stability, usability, and correctness of functionality. Manual testing was performed during the implementation of each feature to validate user flows, identify bugs early, and confirm expected behavior. Postman was extensively used to test and verify the backend API endpoints, ensuring correct request handling, status codes, and response structures across different HTTP methods (GET, POST, PUT, DELETE). For frontend debugging, Chrome DevTools played a crucial role in inspecting elements, monitoring network requests, and troubleshooting layout and styling issues. Additionally, a quick cross-browser check was carried out to verify the application's appearance and functionality on popular browsers like Chrome, Firefox, and Edge. A basic performance review was also performed to identify potential bottlenecks and ensure smooth operation, particularly on the Home and Inventory pages.

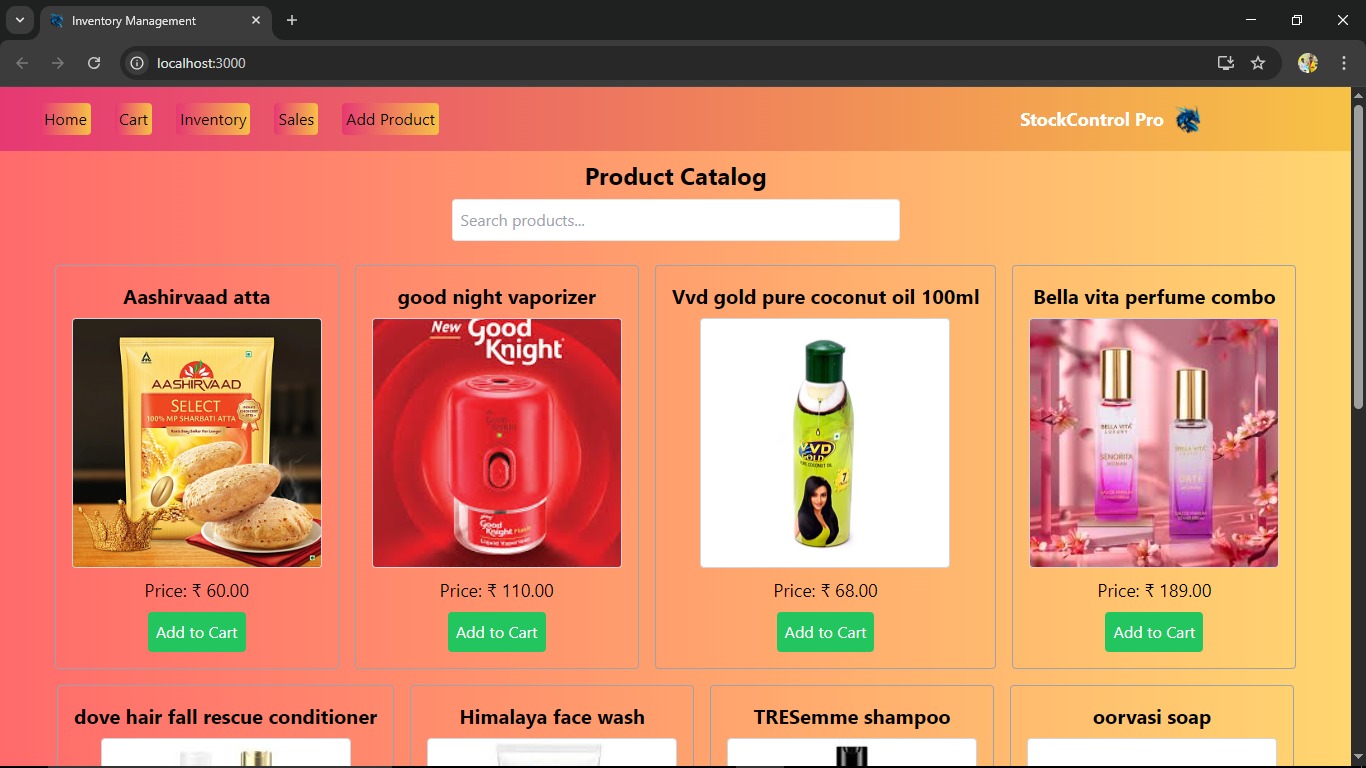
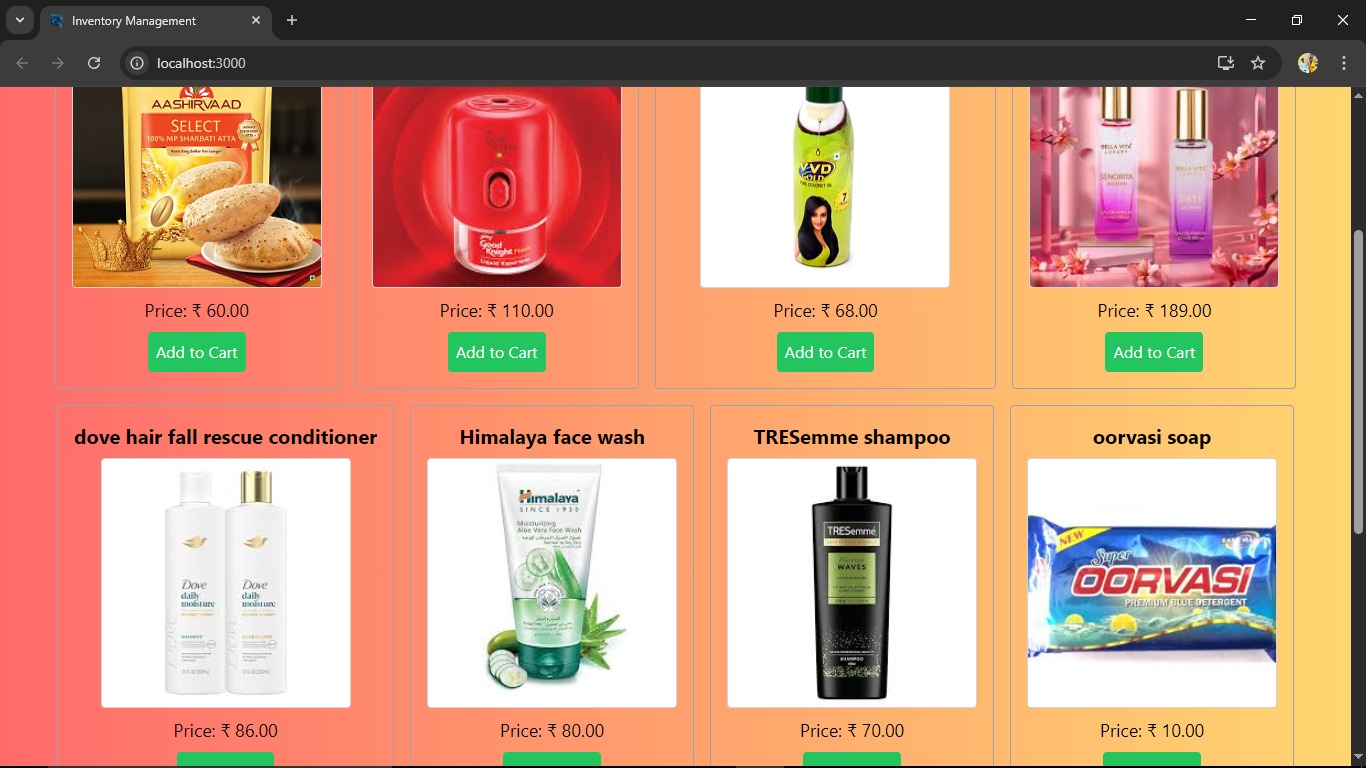
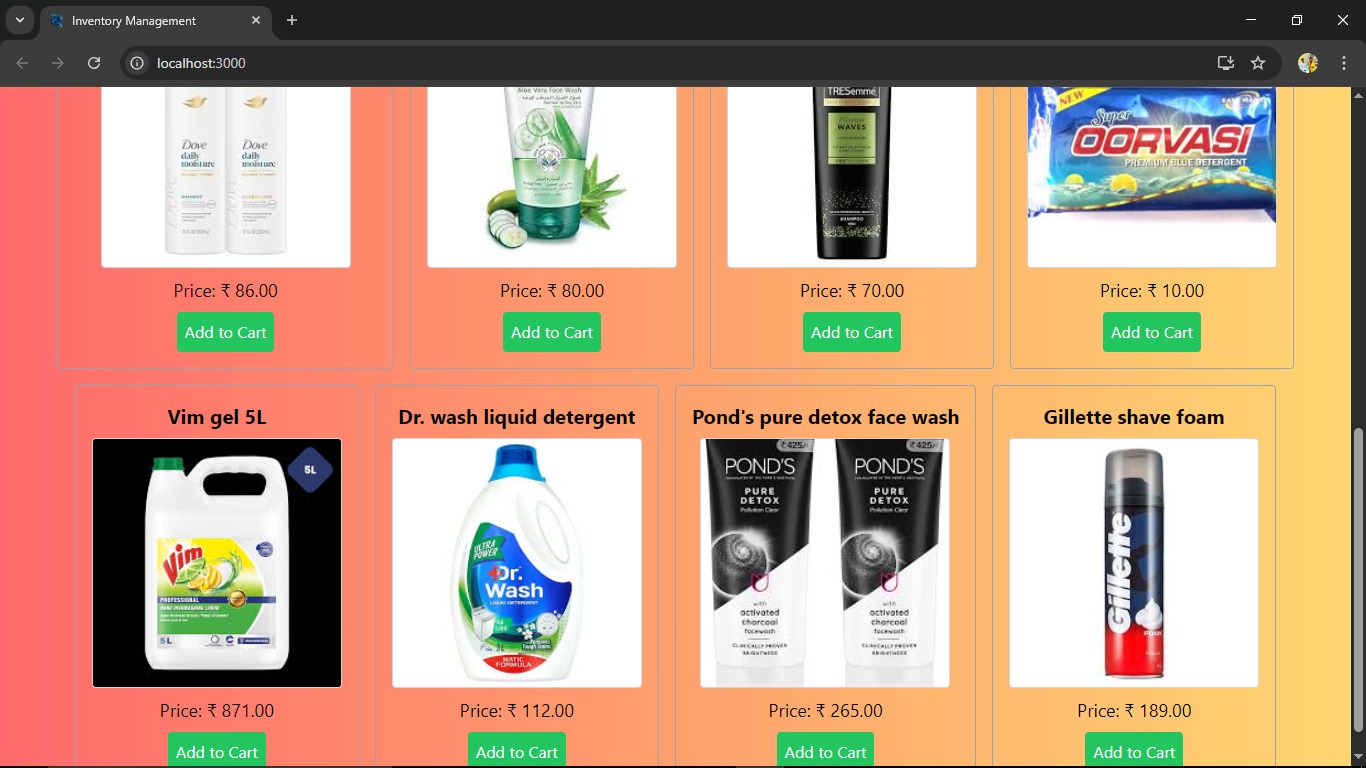
Performed manual testing while building each feature.

Used Postman to check your API calls.

Used Chrome DevTools to inspect and debug the user interface.

Did a quick cross-browser check and basic performance review.

1. **Screenshots or Demo**

* ****
* ****
* ****

1. **Known Issues**

While the development of the Inventory Management application is progressing well, several issues have been identified during the testing phase. One of the main concerns is the compatibility of the application with older web browsers, which may not fully support some modern JavaScript features used within the app. Additionally, the responsiveness of the UI on smaller screens could be improved, especially on mobile devices, as certain components may not scale properly. There are also occasional issues with data synchronization, where updates made to the inventory on one device may take a moment to reflect across all users in real-time. These challenges are actively being worked on, and we plan to address them in upcoming releases to ensure a smoother user experience.

1. **Future Enhancements**
2. **Real-Time Inventory Tracking**

* Description: Automatically update inventory levels as items are sold, returned, or restocked.
* Benefits: Prevents overselling, improves stock accuracy, and ensures up-to-date inventory levels.

1. **Mobile App Integration**

* Description: Allow inventory management through mobile devices (barcode scanning, restocking, alerts).
* Benefits: Enables on-the-go tracking and updates, useful for warehouse or floor staff.

**C. AI-Powered Demand Forecasting**

* Description: Use AI/ML algorithms to predict future stock needs based on historical sales, seasons, and trends.
* Benefits: Prevents overstocking and understocking, reduces waste, and increases profitability.

**D. Automated Reordering System**

* Description: Automatically reorder stock from suppliers when inventory falls below a predefined threshold.
* Benefits: Avoids stockouts and improves supply chain reliability.

**E. Advanced Analytics & Reporting**

* Description: Provide detailed reports on inventory turnover, slow-moving items, shrinkage, and more.
* Benefits: Informs better decision-making and highlights areas for improvement.

**12.CONCLUSION**

The Store Manager: Keep Track of Inventory system successfully addresses the essential needs of inventory management in a retail environment. By providing a streamlined interface for tracking products, managing stock levels, recording sales, and adding new items, the system enhances operational efficiency and reduces the risk of stockouts or overstocking. With its modular architecture and clean user interface, the application is both user-friendly and scalable, allowing for future enhancements such as barcode scanning, automated reordering, cloud deployment, and AI-based forecasting. The combination of a robust backend, responsive frontend, and practical tools for administrators makes this system a valuable solution for small to medium-sized businesses seeking to improve inventory control. Overall, the project demonstrates practical application of full-stack development principles and provides a solid foundation for further growth and integration.