RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

INVENTORY MANAGEMENT SYSTEM

MINI PROJECT REPORT

Submitted by

SHIVA GANESH S	**231801164**
SASI SRIRAM E	**231801159**
VISHAL GANESAN	**231801188**

CS23332 DATABASE MANAGEMENT SYSTEM

Department of Artificial Intelligence and Data Science

Rajalakshmi Engineering College, Thandalam

2024-2025

BONAFIDE CERTIFICATE

Certified that this project report "**LIBRARY MANAGEMENT SYSTEM**" is the bonafide work of **"SHIVA GANESH S (231801164), SASI SRIRAM E (231801159), VISHAL GANESAN (231801188)"**

who carried out the project work under my supervision.

Submitted for the Practical Examination held on ________________________

SIGNATURE	**SIGNATURE**
Dr. GNANASEKAR J M	**Dr. MANORANJINI J**
Head of the Department, Artificial intelligence and data Science, Rajalakshmi Engineering College (Autonomous),Chennai-602105	**Assoc.Professor, Artificial Intelligence and Data Science, Rajalakshmi Engineering College (Autonomous), Thandalam, Chennai-602105**
INTERNAL EXAMINER	**EXTERNAL EXAMINER**

TABLE OF CONTENT

TABLE OF FIGURES

ABSTRACT

The "Inventory Management System" streamlines stock tracking and management to prevent overstocking and stockouts, enhancing profitability and customer satisfaction. Built with a robust DBMS, it offers real-time insights into inventory levels, supplier details, and reorder requirements. Key features include inventory tracking, order processing, supplier management, and reporting, ensuring secure, efficient data storage and retrieval. With multi-user access, transaction management, and real-time updates, the system supports simultaneous users without compromising data accuracy. By automating manual processes, businesses can forecast demand, optimize stock levels, and make data-driven decisions. Scalable and customizable, it adapts to various business sizes and industry needs.

1.INTRODUCTION:

1.1 General:

Effective inventory management is critical for businesses seeking to minimize costs while meeting customer demand efficiently. Inventory mismanagement, such as overstocking or understocking, can lead to financial losses, decreased customer satisfaction, and interrupted workflows. This project, titled "Inventory Management System," focuses on the development of a streamlined, database-driven solution that automates the tracking and management of inventory. By leveraging the capabilities of a Database Management System (DBMS), this inventory system provides real-time data access, minimizes human errors, and offers robust analytical insights for smarter decision-making. Whether it's managing products, monitoring supplier relationships, or tracking order histories, the system is designed to meet the complex demands of inventory management in modern business environments. Through this system, companies can optimize inventory levels, reduce manual workload, and enhance efficiency, which in turn leads to a more agile and customer-responsive operation.

1.2 Objectives

Efficient Stock Management

Enable real-time tracking of stock levels, providing alerts for reorder points to prevent overstocking and stockouts.

Automation of Inventory Processes

Minimize manual inventory tasks by automating order placements, stock updates, and report generation.

Enhanced Security and Access Control

Protect sensitive inventory data by implementing user-based access controls, ensuring that only authorized personnel can modify records.

Cost Optimization

Help businesses reduce storage and inventory holding costs by optimizing stock levels based on demand forecasts.

User-Friendly Interface

Provide an intuitive interface for ease of use, allowing non-technical users to efficiently manage inventory.

1.3 Scope:

The scope of the Inventory Management System (IMS) focuses on streamlining inventory operations through efficient tracking, data management, and process automation. Key points include:

- Core Functionalities: Real-time inventory monitoring, CRUD operations, supplier and transaction management, and low-stock alerts.
- Database Reliability: Relational DBMS principles like normalization, data integrity, and foreign key relationships ensure accurate and consistent data storage.

- User Interface: Flask framework provides a user-friendly platform for managing inventory with secure and efficient interactions.
- Customizability: Adaptable to various industries and business sizes, with options for integration with external systems.
- Future Enhancements: Includes analytics, automated reporting, advanced demand forecasting, and mobile accessibility to expand utility and user convenience

2. SYSTEM OVERVIEW:

2.1 SYSTEM ARCHITECTURE:

The Inventory Management System (IMS) architecture is designed to integrate a robust database with a user-friendly web application for seamless inventory management. The architecture consists of three main layers: Presentation Layer, Application Layer, and Data Layer. These layers interact efficiently to ensure data integrity, user accessibility, and real-time operations.

1. Presentation Layer (Front-End)

This layer handles user interactions and provides a responsive interface for inventory management tasks.

- Components: HTML, CSS, JavaScript (for responsive design).
- Key Features:
  - Forms for CRUD operations (add, update, delete, and view inventory).
  - Dashboards for real-time stock tracking and analytics.
  - User authentication and role-based access

2. Application Layer (Back-End)

This is the core of the system, where the business logic resides. It acts as a bridge between the user interface and the database.

- Framework: Flask (Python).
- Key Features:
  - Implements CRUD operations through API endpoints.
  - Real-time notifications for low-stock levels.
  - Integration of supplier and transaction management modules.
  - Use of ORM (SQLAlchemy) for secure and efficient database interaction.

3. Data Layer (Database)

This layer ensures data reliability and integrity through a relational database.

- Database: SQL (e.g., MySQL, PostgreSQL).
- Key Features:
  - Tables for inventory items, categories, suppliers, and transactions.
  - Normalization to reduce redundancy and improve efficiency.
  - Use of primary and foreign keys for relational consistency.
  - Backup mechanisms for data recovery.

This architecture ensures that the IMS is scalable, efficient, and secure, making it adaptable to businesses of any size.

2.2 MODULES:

The Inventory Management System (IMS) is designed with modular components to ensure seamless functionality, scalability, and efficient inventory operations. Below is an overview of the core modules:

1. User Management:
   This module handles user authentication and role-based access control. It ensures secure login for administrators, inventory managers, and other users, defining permissions based on roles.
2. Inventory Management:
   Tracks stock levels of products and categories. It includes CRUD (Create, Read, Update, Delete) operations to manage items, monitor stock availability, set reorder thresholds, and generate low-stock alerts.
3. Supplier Management:
   Stores and manages supplier details, including contact information, product catalogs, and transaction history. Facilitates communication for reorder processes.
4. Order Processing:
   Manages purchase and sales orders. Tracks order statuses, processes incoming and outgoing stock, and updates inventory levels in real-time.
5. Reporting and Analytics:
   Generates detailed reports on stock movement, supplier performance, and order history. Offers data visualizations for insights into inventory trends and demand forecasting.
6. Dashboard:
   Provides a centralized view of key metrics, including stock levels, pending orders, and performance summaries, enabling quick decision-making.

These modules collectively create a robust, user-friendly system adaptable to various business needs, ensuring operational efficiency and data-driven decision-making.

2.3 USER ROLES AND ACCESS LEVELS:

The IMS implements role-based access control to ensure secure and efficient system usage. Each user role is assigned specific access rights based on responsibilities, promoting accountability and preventing unauthorized actions. Below are the primary user roles and their access levels:

1. **Administrator**:
   - **Access Level**: Full access.
   - **Responsibilities**:
     - Manage user accounts (add, modify, delete roles).
     - Configure system settings and permissions.
     - View and generate all reports.
     - Oversee inventory, supplier, and order management modules.
2. **Inventory Manager**:
   - **Access Level**: Moderate access.
   - **Responsibilities**:

        - Add, update, or remove inventory items and categories.
        - Monitor stock levels and reorder items.
        - View inventory-related reports and analytics.
3. **Procurement Staff**:
    - **Access Level**: Limited access.
    - **Responsibilities**:
        - Manage supplier details and purchase orders.
        - Update received stock and ensure accurate transaction logging.
        - View supplier performance reports.
4. **Viewer/Employee**:
    - **Access Level**: Read-only access.
    - **Responsibilities**:
        - View stock levels, product details, and reports.
        - Monitor inventory status but cannot perform modifications.

This role-based system ensures data security, operational clarity, and efficient task delegation across the organization.

3.SURVEY OF TECHNOLOGIES:

3.1 Software Description :

This project utilizes a combination of software tools to create acomprehensive and efficient supermarket management system:

- **Database Management System (DBMS):** Sql is chosen asthe DBMS for the library management system due to its flexibilitywith unstructured data, ease of scalability, and ability to handle a diverse range of data types efficiently.
- **Integrated Development Environment (IDE):** PyCham isselected as the IDE for its Python-specific features, code completion, debugging tools, and seamless integration with Sql.

3.2 Languages :

- **SQL:** SQL is used as the database backbone, enabling efficient storage, retrieval, and management of inventory data. SQL's structured query language supports complex queries, ensuring data integrity, quick updates, and reliable performance. It is ideal for handling inventory data in a structured, relational format, allowing for easy management of stock, suppliers, and transaction records.
- **FLASK**: Flask, a lightweight and flexible Python web framework, serves as the backend framework for building the system's web interface. Flask facilitates rapid development by offering simplicity and scalability, Flask's integration with SQL databases enables seamless data flow, making it possible for users to track inventory, manage orders, and generate reports in real-time. Together, SQL and Flask create a powerful, user-friendly, and responsive system for effective inventory management.

3.2.1. SQL:

Sql plays a crucial role in the Inventory management system by:

High Performance

- SQL efficiently manages large datasets, handling complex queries and transactions with minimal response time.

Data Integrity

- SQL ensures accuracy and reliability of data with constraints and relational integrity checks, preventing data duplication and inconsistency.

Scalability

- SQL databases can scale vertically, accommodating increasing amounts of data as a business grows.

3.2.2 PYTHON:

• **Develop backend logic:** Implement core functionalities like bookissue/return, fine calculations, and user account management.

• **Interact with the MongoDB database:** Use PyMongo to connect andmanage MongoDB collections, enabling flexible data handling and retrieval.

• **Create the user interface:** Design a user-friendly interface with Tkinter,allowing users to perform library tasks and view records.

• **Integrate with external systems:** Connect the library system to othereducational or resource management tools for seamless data sharing.

4.REQUIREMENTS AND ANALYSIS:

4.1 Functional Requirements:

Products Management:

- Add, edit, and delete product records in the inventory.
- Track product availability and current stock levels.
- Manage product categories (e.g., supplements, protein, vitamins) and brands.

Member Management:

- Create, update, and view customer profiles.
- Manage customer contact details and membership or loyalty status.

Transaction Management:

- Process orders for customers, including adding items to the cart and finalizing the purchase.
- Update product stock levels after each transaction.

Reporting and Analytics:

- Generate reports on inventory levels, best-selling products, and current stock status.
- Analyze product expiration dates and identify products approaching their expiry for promotion or clearance.

4.2 Non-Functional Requirements:

- Performance: Ensure the system efficiently handles numerous inventory transactions, customer orders, and updates to product stock without lag or delays.
- Scalability: Support the growth of the product catalog, customer base, and transaction volume, allowing the system to scale as the business expands.
- Security: Safeguard sensitive customer data (such as payment information) and inventory records through robust authentication, authorization, and encryption methods.
- Usability: Provide an intuitive and easy-to-navigate interface for both administrators (staff) managing inventory and customers browsing or purchasing products.
- Reliability: Ensure the system is stable with minimal downtime, accurately tracking product availability, and ensuring the integrity of transaction data

4.3 Hardware and Software Requirements :

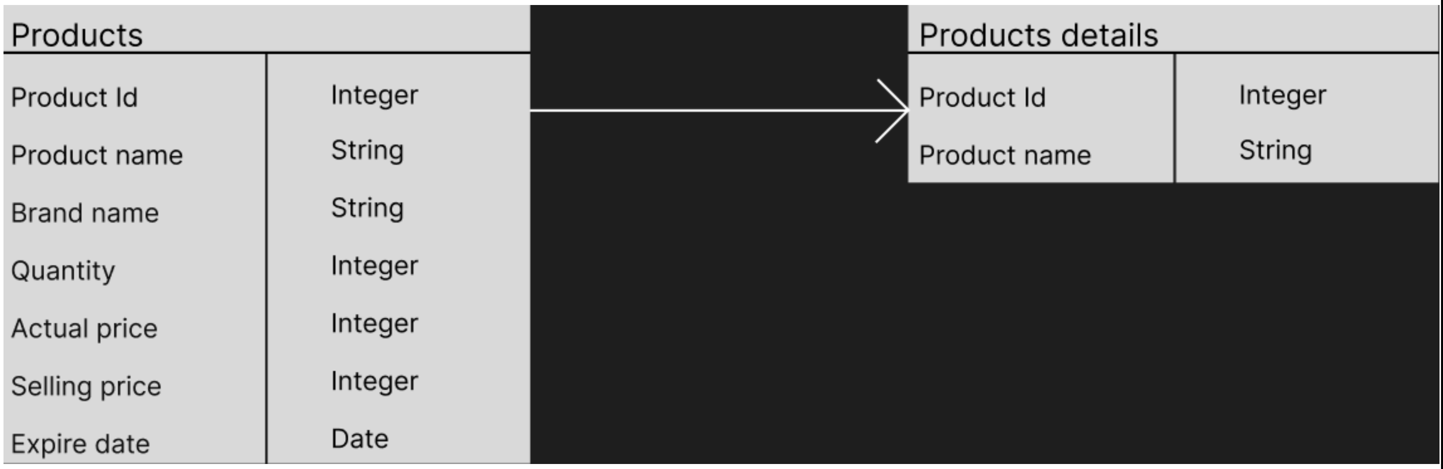
❖ **Hardware :**

- **Server:** A powerful server with sufficient CPU, RAM, and storageto handle the database and application workload.
- **Network:** A reliable network connection to allow access to thesystem from different locations.
- **POS Terminals:** Point-of-sale terminals for processing salestransactions.

❖ **Software :**

- **Database Management System (DBMS):** MySQL, PostgreSQL,or SQL Server or Mongodb.
- **Integrated Development Environment (IDE):** PyCharm orVisual Studio Code.
- **Operating System:** Linux or Windows.

4.4 ER DIAGRAM :



Products	
Product Id	Integer
Product name	String
Brand name	String
Quantity	Integer
Actual price	Integer
Selling price	Integer
Expire date	Date

Products details	
Product Id	Integer
Product name	String

FIG 1.ER DIAGRAM

4.5Architectural Diagram :

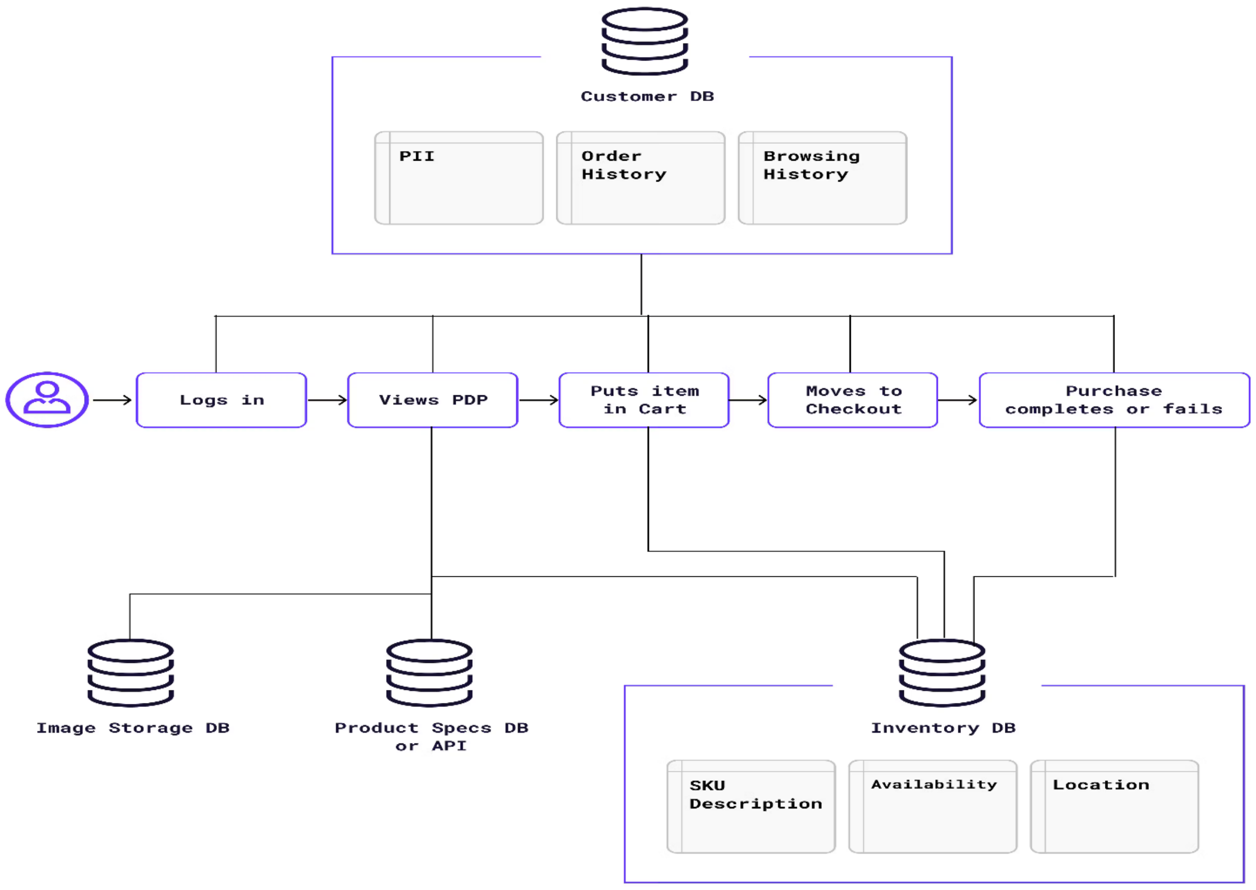


FIG 2.ARCHITECTURAL DIAGRAM

5.SYSTEM DESIGN:

5.1 Database Design and Tables:

The database for the IMS uses a relational model to ensure data consistency and integrity. Core tables include **Products** (product details and stock levels), **Categories** (product grouping), **Suppliers** (supplier information), **Orders** (sales and purchase transactions), and **Users** (role-based access). Relationships are enforced using primary and foreign keys, ensuring normalized data storage and efficient retrieval.

5.2UI Design Overview

The UI is intuitive and user-friendly, built using Flask with Bootstrap for responsive design. Key components include a dashboard displaying stock levels, alerts, and analytics, forms for CRUD operations, and navigation menus for accessing modules like inventory, orders, and reports. Focus is on clarity and ease of use.

5.3 Workflow and Process Diagrams

The process flow covers the user journey, from logging in to add items, placing them, and viewing orders**.**

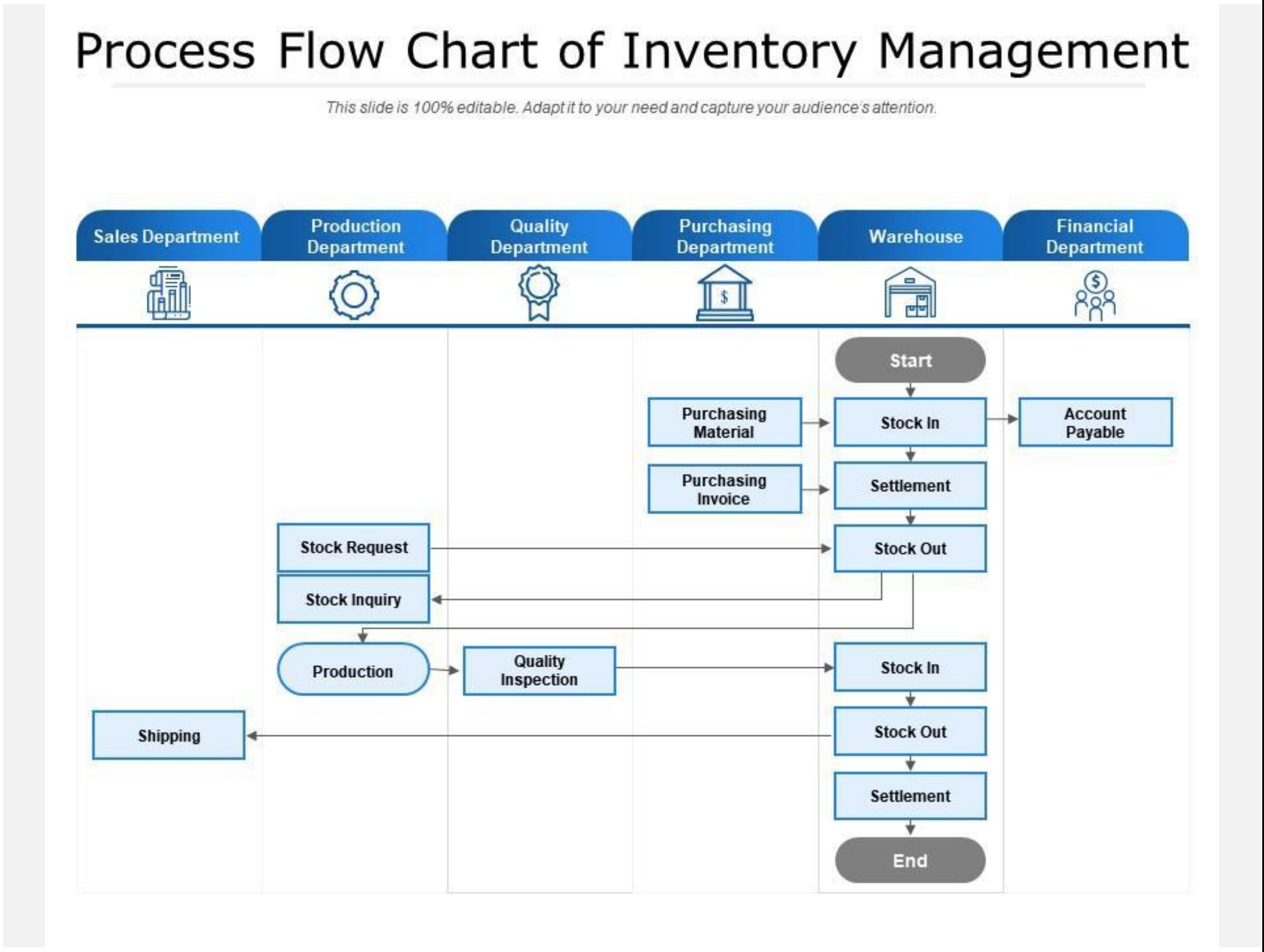


FIG 3. WORKFLOW

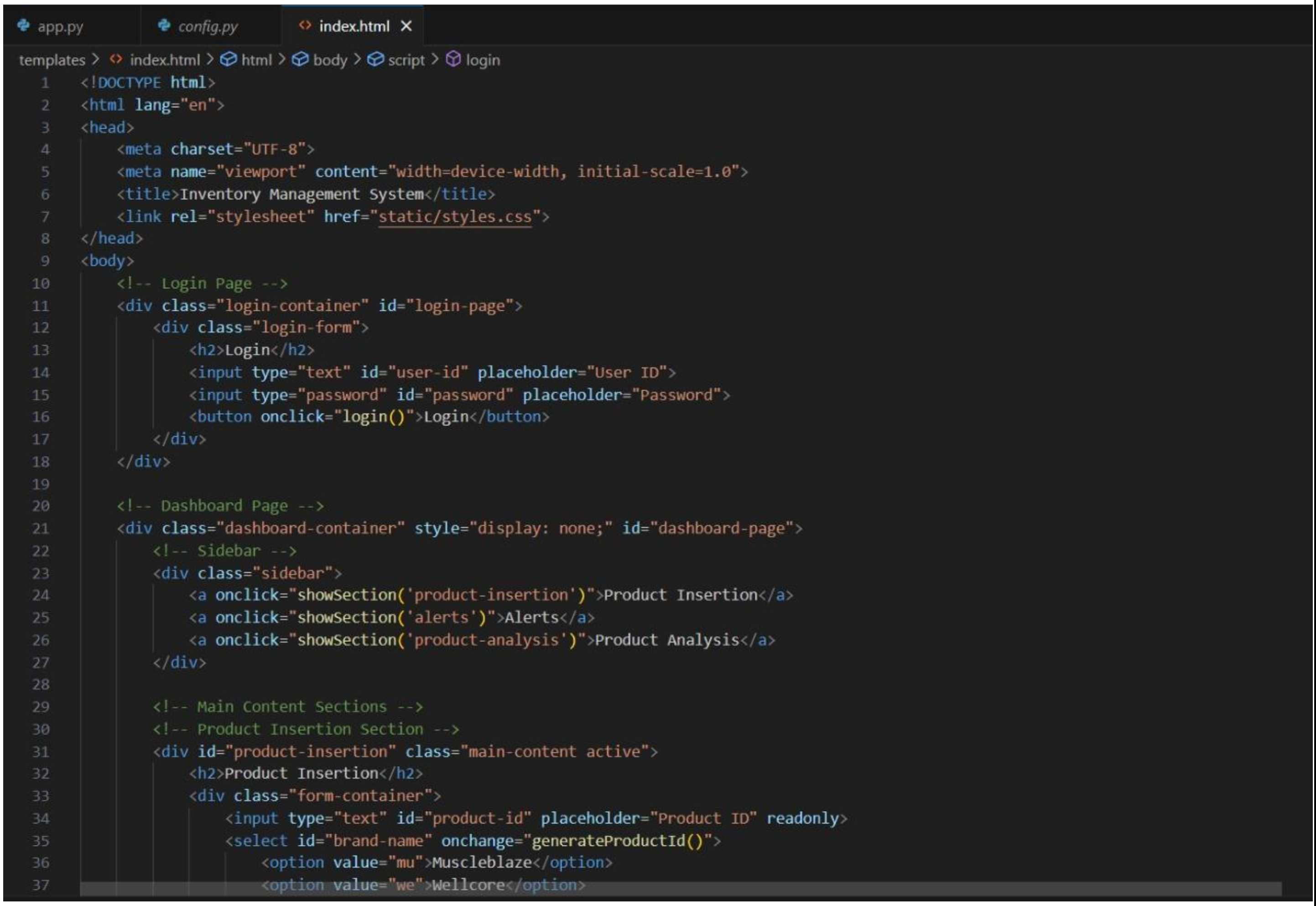
6.IMPLEMENTATION:

6.1 Code Structure and Organization

1. Core Application Directory (app/):
   - routes.py: Handles application routes and defines HTTP endpoints.
   - models.py: Defines database schema using ORM (e.g., SQLAlchemy).
   - forms.py: Manages user input forms using Flask-WTF.
   - utils.py: Contains helper functions for reusable tasks like validation and formatting.
2. Templates Directory (templates/):
   - base.html: Base layout for consistent UI design.
   - Specific templates for modules like dashboard, inventory management, and reports.
3. Static Assets (static/):
   - Subdirectories for CSS, JavaScript, and images to manage front-end assets.
4. Database Migrations (migrations/):
   - Stores migration files for schema updates using Flask-Migrate.
5. Configuration and Dependencies:
   - config.py: Centralized application settings (e.g., database URI, secret keys).
   - requirements.txt: Lists Python dependencies for easy environment setup.
6. Entry Point (run.py):
   - Main script to initialize and run the Flask application.
7. Testing Directory (tests/):
   - Contains unit and integration tests to ensure application reliability.

This structure promotes modularity, reusability, and ease of maintenance.

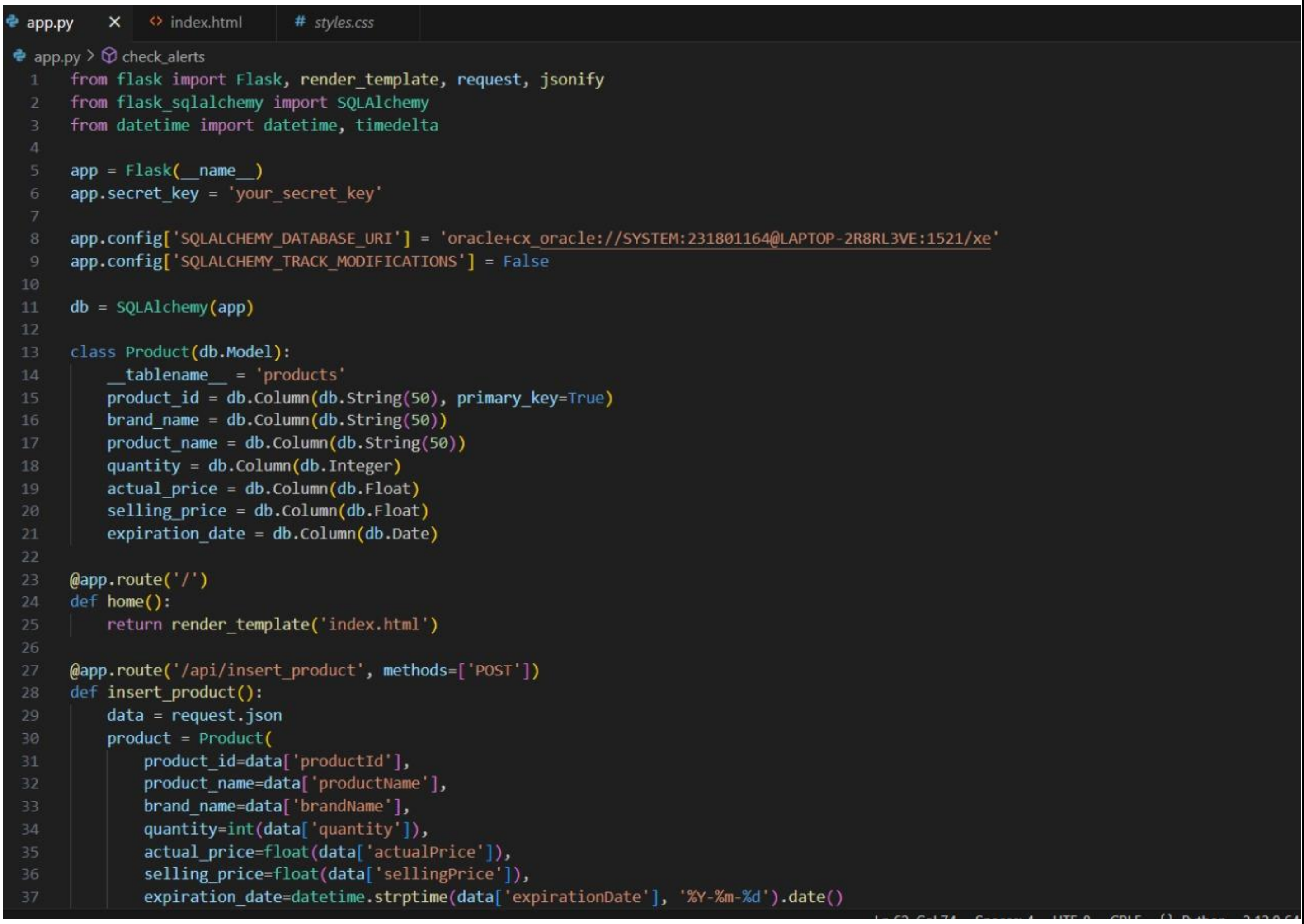
Sample Code



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inventory Management System</title>
    <link rel="stylesheet" href="static/styles.css">
</head>
<body>
    <!-- Login Page -->
    <div class="login-container" id="login-page">
        <div class="login-form">
            <h2>Login</h2>
            <input type="text" id="user-id" placeholder="User ID">
            <input type="password" id="password" placeholder="Password">
            <button onclick="login()">Login</button>
        </div>
    </div>

    <!-- Dashboard Page -->
    <div class="dashboard-container" style="display: none;" id="dashboard-page">
        <!-- Sidebar -->
        <div class="sidebar">
            <a onclick="showSection('product-insertion')">Product Insertion</a>
            <a onclick="showSection('alerts')">Alerts</a>
            <a onclick="showSection('product-analysis')">Product Analysis</a>
        </div>

        <!-- Main Content Sections -->
        <!-- Product Insertion Section -->
        <div id="product-insertion" class="main-content active">
            <h2>Product Insertion</h2>
            <div class="form-container">
                <input type="text" id="product-id" placeholder="Product ID" readonly>
                <select id="brand-name" onchange="generateProductId()">
                    <option value="mu">Muscleblaze</option>
                    <option value="we">Wellcore</option>
```



```python
from flask import Flask, render_template, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timedelta

app = Flask(__name__)
app.secret_key = 'your_secret_key'

app.config['SQLALCHEMY_DATABASE_URI'] = 'oracle+cx_oracle://SYSTEM:231801164@LAPTOP-2R8RL3VE:1521/xe'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class Product(db.Model):
    __tablename__ = 'products'
    product_id = db.Column(db.String(50), primary_key=True)
    brand_name = db.Column(db.String(50))
    product_name = db.Column(db.String(50))
    quantity = db.Column(db.Integer)
    actual_price = db.Column(db.Float)
    selling_price = db.Column(db.Float)
    expiration_date = db.Column(db.Date)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/api/insert_product', methods=['POST'])
def insert_product():
    data = request.json
    product = Product(
        product_id=data['productId'],
        product_name=data['productName'],
        brand_name=data['brandName'],
        quantity=int(data['quantity']),
        actual_price=float(data['actualPrice']),
        selling_price=float(data['sellingPrice']),
        expiration_date=datetime.strptime(data['expirationDate'], '%Y-%m-%d').date()
```

6.2 Key Modules and Their Functions

The User Management Module ensures secure access to the system through authentication and role-based access control. It allows administrators to manage user accounts by adding, updating, or deleting users and ensures session security for all users.

The Inventory Management Module facilitates CRUD operations on inventory items, enabling users to add, update, or remove products. It tracks stock levels in real-time and sends alerts when inventory falls below predefined thresholds, helping businesses avoid stockouts or overstocking.

The Supplier Management Module organizes supplier information, such as contact details and product offerings, and tracks supplier performance. It supports efficient reorder management with automated notifications, ensuring timely procurement.

The Order Processing Module streamlines the handling of sales and purchase orders, updating stock levels automatically after each transaction. It maintains a detailed history of transactions and monitors order statuses to improve tracking.

The Reporting and Analytics Module generates detailed reports and data visualizations, offering insights into stock trends, supplier performance, and overall inventory health. These reports aid in forecasting demand and optimizing stock levels.

Finally, the Dashboard Module provides a centralized view of key inventory metrics, pending orders, and alerts, offering real-time data for quick decision-making and system monitoring.

6.3 Challenges and Solutions

1. Data Redundancy and Inconsistency:

- **Challenge: Maintaining accurate and consistent data across inventory records can be difficult, leading to duplication or errors.**
- **Solution: Use database normalization and enforce primary-foreign key relationships to ensure data integrity and eliminate redundancy.**

2. Scalability Issues:

- **Challenge: The system may struggle to handle increasing data volumes as the business grows.**
- **Solution: Implement a scalable database (e.g., MySQL or PostgreSQL) and use indexing for faster queries and performance optimization.**

3. User Access Control:

- **Challenge: Preventing unauthorized access or modifications to sensitive inventory data.**
- **Solution: Implement role-based access control (RBAC) and secure authentication mechanisms.**

5.Real-Time Updates:

- **Challenge: Ensuring inventory data reflects real-time stock levels during simultaneous transactions.**
- **Solution: Use transaction management and locking mechanisms in the database to maintain data accuracy.**

5. User Adoption:

- **Challenge: Users may face difficulties adapting to the system.**
- **Solution: Design a user-friendly interface with intuitive workflows and provide training for seamless adoption**

7. TESTING AND VALIDATION

7.1 Testing Strategies

The testing process for the Inventory Management System (IMS) begins with **unit testing**, where individual components such as database models, routes, and utility functions are tested in isolation. This ensures each element performs as expected before integration with other parts of the system.
Next, **integration testing** validates the interaction between different modules, such as the communication between the UI and the database. This ensures seamless data flow, particularly for critical operations like stock updates and order processing. **Functional testing** follows, verifying that all features, including CRUD operations, reporting, and low-stock alerts, meet the specified requirements.
To ensure the system can handle real-world demands, **performance testing** is conducted to assess response times, scalability, and stability under varying workloads. **Security testing** is crucial to protect sensitive data, testing authentication, role-based access controls, and data integrity mechanisms for vulnerabilities. Finally, **User Acceptance Testing (UAT)** involves end-users to validate usability and ensure the system aligns with business needs, guaranteeing successful adoption.

7.2 Test Cases and Results

1. Login Functionality: Verify user authentication with valid and invalid credentials.
   - Result: Successful login for valid users; error messages for invalid attempts.
2. CRUD Operations: Test adding, updating, and deleting inventory items.
   - Result: Accurate database updates and UI changes for each operation.
3. Low-Stock Alerts: Trigger alerts when stock falls below the threshold.
   - Result: Alerts generated correctly based on predefined limits.
4. Order Processing: Validate stock updates after purchase or sales orders.
   - Result: Inventory levels updated in real-time.
5. Reports: Generate reports on stock trends and supplier performance.
   - Result: Accurate and readable reports generated successfully.
6. Access Control: Test role-based permissions for different user roles.

- Result: Permissions restricted as defined by user roles

7.3 Bug Fixes and Improvements

The latest release of our Inventory Management System introduces significant bug fixes and enhancements to improve functionality, reliability, and user experience. Key fixes include resolving issues with stock discrepancies during concurrent transactions, ensuring accurate real-time updates and data consistency. We've optimized database queries to handle large-scale inventory datasets efficiently, reducing response times and minimizing system lag.

The stock report generation process has been fixed to eliminate errors in item counts, ensuring accurate and reliable reporting. Search functionality has been upgraded with advanced filters, enabling users to quickly find items based on specific criteria like category, supplier, or stock levels. Low-stock notifications were improved to ensure timely alerts, preventing inventory shortages.

In terms of security, we addressed vulnerabilities by enhancing user authentication systems with stronger password policies and optional two-factor authentication. Bulk inventory updates are now supported through batch processing, significantly improving operational efficiency for large-scale changes. The user interface has been redesigned for a more intuitive and user-friendly experience, particularly for mobile users.

Additional improvements include automated database backup mechanisms to safeguard data against potential losses and better error-handling features to provide clear guidance when issues arise. These updates ensure smoother inventory operations, enhanced system reliability, and a seamless user experience.

8. RESULTS AND DISCUSSION

8.1 Summary of Features

Our Inventory Management System offers a comprehensive suite of tools to simplify and streamline inventory operations. It ensures accurate real-time stock updates, robust security, and user-friendly functionality across devices. The system enhances efficiency with features designed for detailed reporting, automated alerts, and bulk management capabilities.

Key Features:

- Real-time tracking of stock levels and updates.
- Advanced search functionality with customizable filters.
- Automated low-stock and overstock notifications.
- Detailed reporting for stock, sales, and purchase analysis.
- Bulk inventory updates through batch processing.
- Secure user authentication with optional two-factor login.
- Mobile-friendly interface for on-the-go access.
- Automated backups to prevent data loss.

8.2 User Experience Feedback

The Inventory Management System has been well-received for its streamlined functionality and ease of use. Users highlighted several features that improved their overall experience:

Key Feedback Points:

- Real-time stock updates ensure accurate and reliable inventory tracking.
- Advanced search filters save time by enabling quick and precise item retrieval.
- Automated low-stock alerts help prevent inventory shortages efficiently.
- The mobile-friendly interface allows seamless inventory management across devices.
- Security features, including two-factor authentication, provide peace of mind.
- Intuitive design and error-handling make it easy for new users to navigate.

The system is praised for enhancing productivity and reducing operational errors.

8.3 Potential Improvements

While the Inventory Management System is robust, several enhancements could further improve its efficiency and user experience:

Key Improvement Areas:

- **Customizable Dashboards:** Allow users to personalize their dashboard views for quick access to frequently used features.
- **Integration with Third-Party Tools:** Support for accounting software, e-commerce platforms, and supplier management tools for seamless operations.
- **AI-Powered Forecasting:** Introduce demand forecasting features to predict inventory needs based on historical trends and sales data.

- **Enhanced User Roles:** Offer more granular control over user permissions to improve security and workflow management.
- **Barcode and RFID Integration:** Enable faster stock tracking and updates with automated scanning technologies.
- **Offline Mode:** Allow users to work offline with automatic synchronization once reconnected.
- **Multi-Warehouse Support:** Provide better tools for managing inventory across multiple locations.
- **Customer Feedback Portal:** Incorporate a built-in feedback mechanism for continuous improvement.

Implementing these enhancements would make the system even more versatile, efficient, and user-centric.

Output:

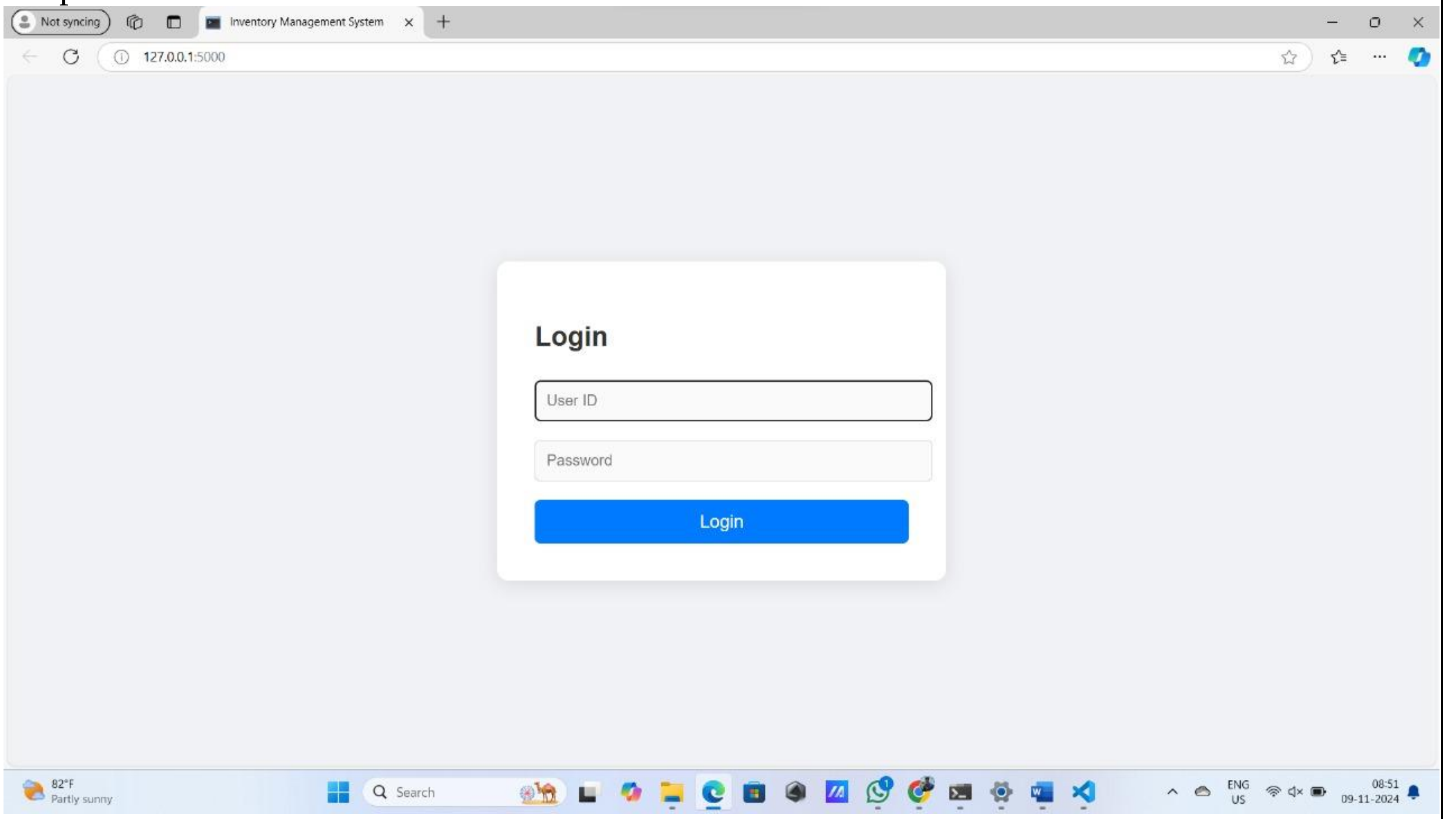
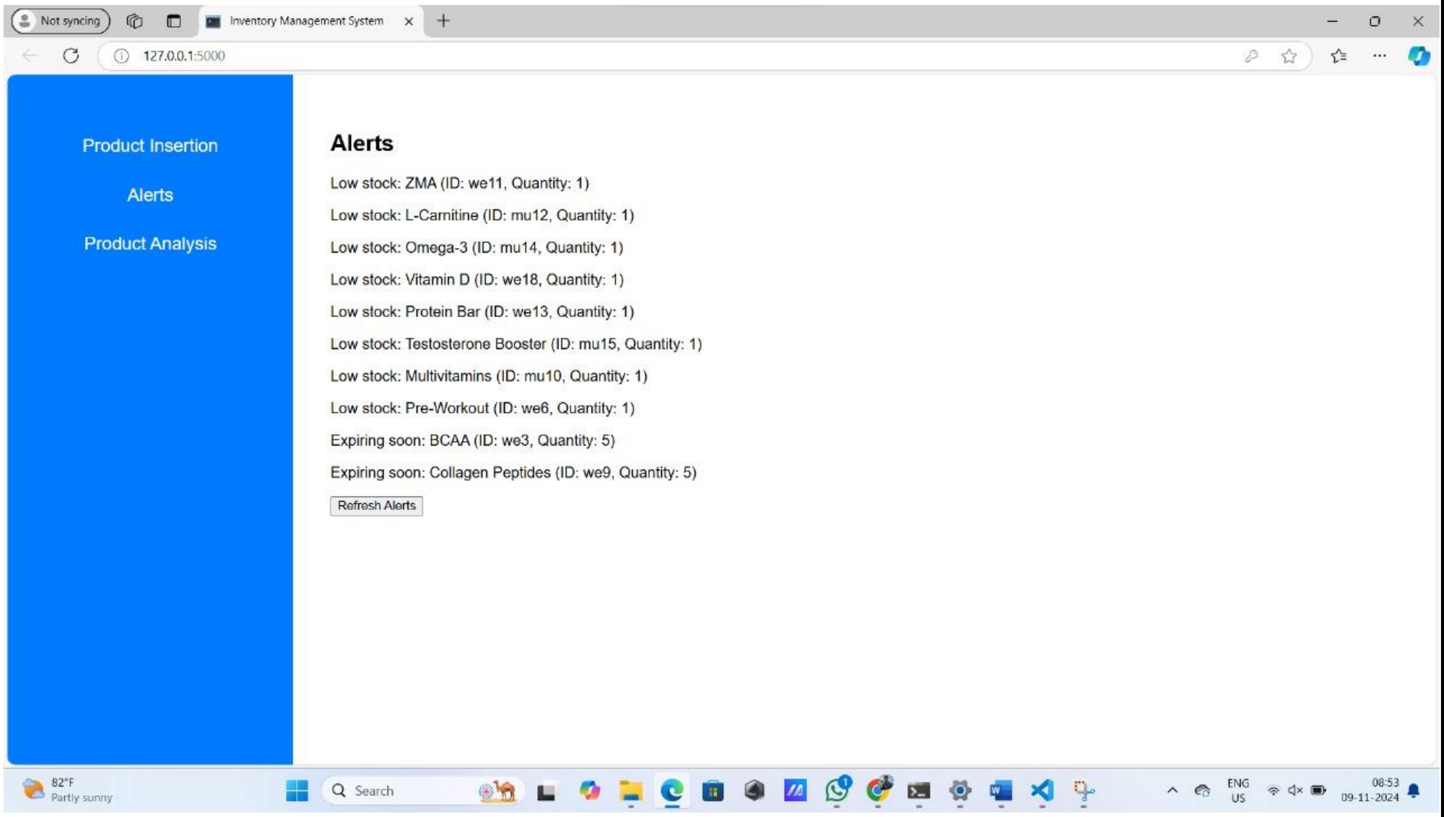


FIG 4.LOGIN PAGE



Alerts

Low stock: ZMA (ID: we11, Quantity: 1)

Low stock: L-Carnitine (ID: mu12, Quantity: 1)

Low stock: Omega-3 (ID: mu14, Quantity: 1)

Low stock: Vitamin D (ID: we18, Quantity: 1)

Low stock: Protein Bar (ID: we13, Quantity: 1)

Low stock: Testosterone Booster (ID: mu15, Quantity: 1)

Low stock: Multivitamins (ID: mu10, Quantity: 1)

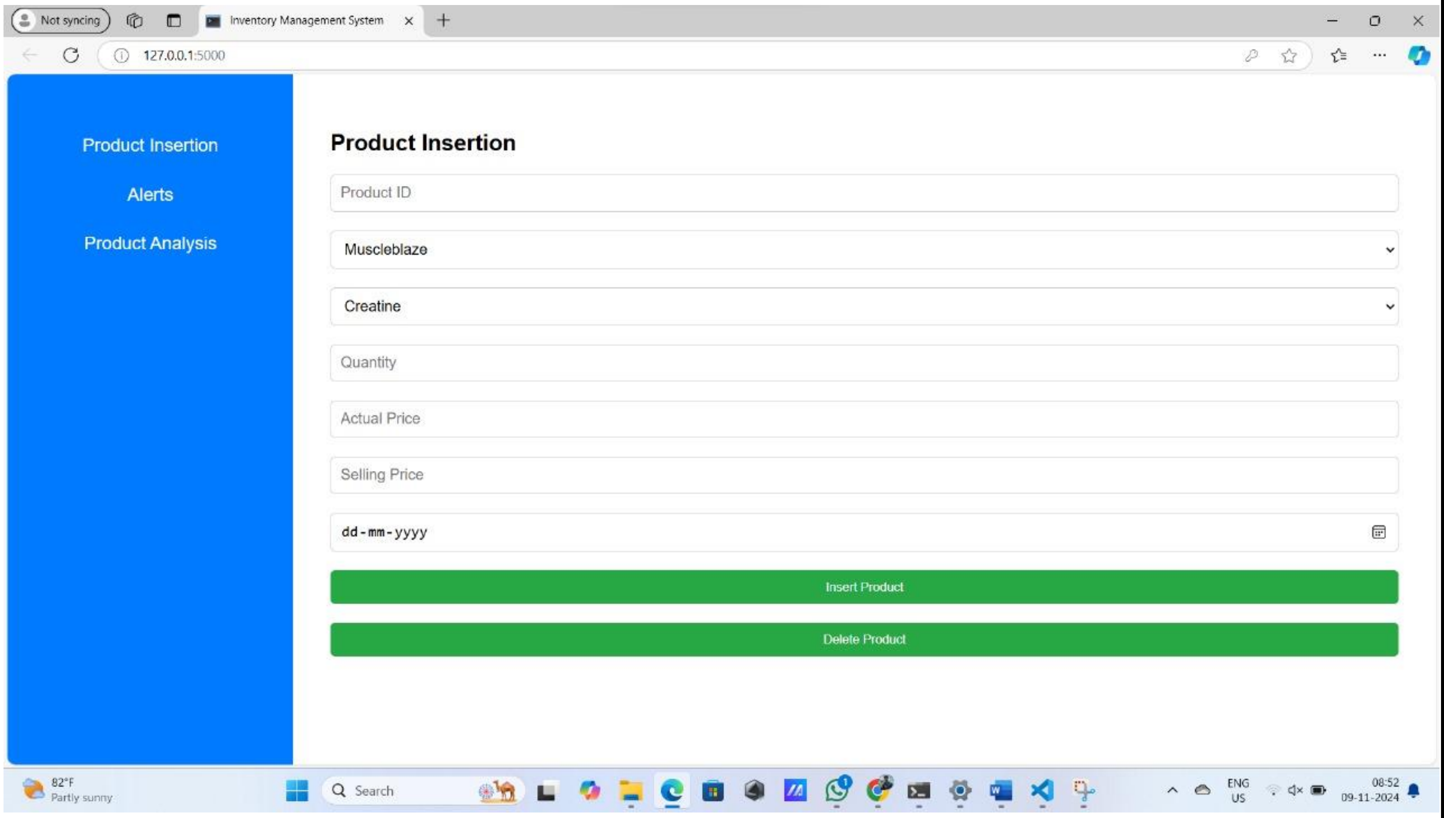
Low stock: Pre-Workout (ID: we6, Quantity: 1)

Expiring soon: BCAA (ID: we3, Quantity: 5)

Expiring soon: Collagen Peptides (ID: we9, Quantity: 5)

Refresh Alerts

FIG 5. ALERTS



Product Insertion

Product ID

Muscleblaze

Creatine

Quantity

Actual Price

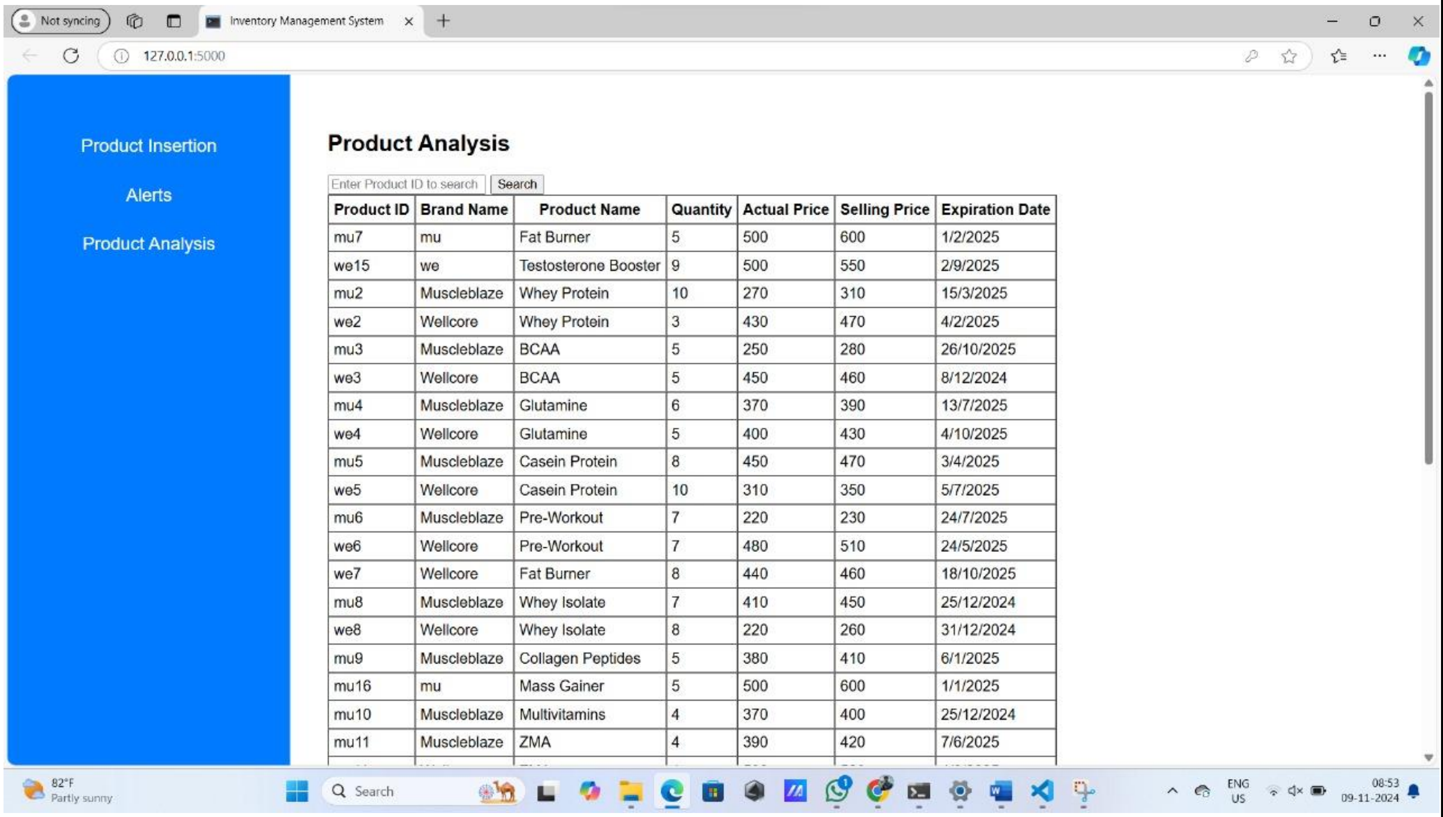
Selling Price

dd-mm-yyyy

Insert Product

Delete Product

FIG 6. PRODDUCT INSERTION



Product Analysis

Enter Product ID to search | Search

Product ID	Brand Name	Product Name	Quantity	Actual Price	Selling Price	Expiration Date
mu7	mu	Fat Burner	5	500	600	1/2/2025
we15	we	Testosterone Booster	9	500	550	2/9/2025
mu2	Muscleblaze	Whey Protein	10	270	310	15/3/2025
we2	Wellcore	Whey Protein	3	430	470	4/2/2025
mu3	Muscleblaze	BCAA	5	250	280	26/10/2025
we3	Wellcore	BCAA	5	450	460	8/12/2024
mu4	Muscleblaze	Glutamine	6	370	390	13/7/2025
we4	Wellcore	Glutamine	5	400	430	4/10/2025
mu5	Muscleblaze	Casein Protein	8	450	470	3/4/2025
we5	Wellcore	Casein Protein	10	310	350	5/7/2025
mu6	Muscleblaze	Pre-Workout	7	220	230	24/7/2025
we6	Wellcore	Pre-Workout	7	480	510	24/5/2025
we7	Wellcore	Fat Burner	8	440	460	18/10/2025
mu8	Muscleblaze	Whey Isolate	7	410	450	25/12/2024
we8	Wellcore	Whey Isolate	8	220	260	31/12/2024
mu9	Muscleblaze	Collagen Peptides	5	380	410	6/1/2025
mu16	mu	Mass Gainer	5	500	600	1/1/2025
mu10	Muscleblaze	Multivitamins	4	370	400	25/12/2024
mu11	Muscleblaze	ZMA	4	390	420	7/6/2025

FIG 7. PRODUCT ANALYSIS

9.CONCLUSION:

The Inventory Management System (IMS) integrates a relational DBMS like SQL with Flask to streamline inventory tracking and management. By employing DBMS principles such as normalization and enforcing constraints, the system ensures data integrity, eliminates redundancy, and maintains consistency across entities like products, categories, and transactions.

Using Flask as the web framework, the IMS provides a user-friendly interface for performing CRUD (Create, Read, Update, Delete) operations, enabling seamless interaction with the database. ORM tools simplify query execution while ensuring performance and security, making the system efficient and reliable for managing inventory data in real-time.

The IMS offers a scalable solution for businesses to optimize inventory operations and reduce manual efforts. Future enhancements, such as automated reporting, analytics, and external integrations, can further increase its value, making it a comprehensive tool for inventory management

10.REFERENCES :

SQL Database Concepts and Best Practices

- *Reference*: "SQL for Web Developers"
- https://www.w3schools.com/

Flask Framework Documentation

- *Reference*: Flask Documentation (Official Site)
- https://flask.palletsprojects.com/en/stable/

Inventory Management System Design and Best Practices

- *Reference*: "Best Practices for Inventory Management," TradeGecko Blog.
- https://en.wikipedia.org/wiki/TradeGecko

ORMs and Flask Database Integration

- *Reference*: "Flask SQLAlchemy – Using SQLAlchemy with Flask," Real Python.
- https://www.python.org/