**A Industrial Oriented Mini Project Report (CS755PC)**

**on**

# SMART INVENTORY MANAGEMENT SYSTEM

*Submitted*

*in partial fulfillment of the requirements for the award of the degree of*

## Bachelor of Technology

in

## Computer Science and Engineering

## (Artificial Intelligence & Machine Learning)

by

## Mr. Abhinav B S C Ambatipudi (21261A6601)

Under the guidance of

**Dr. M. RAMA BAI**

**(Professor & HoD)**



## Department of Emerging Technologies

Mahatma Gandhi Institute of Technology (A)

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) Kokapet(V), Gandipet(M), Hyderabad.

Telangana - 500 075.

## 2024 - 2025

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) GANDIPET, HYDERABAD – 500075, Telangana

## CERTIFICATE



This is to certify that the project entitled - “**SMART INVENTORY MANAGEMENT SYSTEM”** is being submitted by **ABHINAV B S C AMBATIPUDI** bearing Roll No. **21261A6601** in partial fulfillment of the requirements for the Minor- Project (CS755PC) in **COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide work carried out by him.

The results of the investigations enclosed in this report have been verified and found satisfactory.

**PROJECT GUIDE HEAD OF THE DEPARTMENT**

**Dr. M. Rama Bai Dr. M. Rama Bai**

(Professor) (Professor, Dept. of ET)

**External Examiner**

# DECLARATION

This is to certify that the work reported in this project titled ― “**SMART INVENTORY MANAGEMENT SYSTEM”** is a record of work done by me in the **Department of Emerging Technologies**, Mahatma Gandhi Institute of Technology, Hyderabad.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred to in the text. The report is based on the work done entirely by me and not copied from any other source.

**ABHINAV B S C AMBATIPUDI (21261A6601)**

# ACKNOWLEDGEMENTS

I would like to express my sincere thanks to **Dr. K. Chandra Mohan Reddy, Principal MGIT**, for providing the working facilities in college.

I wish to express my sincere thanks and gratitude to **Dr. M. Rama Bai, Professor and HoD**, Department of ET, MGIT, for all the timely support and valuable suggestions during the period of project.

I am extremely thankful to **Dr. M. Rama Bai, Professor and HOD**, and **Dr. Barnali Gupta Associate Professor,** Department of ET, MGIT, Industrial Oriented Mini Project coordinators, for their encouragement and support throughout the project.

I am extremely thankful and indebted to my internal guide **Dr. M. Rama Bai**, **Professor**, Department of ET, for her constant guidance, encouragement and moral support throughout the project.

Finally, I would also like to thank all the faculty and staff of ET Department who helped me directly or indirectly, for completing this project.

**ABHINAV B S C AMBATIPUDI (21261A6601)**

|  |  |
| --- | --- |
| **TABLE OF CONTENTS** |  |
| Certificate | i |
| [Declaration](#_bookmark0) | [ii](#_bookmark0) |
| [Acknowledgement](#_bookmark1) | [iii](#_bookmark1) |
| [List of Figures](#_bookmark2) | [v](#_bookmark2) |
| [List of Tables](#_bookmark3) | vi |
| [Abstract](#_bookmark4) | [vii](#_bookmark4) |
| 1. [**Introduction**](#_bookmark5) | [1](#_bookmark5) |
| 1.1 Motivation | 2 |
| 1.2 [Problem Definition](#_bookmark6) | 2 |
| 1. [**Literature Survey**](#_bookmark11) | 4 |
| 1. [**Proposed**](#_bookmark12) **System** | 7 |
| * 1. Problem Statement | 7 |
| * 1. Key Challenges | 7 |
| 3.3. Objectives | 8 |
| 3.4. Technologies used | 9 |
| 1. **Architecture & UML Diagrams** | 10 |
| * 1. Architecture | 10 |
| * 1. UML diagram | 12 |
| * 1. Data Flow | 13 |
| 1. **Testing & Results** | 16 |
| 1. **Conclusion & Future Scope** | 22 |
| **Bibliography** | 24 |
| **Appendix** | 26 |

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| Figure 4.1 | Architecture of Inventory Management System | 10 |
| Figure 4.2 | UML diagram of Inventory Management System | 12 |
| Figure 4.3 | Class diagram of Inventory Management System | 14 |
| Figure 5.3.1 | Home Page | 18 |
| Figure 5.3.2 | Inventory Data | 18 |
| Figure 5.3.3 | Addition of Stock | 19 |
| Figure 5.3.4 | Selling of Stock | 19 |
| Figure 5.3.5 | Removing of Stock | 20 |
| Figure 5.3.6 | Stock Visualization | 20 |
| Figure 5.3.7 | Critical Stock Alert | 21 |

# LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| Table 2.1 | Comparison of Literature survey | 5 |

**ABSTRACT**

This project presents a “**SMART INVENTORY MANAGEMENT SYSTEM”** tailored for small businesses, focusing on efficiency, user-friendliness, and cost reduction. Designed to address common challenges in inventory management, this system offers an integrated platform to streamline stock handling processes while improving operational efficiency.

Key features of the system include product catalog management, enabling businesses to organize and track their inventory effectively. It provides real-time stock level monitoring, with automated alerts for low inventory, ensuring businesses can proactively restock and avoid disruptions. The system also supports sales and purchase tracking, allowing businesses to maintain comprehensive records of transactions and monitor stock movement seamlessly.

A standout feature is the analytics dashboard, which provides valuable insights into inventory trends, helping businesses optimize stock levels and forecast demand. This data-driven approach empowers small business owners to make informed decisions, reducing overstocking or stockouts and minimizing overhead costs.

The solution is user-friendly, requiring minimal training, and is scalable to adapt to the evolving needs of small businesses. By leveraging smart notifications and actionable analytics, the system helps businesses maintain an efficient inventory cycle, improve customer satisfaction, and enhance profitability.

In conclusion, the Smart Inventory Management System provides a holistic approach to inventory management for small businesses, combining automation, analytics, and intuitive design. It is a practical solution aimed at improving productivity, reducing costs, and supporting sustainable business growth.

# INTRODUCTION

# Inventory management is a critical component of any business operation, directly influencing profitability, customer satisfaction, and overall efficiency. For small businesses, effective inventory management can be the key to staying competitive in a fast-paced market. Unlike large enterprises, small businesses often lack access to advanced inventory tools due to limited budgets and resources, leaving them reliant on manual or outdated systems prone to errors and inefficiencies. To address this gap, the Smart Inventory Management System for Small Businesses offers a modern, scalable, and user-friendly solution designed to optimize inventory processes while reducing operational costs.

# This project is developed with the primary aim of empowering small businesses to achieve a higher degree of control and insight into their inventory operations. By integrating core functionalities such as product catalog management, stock level monitoring, sales and purchase tracking, and analytics-driven decision-making, this system provides a comprehensive toolset tailored to the unique challenges faced by small businesses.

# One of the standout features of this system is real-time stock level tracking. Through this functionality, users receive automated notifications and alerts when inventory levels fall below a pre-defined threshold. This ensures timely replenishment, preventing potential stockouts or overstock situations that could disrupt operations or tie up valuable resources. Furthermore, the sales and purchase tracking module allow businesses to maintain detailed records of transactions, providing a clear view of stock movement and enabling better financial oversight.

# The inclusion of an analytics dashboard adds significant value by transforming raw data into actionable insights. Small business owners can use these insights to identify sales trends, predict demand, and optimize stock levels accordingly. This data-driven approach not only enhances decision-making but also supports long-term strategic planning, allowing businesses to remain agile and responsive in a dynamic market.

## Motivation

The motivation behind the **Smart Inventory Management System** stems from the challenges small businesses face in efficiently managing inventory due to limited resources and reliance on manual processes. Ineffective inventory management can lead to stockouts, overstocking, and financial losses. This project aims to empower small businesses by providing an affordable, user-friendly, and data-driven solution. By integrating automation, real-time tracking, and analytics, the system helps streamline operations, reduce costs, and enhance decision-making. Ultimately, it supports small businesses in achieving operational efficiency and sustainable growth, enabling them to compete effectively in today’s dynamic market.

## Problem Definition

Small businesses often struggle with inefficient inventory management due to limited resources and reliance on manual or outdated systems. These challenges lead to common issues such as stockouts, overstocking, inaccurate tracking, and increased operational costs. Without access to real-time data or actionable insights, businesses face difficulties in forecasting demand and optimizing stock levels, resulting in lost sales or excess inventory. Additionally, the lack of automation makes inventory processes time-consuming and error prone. This project addresses these problems by developing a **Smart Inventory Management System** that integrates real-time tracking, smart notifications, and analytics to streamline inventory processes and improve operational efficiency.

## Existing System

In most small businesses, inventory management relies on manual processes or basic tools like spreadsheets and standalone software. While these methods are affordable and simple to implement, they often lack the capabilities needed for efficient, real-time inventory tracking and decision-making. The reliance on outdated systems makes managing stock levels and predicting trends time-consuming and error-prone, limiting the business's ability to scale effectively.

### Main Points about the Existing System

* + 1. **Manual and Error-Prone Processes** Current systems often require manual data entry, leading to errors in stock records. This results in discrepancies, delayed decision-making, and increased risk of stockouts or overstocking.
    2. **Lack of Real-Time Monitoring** Existing solutions lack real-time tracking and notifications. Business owners cannot respond quickly to low inventory levels or sudden demand surges, impacting operational efficiency.
    3. **Limited Analytics and Forecasting** Basic inventory tools fail to provide actionable insights or demand forecasts. Without data-driven analytics, businesses struggle to optimize stock levels, leading to inefficient resource utilization and missed opportunities for growth.

## Proposed System

The proposed **Smart Inventory Management System** offers a modern, automated solution to overcome the limitations of existing systems. It integrates **real-time stock tracking** with smart notifications, enabling businesses to proactively manage inventory and avoid stockouts or overstocking. The system includes **product catalogue management** and **sales and purchase tracking** to streamline inventory processes and maintain accurate records. An **analytics dashboard** provides actionable insights into inventory trends, helping businesses optimize stock levels and forecast demand. Designed to be user-friendly and scalable, the proposed system empowers small businesses to enhance efficiency, reduce operational costs, and support sustainable growth through data-driven decision-making.

## Requirements Specification

### Software Requirements

**Language** Python 3.6

**Operating system** Windows / Linux / macOS

**Tools** Anaconda Navigator, Jupyter Notebook, Numpy, Pandas, Matplotlib, Plotly

# LITERATURE SURVEY

* 1. **Scope of the Project:**

The **Inventory Management System (IMS)** is a comprehensive application designed to streamline inventory operations for businesses, ensuring efficient stock tracking, replenishment, and optimization. The primary scope of this project encompasses the automation of key inventory processes such as stock addition, sales tracking, and removal of obsolete inventory. The system aims to enhance accuracy by minimizing human errors, providing real-time updates, and maintaining a centralized database for all inventory-related data. It features functionalities such as data visualization, where inventory trends and performance metrics are presented through interactive charts and graphs to support data-driven decisions.

Additionally, the IMS incorporates critical stock alerts, notifying users when stock levels fall below predefined thresholds, thereby preventing disruptions due to stockouts. The system also enables the generation of detailed reports, facilitating better planning and auditing. Designed for scalability and adaptability, the IMS is suitable for a wide range of industries, from small-scale businesses to large enterprises, as it can handle varying inventory sizes and complexities.

The user-friendly interface ensures accessibility for both technical and non-technical users, while the backend leverages technologies such as Streamlit and SQLite for robust data management and real-time processing. Security is prioritized through controlled access and encryption mechanisms to safeguard sensitive inventory data. By reducing manual intervention and increasing operational efficiency, the IMS contributes to significant cost savings and productivity improvements for organizations.

This project’s scope extends to future integrations with emerging technologies like machine learning for demand forecasting, IoT for real-time stock monitoring, and cloud-based solutions for enhanced accessibility and collaboration. Overall, the IMS is positioned as a valuable tool for optimizing inventory management, empowering businesses to achieve better control over their resources and drive growth.

**Table 2.1 Comparison of Literature survey**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.NO** | **YEAR** | **AUTHOR** | **TITLE** | **TECHNIQUES** | **ADVANTAGES** | **DISADVANTAGES** |
| 1 | 2018 | Jeremiah Nyaga Wanjira and Dr. John Mungai Njagiru | Inventory management practices and financial performance of small and medium scale enterprises | * Statistical Analysis and Modeling * Demand Forecasting Models | * Improved Decision-Making * Scalability * Cost Reduction * Efficiency and Accuracy | * Implementation Costs * Complexity * Data Dependency * Cybersecurity Risks |
| 2 | 2015 | Hazem Kaylani and Abeer Qaderi | Performance improvement of inventory management system processes by an automated warehouse management system | * Custom Warehouse Management Software * Product Lifecycle Management | * Improved Efficiency * Enhanced Reliability * Space Optimization | * Initial Implementation Cost * Training Requirements * System Downtime Risks |
| 3 | 2019 | Khairy A.H. Kobbacy and Yansong Liang | Towards the development of an intelligent inventory management system | * Automatic Demand and Lead Time Pattern Identification * Statistical Testing for Demand Identification | * Cost Reduction * Improved Accuracy * Flexibility * Real-World Applicability | * Complexity of Implementation * Data Dependency * Cost of System Development * Continuous Monitoring and Adjustment |

# PROPOSED SYSTEM

# Problem Statement

Managing inventory is a critical challenge for businesses of all sizes. Traditional methods, such as manual tracking and spreadsheets, are prone to human error, inefficiency, and lack of real-time updates, often leading to overstocking or stockouts. These issues disrupt supply chains, increase operational costs, and reduce customer satisfaction.

The goal of the Inventory Management System (IMS) is to streamline inventory processes by providing an intuitive and automated solution. The IMS, implemented using Streamlit, offers features for tracking stock levels, updating inventory in real time, generating insightful reports, and sending low-stock alerts. The application aims to simplify inventory management for small and medium-sized businesses by reducing errors, enhancing visibility, and improving decision-making.

# Key Challenges

# Data Management

# Designing an efficient database schema to store inventory data securely. Handling large datasets if the application scales to accommodate multiple users or businesses. Implementing proper indexing for quick retrieval and updates of inventory data.

# Real-Time Functionality

# Implementing real-time updates for inventory changes (e.g., reflecting stock changes immediately when a transaction occurs). Syncing data across multiple users or devices to avoid inconsistencies.

# Scalability and Performance

# Designing the application to handle more users and larger inventories as the system grows. Optimizing queries and operations to maintain performance.

# Objectives

# The primary objective of the IMS is to streamline and simplify the management of inventory for businesses, enhancing efficiency and reducing manual effort. The application aims to provide an intuitive platform for tracking, updating, and analysing stock levels, ensuring real-time data visibility.

# Accurate Stock Tracking

# Minimize errors in inventory records by providing automated updates for stock additions, removals, and adjustments.

# Real-Time Monitoring

# Enable real-time visibility into stock levels, helping users make informed decisions to avoid overstocking or stockouts.

# User-Friendly Interface

# Design an intuitive and accessible interface using Streamlit, ensuring ease of use for non-technical users.

# Low-Stock Alerts

# Implement alert systems to notify users of critical inventory levels, ensuring timely restocking and uninterrupted business operations.

# Comprehensive Reporting

# Generate insightful reports on inventory usage, trends, and forecasts to support strategic decision-making.

# Data Security

# Protect sensitive inventory data through secure authentication mechanisms and role-based access control.

# Scalability

# Ensure the application can adapt to growing business needs, handling increased inventory data and multiple users efficiently.

# Technologies Used

# Frontend

# Streamlit is a Python-based framework for creating interactive, data-driven web applications. It provides an easy way to design user-friendly interfaces without requiring extensive frontend development knowledge. With Streamlit, you can display data tables, charts, and interactive forms, making it ideal for inventory management applications.

# Backend

# Python is used to implement the business logic of the application. It handles inventory operations like adding, updating, or deleting stock records, interacts with the database, and processes data for reports and visualizations. Python's extensive libraries and community support make it a versatile choice for backend development.

# Database

# SQLite is a lightweight database that stores all inventory data locally in a file-based system. It is easy to set up and use, making it suitable for small-scale applications. SQLite ensures quick read/write operations, ensuring seamless inventory management for the IMS.

# Data Visualization

# Pandas is used for data manipulation, while Matplotlib and Plotly are employed to create visualizations like bar graphs, pie charts, and trend lines. These tools help users analyze inventory trends, usage patterns, and stock predictions, making decision-making easier.

# Deployment

# Deployment platforms like Streamlit Cloud, Heroku, or Render allow you to host the IMS online, making it accessible from any device with internet access. These platforms provide easy deployment pipelines and scalability options.

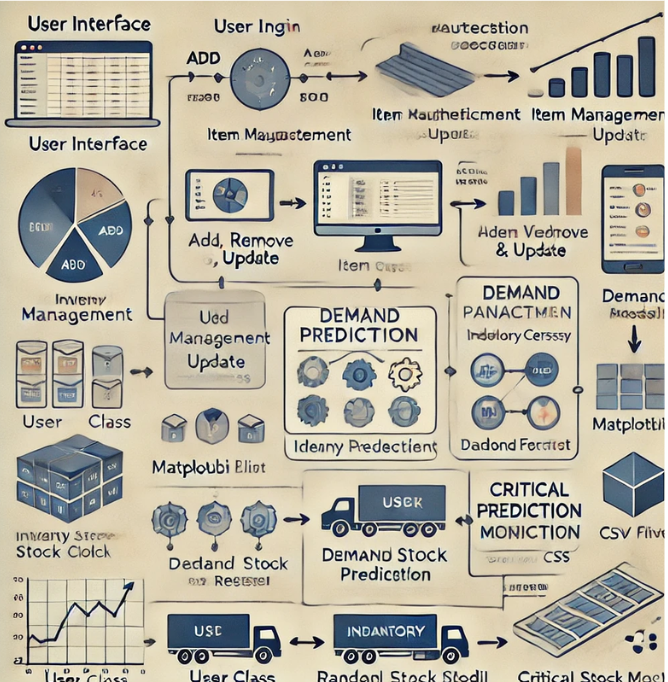
# Version Control

# Git and GitHub enable version control, allowing you to track changes in your application, collaborate with others, and manage code efficiently. They are essential tools for maintaining the codebase and rolling back changes if needed.

# ARCHITECTURE & UML DIAGRAMS

# Architecture

The architecture of the Inventory Management System (IMS) project is designed to provide a seamless and efficient solution for managing inventory operations. It follows a modular and layered structure to ensure scalability, maintainability, and robustness. At the core, the **frontend** is developed using Streamlit, offering an interactive and user-friendly interface for data visualization and inventory manipulation. The **backend**, implemented in Python, acts as the intermediary between the frontend and the database, handling business logic, data processing, and application workflows. The **database layer** utilizes SQLite for lightweight, file-based data storage, ensuring efficient management of inventory records, including stock levels, item details, and transaction history.



**Figure 4.1 Architecture of Inventory Management System**

* + 1. **User Interface**

The application starts with a clean and interactive interface designed using Streamlit, allowing users to view and interact with inventory data through tables, charts, and input forms.

* + 1. **User Login**

A secure login system ensures authentication and access control. Only authorized users can perform specific tasks based on their roles.

* + 1. **Item Management**

This involves adding, removing, or updating inventory items. Users can manage stock quantities and item details through intuitive forms.

* + 1. **Inventory Management**

Tracks the overall inventory status, presenting data visually with pie charts and graphs to analyse stock distribution and trends.

* + 1. **Demand Prediction**

Using tools like Pandas or machine learning models, the system predicts future inventory requirements, helping avoid understocking or overstocking.

* + 1. **Data Visualization**

Reports and visual insights, such as demand trends and critical stock levels, are displayed using Matplotlib or Plotly for better decision-making.

* + 1. **Critical Stock Monitoring**

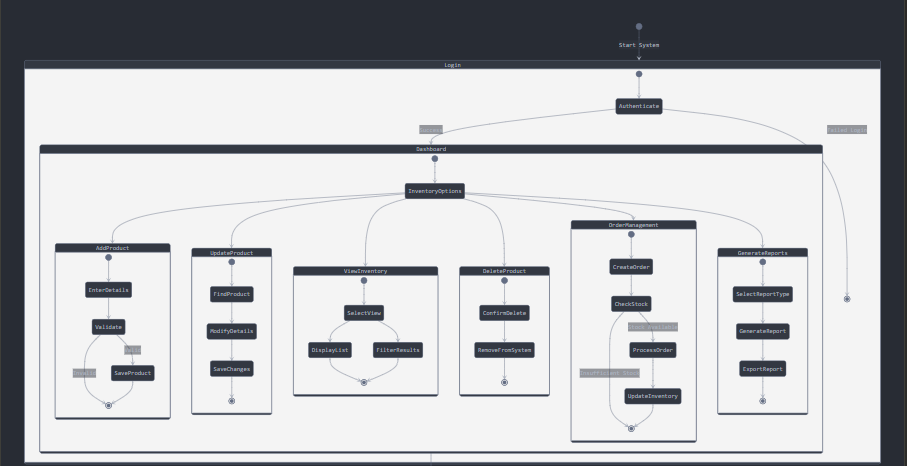
The system flags items reaching low stock levels, ensuring timely restocking.

* + 1. **CSV/Excel Integration**

Users can import/export data in formats like CSV or Excel for offline access and manipulation.

* 1. **Unified Modelling Language (UML)**

It typically includes multiple types of diagrams to capture different aspects of the system. A **Use Case Diagram** would outline the system's functionalities from the user's perspective, showing how different users (such as admins, inventory managers, and regular users) interact with various components like inventory management, item updates, and reporting features. **Class Diagrams** represent the system's main objects and their relationships, such as Inventory, Item, and User classes, detailing the attributes and methods for each entity. **Sequence Diagrams** would illustrate the flow of operations, such as how a user updates stock, or how demand prediction data is processed and visualized**. Activity Diagrams** might be used to show the workflow of tasks such as adding, removing, or updating inventory items. Together, these UML diagrams provide a comprehensive understanding of the system’s structure, behavior, and interactions, making it easier for developers to implement, maintain, and extend the project.

****

**Figure 4.2 UML Diagram of Inventory Management System**

* + 1. **System Initialization**

Starts with user authentication.

Requires successful login to access dashboard

**4.2.2 Dashboard Modules**

Inventory Management

Add new products

Update existing products

View inventory

Remove products

Each action follows a detailed workflow with validation

**4.2.3 Product Addition Workflow**

Enter product details

Validate information

Save to database

Update inventory count

## 4.2.4 Product Update Workflow

## Search for product

## Modify details

## Validate changes

## Update database

## Record change history

## 4.2.5 Order Management

## Create new order

## Check inventory availability

## Process order if stock is sufficient

## Update inventory levels

## Generate invoice

## Record transaction

* 1. **Data Flow**

In the Inventory Management System (IMS), users place orders containing products, which are managed in the Inventory. The Inventory class tracks stock levels, updating quantities as orders are placed. Suppliers provide products, which are added to the inventory. When an order is placed, the system updates the stock and calculates the total amount. Users can add, update, or delete products in the inventory. The system generates reports based on order and inventory data, such as sales and stock reports. Users log in and manage their profiles, interacting with products and orders. The flow between these components ensures efficient inventory tracking, order processing, and reporting within the IMS.

**A diagram of a diagram

Description automatically generated with medium confidence**

**Figure 4.3 Class Diagram of Inventory Management System**

**4.3.1 Supplier**

The Supplier class holds information about suppliers, including their supplierID, name, and contactDetails. The data flow shows that a Supplier "supplies" products to the Inventory, indicating that the Supplier provides products that will be added to the inventory system.

**4.3.2 Inventory**

The Inventory class manages the stock of products. It contains attributes like inventoryID, productID, and quantity. The inventory is linked to Product (which lists the details of items) and handles stock management (add, reduce, and check stock).

**4.3.3 Product**

The Product class contains details like productID, name, category, price, and quantity. The data flow indicates that products are "managed" by the Inventory, and users can add, update, or delete product entries.

**4.3.4 Order**

The Order class holds details about customer orders, including orderID, date, status, and totalAmount. The data flow indicates that an Order "contains" products, and actions like placing an order and updating its status are tracked. The Order class interacts with Inventory by updating stock quantities when products are ordered.

**4.3.5 User**

The User class represents the users (such as admins or employees) interacting with the system. It contains personal details (name, email, password, role) and has methods for logging in, logging out, and updating profiles. The data flow shows that User interacts with products and orders, placing orders and managing products in the system.

**4.3.6 Report**

The Report class generates different types of reports like sales and inventory reports.

1. **TESTING AND RESULTS**
   1. **Objectives of Testing**

**5.1.1 Functionality Testing**

Verify that all features (e.g., adding/removing suppliers, managing inventory, placing orders, generating reports) work as intended. Ensure that methods like addSupplier(), createOrder(), etc., behave correctly under various conditions.

**5.1.2 Data Integrity**

Confirm that inventory stock updates accurately when orders are placed, or products are added/removed. Validate that data between classes (e.g., Product and Order) is consistent.

**5.1.3 Usability Testing**

Ensure the UI is user-friendly, intuitive, and displays accurate information.

**5.1.4 Error Handling**

Test edge cases (e.g., invalid inputs, order placement for out-of-stock products). Verify that proper error messages and exceptions are triggered.

**5.1.5 Performance Testing**

Check how the system performs under high data loads or concurrent user interactions.

**5.1.6 Security Testing**

Validate user authentication and role-based access to prevent unauthorized actions.

**5.1.7 Integration Testing**

Ensure seamless interaction between components (e.g., User, Inventory, and Report generation).

**5.2 Testing Methodologies**

**5.2.1 Unit Testing**

Objective Test individual components or methods (e.g., addSupplier(), createOrder()) for correctness. Tools Use Python's unittest or pytest. Example Test if the addStock() function correctly updates the inventory.

**5.2.2 Integration Testing**

Objective Test the interaction between different modules (e.g., Inventory and Product classes). Example Verify that an order reduces the stock in the inventory and updates the total order amount.

**5.2.3 Functional Testing**

Objective Test whether the system meets functional requirements. Example Verify that users can log in, add products, place orders, and generate reports as expected.

**5.2.4 System Testing**

Objective Test the system, including all modules working together. Example Simulate end-to-end scenarios like a user placing an order and generating a report to ensure proper data flow.

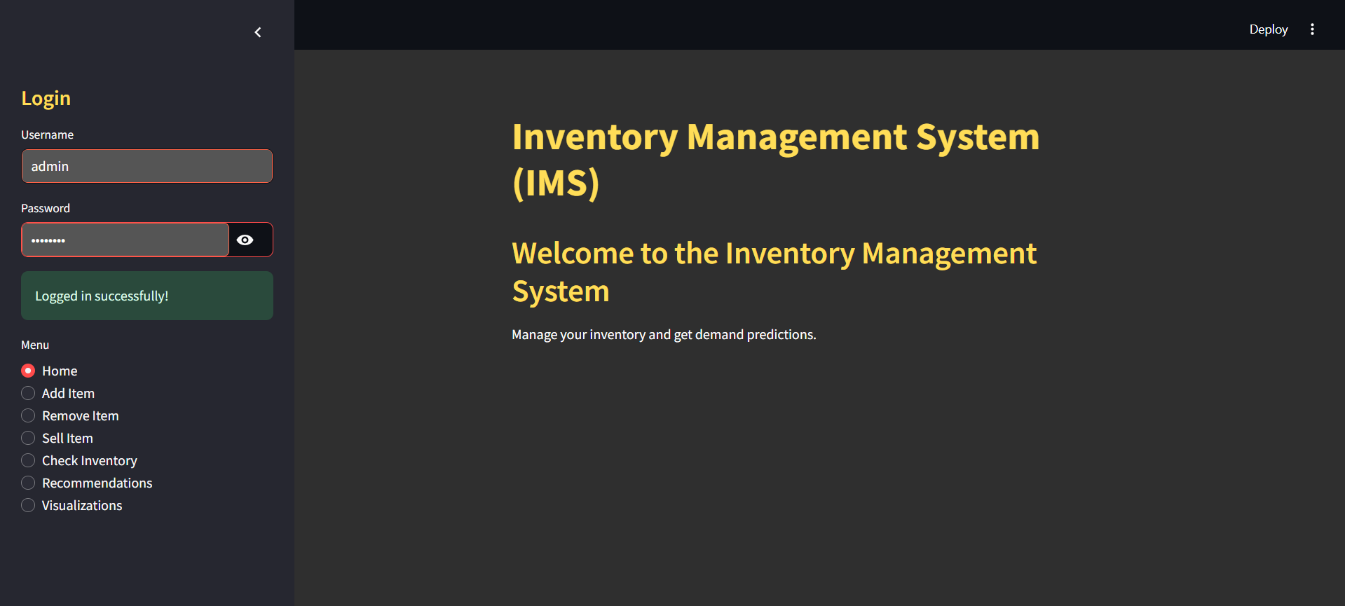
**5.2.5 Performance Testing**

Objective Test system responsiveness and stability under varying loads. Example Check how the IMS handles 100 simultaneous users updating inventory.

**5.2.6** **Usability Testing**

Objective Evaluate the user interface (Streamlit) for intuitiveness and ease of use. Example Confirm that the dashboard displays inventory and orders clearly.

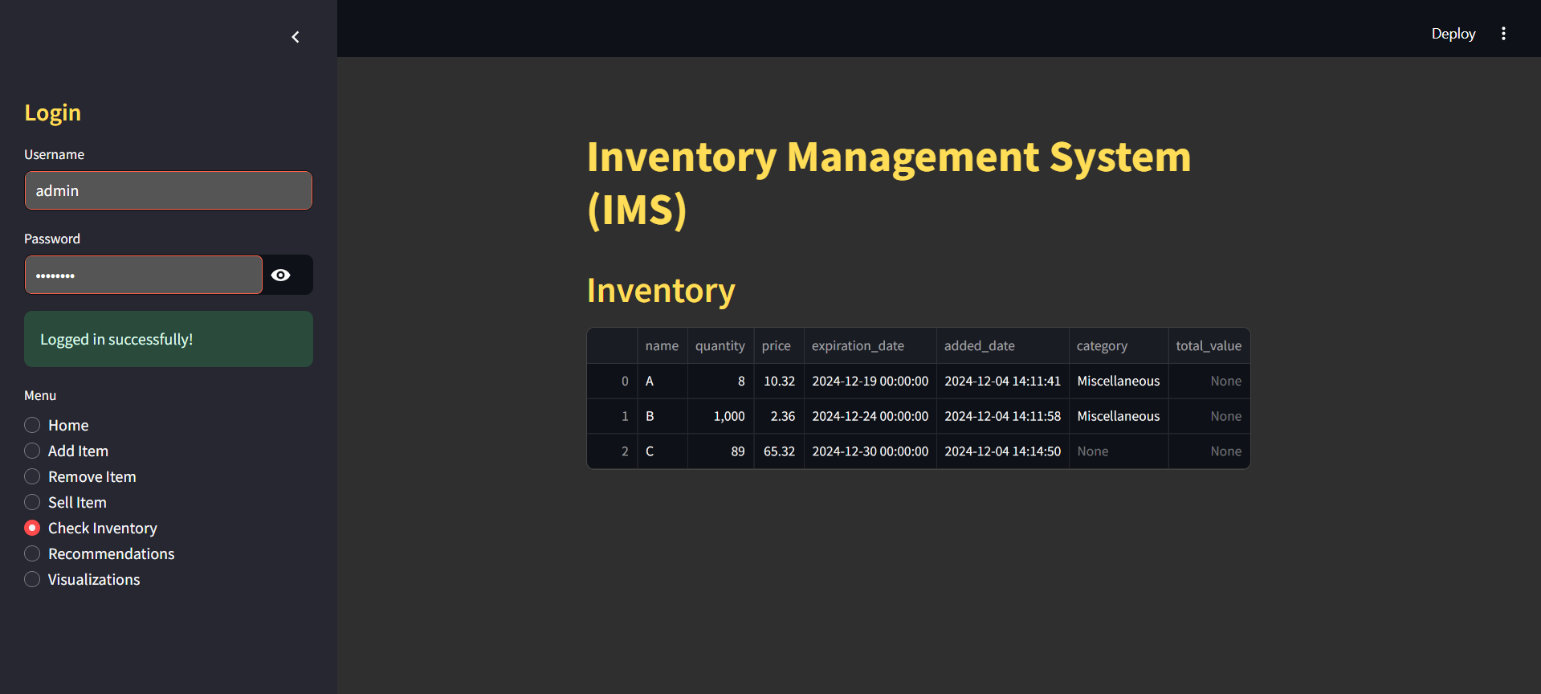
* 1. **Expected Outcome**
     1. **Home Page**



**Figure 5.3.1 Home Page**

The home page provides a dashboard view with navigation options to manage inventory efficiently, including stock addition, removal, sales, and visualizations for a seamless user experience.

* + 1. **Inventory Data**



**Figure 5.3.2 Inventory Data**

Displays a comprehensive table of current inventory data, including item details, quantities, and statuses, ensuring users have a clear overview of all inventory records.

* + 1. **Addition of Inventory Stock**

A screenshot of a computer

Description automatically generated

**Figure 5.3.3 Addition of Stock**

A feature for adding new inventory items or updating existing stock quantities, ensuring accurate record-keeping and streamlined inventory growth.

* + 1. **Selling of Inventory Stock**

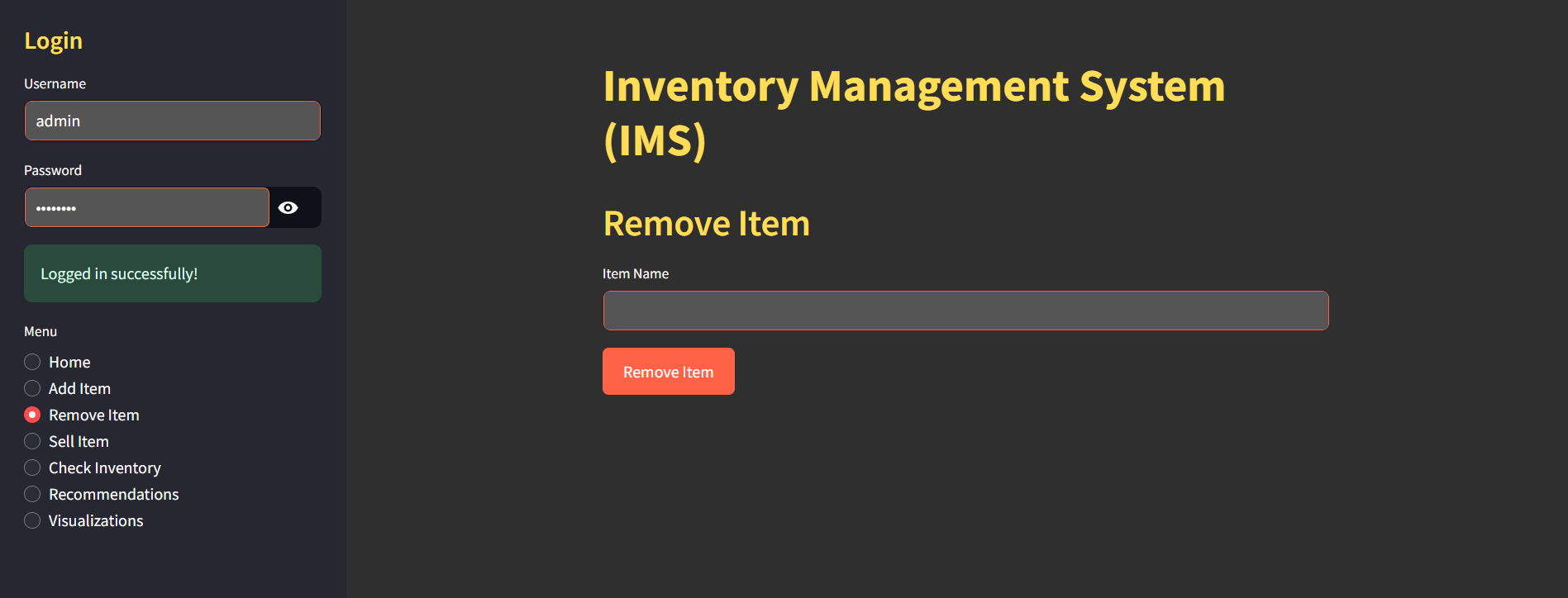
A screenshot of a computer

Description automatically generated

**Figure 5.3.4 Selling of Stock**

Captures the sales process by deducting stock quantities and updating inventory records automatically, providing real-time sales tracking.

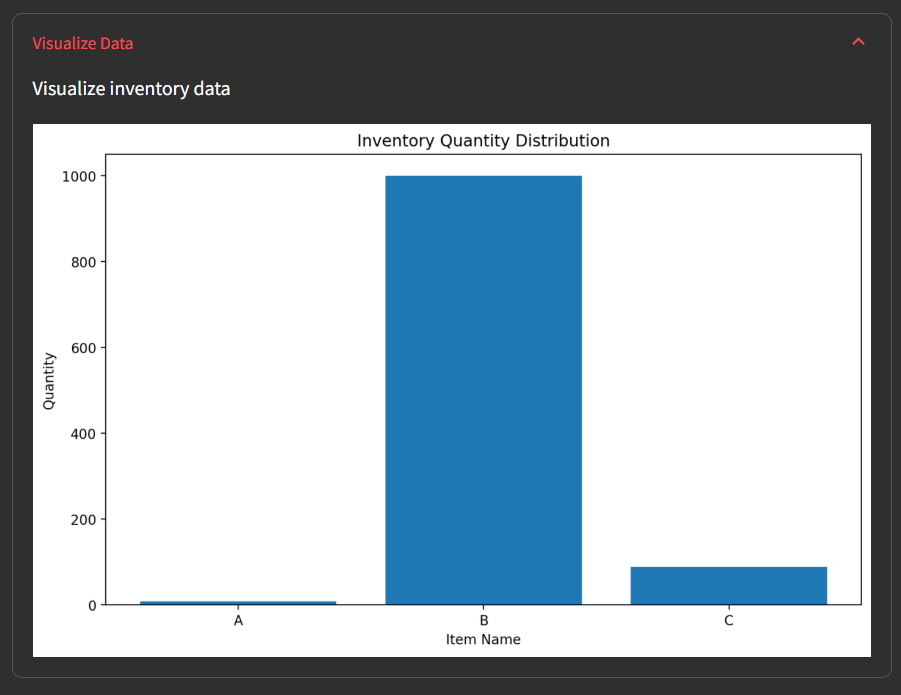
* + 1. **Removing of Inventory Stock**



**Figure 5.3.5 Removing of Stock**

Enables the removal of outdated, damaged, or unwanted stock from the inventory, maintaining cleanliness and relevance in stock data.

* + 1. **Inventory Stock Visualization**



**Figure 5.3.6 Stock Visualization**

Displays visual charts and graphs for inventory insights, such as stock trends and distribution, aiding in data-driven decision-making.

* + 1. **Critical Stock Alert**

A screenshot of a computer

Description automatically generated

**Figure 5.3.7 Critical Stock Alert**

Provides automated notifications or alerts for critical stock levels, ensuring timely restocking and preventing stockouts.

1. **CONCLUSION AND FUTURE SCOPE**
   1. **Conclusion**

The Inventory Management System (IMS) project is a robust and efficient solution designed to simplify and optimize inventory operations for businesses. Through its well-structured architecture, the system seamlessly integrates key functionalities such as supplier management, inventory tracking, order processing, and report generation. Built using the Streamlit framework, the IMS offers an intuitive and user-friendly interface, making it accessible for users with varying technical expertise.

The system ensures that businesses can effectively manage their inventory with real-time updates, accurate stock tracking, and streamlined order management. By automating critical operations like updating stock levels, generating sales and inventory reports, and managing supplier and product data, the IMS reduces manual effort, minimizes errors, and enhances overall productivity. Its role-based access control and secure login mechanisms further ensure that only authorized users can interact with the system, safeguarding sensitive business data.

The project also demonstrates the practical application of key software engineering principles, such as modularity, scalability, and maintainability. Each component of the system—from supplier and product management to reporting—has been carefully designed and tested to ensure seamless interaction and reliability. Additionally, features like error handling and input validation ensure that the system performs well even under edge cases, providing a robust solution for real-world scenarios.

In conclusion, the IMS serves as a comprehensive tool that addresses the challenges businesses face in managing their inventory. It not only streamlines inventory operations but also provides valuable insights through its reporting functionality, enabling informed decision-making. This project highlights the potential of software solutions to improve operational efficiency and sets the foundation for future enhancements, such as integrating predictive analytics or mobile accessibility.

* 1. **Future Work**
     1. **Predictive Analytics Integration** Incorporate machine learning algorithms to forecast inventory demand based on historical data and market trends.
     2. **Mobile App Development** Develop a mobile-friendly version of the IMS to enable users to manage inventory and monitor reports on the go.
     3. **Real-time Notifications** Implement push notifications or email alerts for critical events such as low stock levels, order status updates, or supplier delays.
     4. **Advanced Reporting and Dashboards** Enhance the reporting module with interactive dashboards, visual analytics, and custom report generation capabilities.
     5. **Barcode and QR Code Integration** Add functionality to scan barcodes or QR codes for faster inventory updates, product identification, and order processing.
     6. **Integration with E-commerce Platforms** Enable seamless integration with popular e-commerce platforms to automate order synchronization and inventory updates.
     7. **Multi-Warehouse Support** Extend the system to support and manage inventory across multiple warehouses or storage locations.
     8. **Role-based Access Enhancements** Introduce finer-grained role-based permissions to define specific access levels for various user roles.
     9. **Cloud Integration** Migrate to a cloud-based infrastructure to provide scalability, remote accessibility, and real-time collaboration.
     10. **Automated Supplier Management** Enable automatic supplier order generation when stock falls below predefined thresholds.
     11. **IoT Integration** Use IoT devices to track inventory in real time, ensuring precise monitoring of stock levels and conditions.
     12. **Data Backup and Recovery** Implement automated data backup and recovery systems to safeguard against data loss.

**BIBLIOGRAPHY**

1. **Kaur, H., & Singh, D. (2021)**

**Inventory Management System Using Python and SQL. *International Journal of Computer Science and Information Security (IJCSIS)*, *19*(04), 987–992.**

This paper presents a Python and SQL-based inventory management system designed to streamline inventory tracking and control processes. It emphasizes the system’s ability to handle real-time inventory updates and reduce manual efforts, making it suitable for small-to-medium-sized businesses.

1. **Sharma, P., Gupta, R., & Roy, A. (2020)**

**Streamlit-based Inventory Management Application for Small Businesses. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, *10*(02), 45–50.**

The authors developed an Inventory Management System using Streamlit, focusing on ease of deployment and user-friendly interfaces for small businesses. The application simplifies stock management, generates analytics reports, and optimizes stock replenishment processes using minimal resources.

1. **Das, S., & Bose, R. (2019)**

**Optimization of Inventory Tracking with Machine Learning Integration. *Journal of Engineering and Technology Research*, *11*(3), 210–216.**

This study explores integrating machine learning models into inventory tracking systems. It highlights predictive analytics for demand forecasting, inventory optimization, and reduction of wastage. The system focuses on adaptive learning for continuous improvement in stock level management and cost efficiency.

1. **Kumar, A., & Verma, S. (2022)**

**Enhancing Efficiency in Inventory Systems Using Flask Framework. *International Journal of Advanced Research in Computer Science (IJARCS)*, *13*(01), 35–40.**

This research focuses on a Flask-based inventory management framework tailored for e-commerce platforms. The system provides seamless tracking, inventory categorization, and real-time updates. The paper discusses the integration of APIs and how automation improves workflow efficiency and inventory accuracy.

1. **Patel, N., Shah, M., & Desai, H. (2021)**

**Real-Time Inventory Management with Streamlit and SQLite. *Journal of Emerging Trends in Computing and Information Sciences*, *12*(06), 437–443.**

The authors demonstrate an Inventory Management System using Streamlit and SQLite, emphasizing real-time data processing and storage efficiency. The paper highlights the simplicity of implementation and how it aids small businesses in reducing overheads while maintaining accurate inventory records.

**APPENDIX**

**import** pandas **as** pd

**import** numpy **as** np

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn.preprocessing **import** StandardScaler

**import** streamlit **as** st

**import** datetime

**import** os

**import** matplotlib.pyplot **as** plt

**class** DemandPredictor

**def** \_\_init\_\_(self, inventory\_data)

        self.inventory\_data **=** inventory\_data

        self.model **=** **None**

        self.scaler **=** StandardScaler()

**def** prepare\_data(self)

        # Feature engineering

        df **=** self.inventory\_data.copy()

        df['days\_since\_added'] **=** (pd.Timestamp.now() **-** df['added\_date']).dt.days

        # Select relevant features

        features **=** ['quantity', 'price', 'days\_since\_added']

        target **=** 'quantity'  # Predicting future quantity

        X **=** df[features]

        y **=** df[target]

**return** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)

**def** train\_model(self)

        X\_train, X\_test, y\_train, y\_test **=** self.prepare\_data()

        # Scale features

        X\_train\_scaled **=** self.scaler.fit\_transform(X\_train)

        X\_test\_scaled **=** self.scaler.transform(X\_test)

        # Train Random Forest Regressor

        self.model **=** RandomForestRegressor(n\_estimators**=**100, random\_state**=**42)

        self.model.fit(X\_train\_scaled, y\_train)

        # Calculate and display model performance

        train\_score **=** self.model.score(X\_train\_scaled, y\_train)

        test\_score **=** self.model.score(X\_test\_scaled, y\_test)

        st.write(**f**"Model Training Score {train\_score**.2f**}")

        st.write(**f**"Model Testing Score {test\_score**.2f**}")

**def** predict\_demand(self, item\_features)

**if** self.model **is** **None**

            st.warning("Model not trained. Please train the model first.")

**return** **None**

        # Scale input features

        scaled\_features **=** self.scaler.transform([item\_features])

        # Predict demand

        predicted\_demand **=** self.model.predict(scaled\_features)[0]

**return** max(0, round(predicted\_demand, 2))

**def** recommend\_restock(self)

        recommendations **=** []

**for** \_, item **in** self.inventory\_data.iterrows()

            features **=** [

                item['quantity'],

                item['price'],

                (pd.Timestamp.now() **-** item['added\_date']).days

            ]

            predicted\_demand **=** self.predict\_demand(features)

**if** predicted\_demand **and** predicted\_demand **<** CRITICAL\_STOCK\_THRESHOLD

                recommendations.append({

                    'name' item['name'],

                    'current\_quantity' item['quantity'],

                    'predicted\_demand' predicted\_demand,

                    'recommended\_restock' max(CRITICAL\_STOCK\_THRESHOLD **-** item['quantity'], 0)

                })

**return** pd.DataFrame(recommendations)

# Path to the CSV file for saving/loading inventory data

INVENTORY\_FILE **=** "inventory\_data.csv"

CRITICAL\_STOCK\_THRESHOLD **=** 10  # Set critical stock threshold

**class** Inventory

**def** \_\_init\_\_(self)

        self.inventory\_df **=** self.load\_inventory()

**def** load\_inventory(self)

        """Load inventory data from CSV file if it exists."""

**if** os.path.exists(INVENTORY\_FILE)

            inventory\_df **=** pd.read\_csv(INVENTORY\_FILE)

            inventory\_df['expiration\_date'] **=** pd.to\_datetime(inventory\_df['expiration\_date'], errors**=**'coerce')

            inventory\_df['added\_date'] **=** pd.to\_datetime(inventory\_df['added\_date'], errors**=**'coerce')

**return** inventory\_df

**return** pd.DataFrame(columns**=**["name", "quantity", "price", "expiration\_date", "added\_date"])

**def** save\_inventory(self)

        """Save inventory data to CSV file."""

        self.inventory\_df.to\_csv(INVENTORY\_FILE, index**=False**)

        st.success("Inventory saved successfully!")

**def** add\_item(self, name, quantity, price, expiration\_date)

        """Add or update an item in the inventory."""

        # Ensure the expiration date has time set to 000000

**if** isinstance(expiration\_date, datetime.date)

            expiration\_date **=** datetime.datetime.combine(expiration\_date, datetime.time(0,0))  # type ignore # Combine date with 000000 time

**if** name **in** self.inventory\_df["name"].values

            self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "quantity"] **+=** quantity

            self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "price"] **=** price

**else**

            new\_item **=** pd.DataFrame({

                "name" [name],

                "quantity" [quantity],

                "price" [price],

                "expiration\_date" [expiration\_date],

                "added\_date" [datetime.datetime.now()]

            })

            self.inventory\_df **=** pd.concat([self.inventory\_df, new\_item], ignore\_index**=True**)

        self.save\_inventory()

**def** remove\_item(self, name)

        """Remove an item from the inventory."""

        self.inventory\_df **=** self.inventory\_df[self.inventory\_df["name"] **!=** name]

        self.save\_inventory()

**def** reduce\_quantity(self, name, quantity\_to\_reduce)

        """Reduce the quantity of an item in the inventory."""

**if** name **in** self.inventory\_df["name"].values

            current\_quantity **=** self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "quantity"].values[0] # type ignore

            new\_quantity **=** max(0, current\_quantity **-** quantity\_to\_reduce)

            self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "quantity"] **=** new\_quantity

        self.save\_inventory()

**def** get\_inventory(self)

        """Return the current inventory DataFrame."""

**return** self.inventory\_df

**def** check\_critical\_stock(self)

        """Check if any item has a quantity below the critical threshold."""

        critical\_items **=** self.inventory\_df[self.inventory\_df["quantity"] **<** CRITICAL\_STOCK\_THRESHOLD]

**return** critical\_items

**def** calculate\_days\_to\_expire(self, expiration\_date)

        """Calculate the days remaining until the expiration date."""

**if** pd.isna(expiration\_date)

**return** **None**

**return** (expiration\_date **-** datetime.datetime.now()).days

**def** sell\_item(self, name, quantity\_to\_sell)

        """Sell a certain quantity of an item from the inventory."""

**if** name **in** self.inventory\_df["name"].values

            current\_quantity **=** self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "quantity"].values[0] # type ignore

            new\_quantity **=** max(0, current\_quantity **-** quantity\_to\_sell)

            self.inventory\_df.loc[self.inventory\_df["name"] **==** name, "quantity"] **=** new\_quantity

        self.save\_inventory()

**class** User

**def** \_\_init\_\_(self, username, password)

        self.username **=** username

        self.password **=** password

**def** authenticate(self, entered\_username, entered\_password)

        """Authenticate user credentials."""

**return** self.username **==** entered\_username **and** self.password **==** entered\_password

# Custom CSS for enhanced styling

# Custom CSS for dark mode with vibrant and high-contrast colors

**def** set\_custom\_style()

    st.markdown("""

    <style>

    /\* Global Styles \*/

    .stApp {

        background-color #2f2f2f;  /\* Dark background \*/

        color #ffffff;  /\* Light text color \*/

        font-family 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

    }

    /\* Login Container \*/

    .login-container {

        max-width 400px;

        margin auto;

        padding 30px;

        background-color #333333;  /\* Darker background for the login box \*/

        border-radius 12px;

        box-shadow 0 4px 6px rgba(0, 0, 0, 0.3);

        color #ffffff;  /\* Light text color \*/

    }

    /\* Buttons \*/

    .stButton > button {

        background-color #FF6347;  /\* Vibrant red \*/

        color white;

        border none;

        border-radius 6px;

        padding 10px 20px;

        transition all 0.3s ease;

    }

    .stButton > buttonhover {

        background-color #FF4500;  /\* Darker red on hover \*/

        transform scale(1.05);

    }

    /\* Sidebar \*/

    .css-1aumxhk {

        background-color #444444;  /\* Dark sidebar \*/

        color #ffffff;  /\* Light text in sidebar \*/

    }

    /\* Headers \*/

    h1, h2, h3 {

        color #FFDD57;  /\* Vibrant yellow for headers \*/

    }

    /\* Input Fields \*/

    .stTextInput > div > div > input {

        border-radius 6px;

        border 1px solid #FF6347;  /\* Red border for input fields \*/

        padding 10px;

        color white;

        background-color #555555;  /\* Dark background for input fields \*/

    }

    /\* Warning and Success Messages \*/

    .stAlert {

        border-radius 8px;

    }

    /\* Success Message Styling \*/

    .stSuccess {

        background-color #32CD32;  /\* Vibrant green \*/

        color white;

    }

    /\* Error Message Styling \*/

    .stError {

        background-color #FF4500;  /\* Vibrant red \*/

        color white;

    }

    </style>

    """, unsafe\_allow\_html**=True**)

# Set custom styles

set\_custom\_style()

# Initialize the inventory and user system

inventory **=** Inventory()

demand\_predictor **=** DemandPredictor(inventory.get\_inventory())

# Login credentials

username **=** "admin"

password **=** "admin123"

# Streamlit app layout

st.title("Inventory Management System (IMS)")

# User login screen

st.sidebar.title("Login")

entered\_username **=** st.sidebar.text\_input("Username")

entered\_password **=** st.sidebar.text\_input("Password", type**=**"password")

# Authenticate user

user **=** User(username, password)

**if** user.authenticate(entered\_username, entered\_password)

    st.sidebar.success("Logged in successfully!")

    # Display inventory management options

    menu **=** st.sidebar.radio("Menu", ["Home", "Add Item", "Remove Item", "Sell Item", "Check Inventory", "Recommendations", "Visualizations"])

**if** menu **==** "Home"

        st.header("Welcome to the Inventory Management System")

        st.write("Manage your inventory and get demand predictions.")

**elif** menu **==** "Add Item"

        st.header("Add Item")

        name **=** st.text\_input("Item Name")

        quantity **=** st.number\_input("Quantity", min\_value**=**0)

        price **=** st.number\_input("Price", min\_value**=**0.0, format**=**"%.2f")

        expiration\_date **=** st.date\_input("Expiration Date")

**if** st.button("Add Item")

            inventory.add\_item(name, quantity, price, expiration\_date)

            st.success("Item added successfully!")

**elif** menu **==** "Remove Item"

        st.header("Remove Item")

        name **=** st.text\_input("Item Name")

**if** st.button("Remove Item")

            inventory.remove\_item(name)

            st.success(**f**"Item '{name}' removed successfully!")

**elif** menu **==** "Sell Item"

        st.header("Sell Item")

        name **=** st.text\_input("Item Name")

        quantity\_to\_sell **=** st.number\_input("Quantity to Sell", min\_value**=**1)

**if** st.button("Sell Item")

            inventory.sell\_item(name, quantity\_to\_sell)

            st.success(**f**"{quantity\_to\_sell} units of '{name}' sold successfully!")

**elif** menu **==** "Check Inventory"

        st.header("Inventory")

        inventory\_df **=** inventory.get\_inventory()

        st.write(inventory\_df)

**elif** menu **==** "Recommendations"

        st.header("Restock Recommendations")

        recommendations **=** demand\_predictor.recommend\_restock()

        st.write(recommendations)

**elif** menu **==** "Visualizations"

        st.header("Inventory Visualizations")

        chart\_type **=** st.selectbox("Choose Chart Type", ["Bar Chart", "Pie Chart", "Line Chart"])

        inventory\_df **=** inventory.get\_inventory()

**if** chart\_type **==** "Bar Chart"

            fig, ax **=** plt.subplots(figsize**=**(10, 6))

            ax.bar(inventory\_df["name"], inventory\_df["quantity"], color**=**'skyblue')

            ax.set\_title("Inventory Quantities by Item", fontsize**=**16)

            ax.set\_xlabel("Item Name", fontsize**=**14)

            ax.set\_ylabel("Quantity", fontsize**=**14)

            ax.tick\_params(axis**=**'x', rotation**=**45)

            st.pyplot(fig)

**elif** chart\_type **==** "Pie Chart"

            fig, ax **=** plt.subplots(figsize**=**(8, 8))

            inventory\_df.set\_index("name")["quantity"].plot.pie(autopct**=**"%1.1f%%", ax**=**ax)

            ax.set\_ylabel("")

            ax.set\_title("Inventory Distribution", fontsize**=**16)

            st.pyplot(fig)

**elif** chart\_type **==** "Line Chart"

            fig, ax **=** plt.subplots(figsize**=**(10, 6))

            ax.plot(inventory\_df["name"], inventory\_df["quantity"], marker**=**'o', color**=**'purple')

            ax.set\_title("Inventory Quantities Over Time", fontsize**=**16)

            ax.set\_xlabel("Item Name", fontsize**=**14)

            ax.set\_ylabel("Quantity", fontsize**=**14)

            ax.tick\_params(axis**=**'x', rotation**=**45)

            st.pyplot(fig)

**else**

    st.sidebar.error("Invalid username or password.")