

INTRODUCTION

1.1 Problem Definition

In today's rapidly evolving retail industry, supermarkets play a crucial role in fulfilling the daily needs of consumers. With the vast variety of products, high volume of stock movement, and the necessity to ensure freshness and availability, efficient inventory management becomes not only essential but also critical to business success. Traditional inventory systems often rely heavily on manual entries and periodic stocktaking, which are prone to human error, delays, and lack of real-time visibility into stock levels. These shortcomings can lead to problems such as understocking, overstocking, product expiries, and ultimately, customer dissatisfaction or financial loss.

To overcome these challenges, the need for a **Smart Inventory Management System (SIMS)** arises—a technology-driven solution that brings automation, intelligence, and real-time insights into stock handling. The proposed Smart Inventory Management System is designed specifically for supermarkets, aiming to streamline and enhance inventory processes with the help of modern technologies such as Flask (Python web framework), MySQL (for backend data storage), and AWS Simple Notification Service (SNS) for automated notifications.

This system continuously monitors product stock levels and expiry dates. When the quantity of a product drops below a predefined threshold (understock) or exceeds a maximum limit (overstock), the system automatically detects the issue. Similarly, products nearing their expiration date are flagged in advance. These real-time insights are critical for proactive decision-making. The system then triggers alerts to the store administrator and suppliers using AWS SNS, sending SMS or email notifications to subscribed users. This ensures that essential products are reordered in time and excessive or expired stock is avoided.

Moreover, the platform provides an admin dashboard that displays a summary of sales, purchases, expiry alerts, understocked items, and reorder requests. Admins have the capability to review reorder requests and either approve or reject them.

If approved, a notification is sent directly to the supplier, thereby reducing the communication gap and enhancing supplier coordination. The use of AWS SNS ensures that the system remains scalable, secure, and reliable when sending real-time alerts to multiple recipients.

The integration of smart features not only reduces manual work load but also enhances overall inventory visibility, minimizes waste, and improves cost efficiency. By implementing such a system, supermarkets can better manage their inventory, reduce losses due to expired or unsold goods, and offer a better shopping experience for their customers.

This project, Smart Inventory Management System for Supermarkets, addresses these issues by leveraging a web - based application built using Flask, integrated with a MySQL database and Amazon Web Services (AWS) Simple Notification Service (SNS). The system automatically monitors stock levels and product expiry dates, sends real - time notifications for understocked and overstocked items, and allows administrators to approve or reject reorder requests. Notifications are sent directly to subscribed users or suppliers via SMS or email using AWS SNS, ensuring timely and efficient inventory control.

In conclusion, the Smart Inventory Management System for Supermarkets is a robust, user-friendly, and scalable solution that transforms traditional inventory management into a proactive and automated process. By harnessing cloud services, web technologies, and intelligent automation, it significantly improves inventory accuracy, operational efficiency, and customer satisfaction in the retail sector.

SYSTEM ANALYSIS

2.1 Existing System

1. **Manual Inventory Tracking :** Most supermarkets use paper logs or basic spreadsheets to manage stock, which leads to human errors, delayed updates, and inefficient inventory management.
2. **No Real-Time Stock Alerts :** There is no automated system to alert staff when items are understocked or overstocked. As a result, reordering is often reactive rather than proactive, causing delays and missed sales.
3. **Poor Expiry Date Monitoring :** Expiry dates are tracked manually, increasing the risk of expired goods remaining on shelves. This can lead to health risks, product wastage, and customer dissatisfaction.
4. **Lack of Automated Supplier Communication :** Reorders are placed manually via phone or email, which is time - consuming and not scalable. Delays in supplier communication can result in stockouts.
5. **Limited Reporting and No Remote Access :** Traditional systems do not provide detailed analytics or dashboards. Also, they lack cloud or mobile access, preventing managers from monitoring inventory remotely or in real time.

Disadvantages:

- ❖ Manual Errors - Data entry by hand often leads to mistakes such as wrong quantities, missed expiry dates, or incorrect product details. These small errors can cause major issues in stock management.
- ❖ Time - Consuming Processes - Inventory checks, reorder decisions, and supplier communications take a lot of time when done manually. This delays actions and reduces overall productivity.
- ❖ No Real-Time Monitoring : Traditional systems don't provide real-time updates, meaning stock levels can be inaccurate. Managers may not notice shortages or excess stock until it's too late.
- ❖ Lack of Notifications and Alerts : There are no automated alerts for understock, overstock, or upcoming expiry dates. This can lead to empty shelves, financial loss, and wasted products.

- ❖ **Limited Decision -Making Support** : Without automated reports or analytics, it's difficult to make informed business decisions. Managers lack insights into sales trends, reorder frequency, or profitability.
- ❖ **High Risk of Stockouts or Overstocking** : Without automated thresholds and reorder triggers, stores often run out of popular items or over-purchase slow-moving stock.
- ❖ **Inefficient Supplier Coordination** : Communication with suppliers is usually done manually, increasing the chances of delays and miscommunication in placing orders.

2.2 Proposed System

➤ **Automated Stock Monitoring**

The system continuously monitors inventory levels and categorizes products based on predefined thresholds (understocked, overstocked).

This reduces manual checking and provides live updates on stock availability.

➤ **Expiry Date Tracking**

Products nearing expiration are detected based on a configured date range (e.g., 7 days before expiry). Notifications are generated to alert the admin to take necessary actions such as discounts or removals.

➤ **Real-Time Notifications via AWS SNS**

The system sends automated alerts to subscribed users or managers via SMS / email using **Amazon SNS**. Alerts include :

- ◆ Understock warnings
- ◆ Overstock notification
- ◆ Product expiry alerts
- ◆ Approved reorder confirmations to suppliers

➤ **Reorder Request Management**

A built-in reorder approval system allows the admin to view understocked items and approve or reject reorder requests. Once approved, a confirmation message is automatically sent to the supplier via email or SMS.

➤ **User Authentication**

Only authorized users (admins) can access the system, preventing unauthorized access to sensitive inventory data.

➤ **Dashboard Overview**

The admin can view a summary of stock conditions, including low stock alerts, expiry warnings, and reorder statuses, all in one place.

➤ **Data Storage in Cloud (AWS RDS)**

The system utilizes **AWS RDS (MySQL)** for secure and scalable cloud - based data storage, enabling fast and reliable access to inventory records from anywhere.

ADVANTAGES:

- ❖ **Real-Time Inventory Visibility :** The system offers instant access to stock levels, product categories, pricing, and expiry details, allowing the admin to make fast and informed decisions.
- ❖ **Reduces Stockouts and Overstock Situations :** By automatically tracking stock against set thresholds, the system ensures timely reorders for understocked products and alerts on overstocked items, reducing waste and lost sales.
- ❖ **Cost Efficiency :** Automation reduces the need for manual labor in inventory tracking and ordering processes, which helps in minimizing operational costs.
- ❖ **Supplier Communication Automation :** Through AWS SNS integration, suppliers receive instant notifications when orders are approved - speeding up the restocking process and reducing downtime.
- ❖ **Enhances Productivity :** By minimizing manual checks and automating alerts and reports, employees can focus on more critical areas of the business, boosting overall productivity.
- ❖ **User-Friendly Interface :** The system provides an intuitive web-based dashboard for admins to easily view reports, manage stock, approve orders, and monitor alerts without requiring technical expertise.
- ❖ **Data Security and Backup :** With data stored on **AWS RDS**, the system benefits from built-in security features, data encryption, and reliable backup mechanisms, reducing the risk of data loss.

SYSTEM REQUIREMENT

3.1 Hardware Specification

- ◆ **Processor** : AWS EC2 t2.medium
- ◆ **RAM** : Minimum 4 GB (8 GB recommended)
- ◆ **Storage** : 50 GB SSD
- ◆ **Input Devices** : Keyboard, Mouse
- ◆ **Network** : Stable internet connection with public IP for external access

3.2 Software Specification

- ◆ **Operating System** : Linux (AWS Linux 2)
- ◆ **Frontend** : Html, CSS, js
- ◆ **Backend** : Python
- ◆ **Developed Platform** : Pycharm
- ◆ **Database** : MySQL (hosted on AWS RDS)
- ◆ **Cloud Services** : AWS SNS (Simple Notification Services)

3.3 Software Description

AWS Linux : Amazon Linux 2 is a stable, secure, and high-performance Linux environment provided by Amazon Web Services (AWS). It is designed specifically for cloud-based applications, offering long-term support and integration with many AWS tools and services. As an evolution of the original Amazon Linux, Amazon Linux 2 provides improved system stability, performance tuning, and development support, making it an ideal operating system for both development and production environments in the AWS cloud. Amazon Linux 2 supports modern versions of essential programming languages, system tools, and development frameworks, which makes it highly suitable for deploying applications like web servers, backend services, and microservices. It comes with the latest software packages through the Amazon Linux Extras repository and uses the yum package manager, which simplifies software installation and

management. A key feature of Amazon Linux 2 is its tight integration with AWS infrastructure. It works seamlessly with EC2 instances, supports Docker containers, and integrates smoothly with services like **AWS Identity and Access Management (IAM)**, **Amazon RDS**, and **Amazon SNS**. Developers can configure automatic scaling, load balancing, and cloud storage access directly on the OS, thanks to its built-in AWS compatibility. Amazon Linux 2 also emphasizes **security**.

Python : Python is an interpreter, interactive, object - oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It supports multiple programming paradigms beyond object - oriented programming, such as procedural and functional programming. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many UNIX variants including Linux and Mac OS, and on windows. Python is a high - level general - purpose programming language that can be applied to many different classes of problem.

MySQL : **MySQL** is an open-source, relational database management system (RDBMS) that is widely used for managing and organizing data in structured formats. It is based on the **Structured Query Language (SQL)**, which is the standard language for accessing and manipulating databases. Developed originally by MySQL AB and now owned by Oracle Corporation, MySQL has become one of the most popular database systems due to its speed, reliability, and ease of use. MySQL is widely used in web applications and is often the database of choice for popular platforms such as WordPress, Facebook, and Twitter. It is also highly compatible with major programming languages like Python, PHP, Java, and more, making it versatile and suitable for both small-scale and enterprise-level projects. One of the main strengths of MySQL is its **client-server architecture**. In this model, the MySQL server manages database files and client programs send queries to the server, which processes them and returns results. MySQL supports multiple users and databases simultaneously, ensuring that applications can scale easily without performance degradation.

AWS SNS : **Amazon Simple Notification Service (Amazon SNS)** is a fully managed messaging service provided by **Amazon Web Services (AWS)** that enables the delivery of messages to a large number of recipients in real time. SNS is designed to support application-to-

application (A2A) and application-to-person (A2P) communication, making it ideal for sending notifications, alerts, and updates to various endpoints such as **email**, **SMS (text messages)**, **mobile push notifications**, and even other AWS services. At its core, SNS follows a **publish-subscribe (pub/sub) model**. In this model, applications (called **publishers**) send messages to a **topic**, and subscribers (such as mobile numbers, email addresses, or Lambda functions) receive the messages based on their subscription to that topic. This architecture enables scalable and decoupled communication between services or users.

HTML : Html elements such as headings, paragraphs, lists, and sections. This would make content difficult to organize and understand. HTML tags define how content should be displayed, including text formatting, font styles, colors, and layout. Without HTML, web pages would appear as unformatted blocks of text. HTML's hyper linking feature allows users to navigate between different web pages and resources. Without HTML, there would be no clickable links, making impossible to navigate the web. HTML tags such as , <audio>, <video> allow for the embedding of images, audio, video, and media types within web pages. Without HTML, multimedia content couldn't be integrated into web pages. HTML's semantic elements provide meaning to different parts of a webpage, aiding accessibility, SEO (Search Engine Optimization), and content interpretation. Without HTML, web content would lack semantic structure, making it less accessible and harder for search engines to index. HTML is essential for creating structured, formatted, and interactive web pages. Without it, the web as we know it would be a chaotic mess of unformatted text with no navigation, interactivity, or multimedia content.

CSS : It is a style sheet language used to describe the presentation of a document written in HTML or XML. CSS describes how HTML elements should be displayed on screen, in print, or in other media. It controls the layout, design, colors, fonts, and other visual aspects of web pages. By separating the content (HTML) from its presentation (CSS), web developers can create more flexible and consistent designs across multiple web pages and services. CSS uses selectors to target HTML elements that you want to style. Selectors can target elements by tag name, class, ID, attributes, or their relationship with other elements. CSS properties define specific visual aspects of an element, such as its color, size, margin, padding, font, and position. Each property has one or more values that specify how the property should be applied.

SYSTEM STUDY

A system study for a parking management system involves a thorough analysis of the existing parking space management process. The study would aim to identify the problems and inefficiencies of the current system, as well as the opportunities for improvement. The study would start by collecting data on the current parking system, such as the number of parking spaces, the frequency of vehicles entering and exiting the parking lot, and the parking fees charged. This information would be used to identify the bottlenecks and issues in the system. Next, the stakeholders involved in the parking management system, such as the parking lot operator, vehicle owners, and regulatory authorities, would be identified, and their needs and expectations would be studied.

This would help design a parking management system that meets the requirements of all stakeholders. The system study would also involve researching best practices and available technology solutions in the parking management industry. Once the research is complete, a conceptual design of the proposed parking management system would be developed, including the hardware and software components, as well as the user interface and user experience design.

The conceptual design would then be evaluated for feasibility, cost-effectiveness, and scalability. Finally, a detailed project plan for the implementation of the proposed parking management system would be developed, including timelines, budget, and resource requirements. The plan would also include a detailed risk assessment and mitigation strategy to ensure successful implementation of the parking management system.

SYSTEM DESIGN

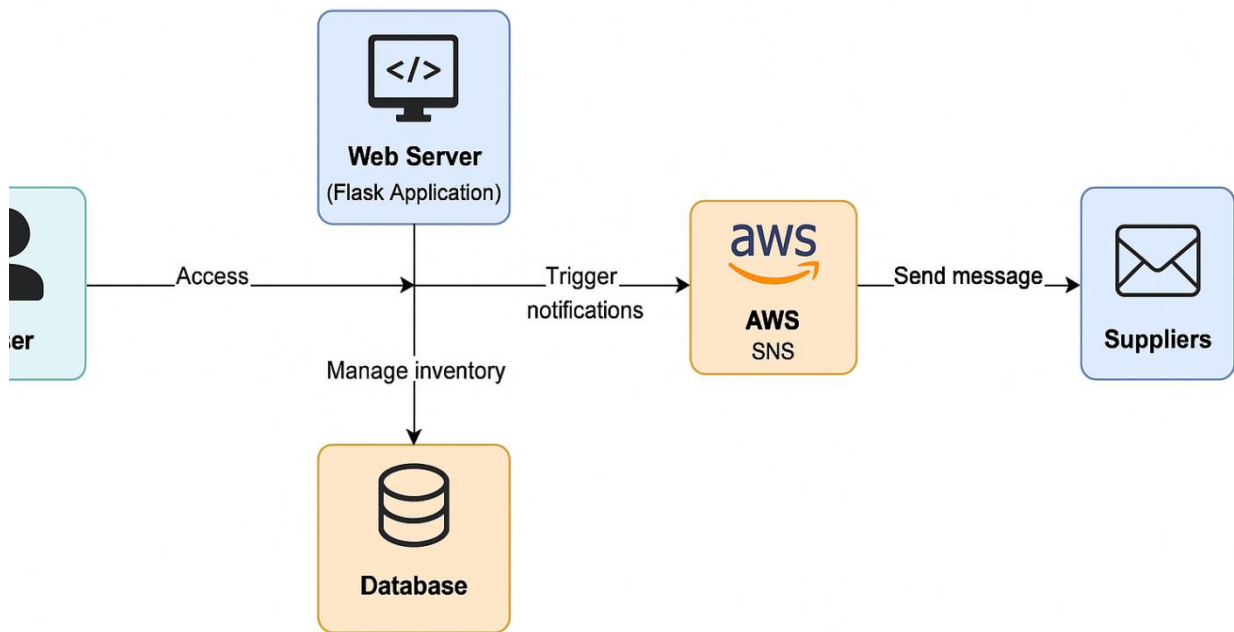
The **Smart Inventory Management System** is designed to streamline and automate the management of supermarket inventory. The system architecture integrates a web-based application, a centralized cloud-hosted database, and an automated notification service using AWS. This design ensures the system operates efficiently, maintains stock levels, alerts administrators of critical changes, and facilitates supplier communication without manual intervention.

5.1 System Architecture

The system follows a **three-tier architecture** consisting of the **Presentation Layer**, **Application Layer**, and **Data Layer**:

- ❖ **Presentation Layer:** This is the user interface developed using HTML, CSS, and JavaScript, served through Flask templates. It enables users (e.g., supermarket admins) to interact with the system through dashboards, reports, reorder screens, and login pages.
- ❖ **Application Layer:** Built using Python with Flask, this layer handles the business logic. It processes user inputs, queries the database, checks inventory thresholds, triggers notifications, and updates reorder statuses.
- ❖ **Data Layer:** A **MySQL database** hosted on **AWS RDS** serves as the central storage for all inventory-related data. It stores product details, stock levels, reorder requests, user credentials, and expiration dates.

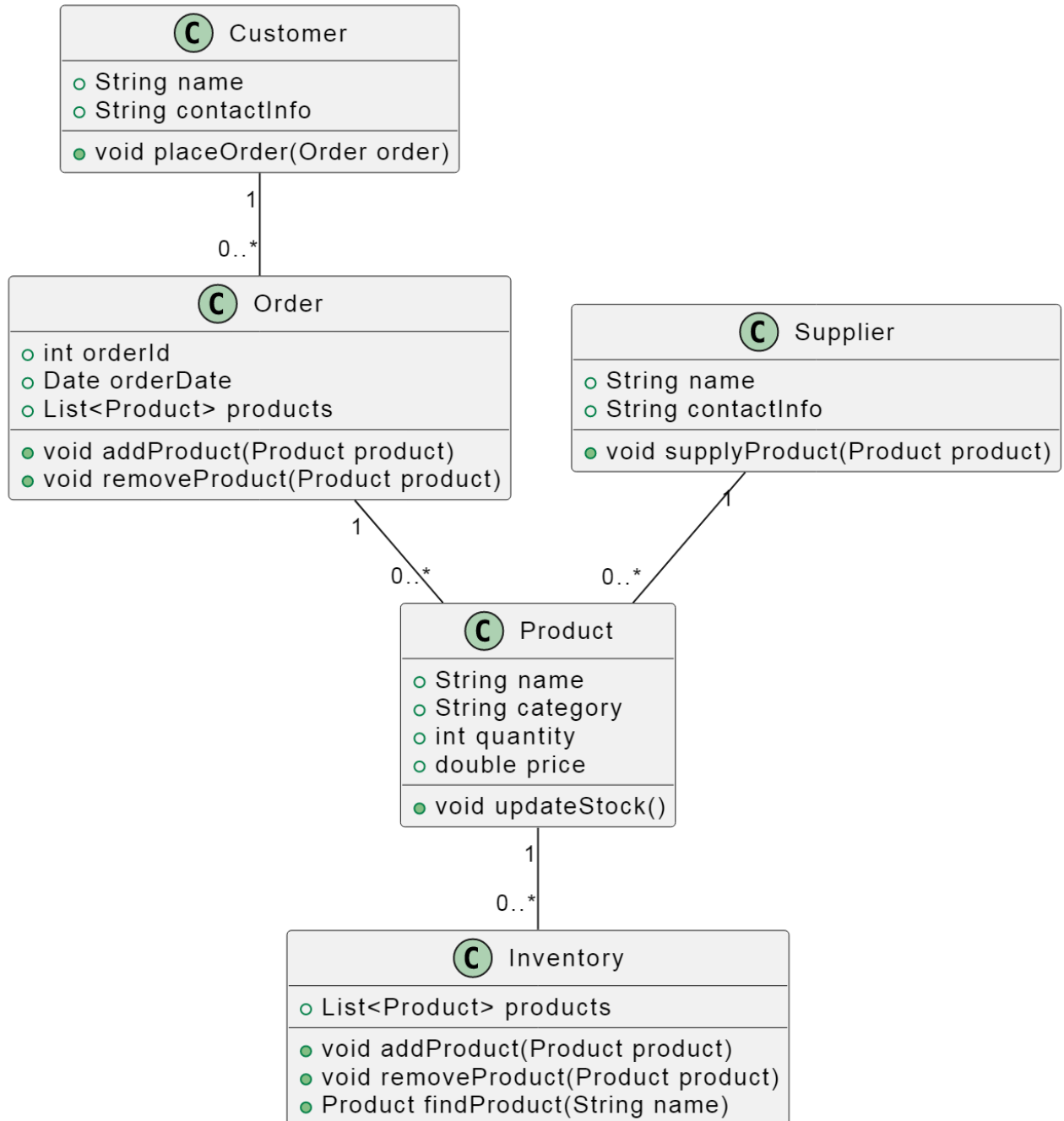
Smart Inventory Management System



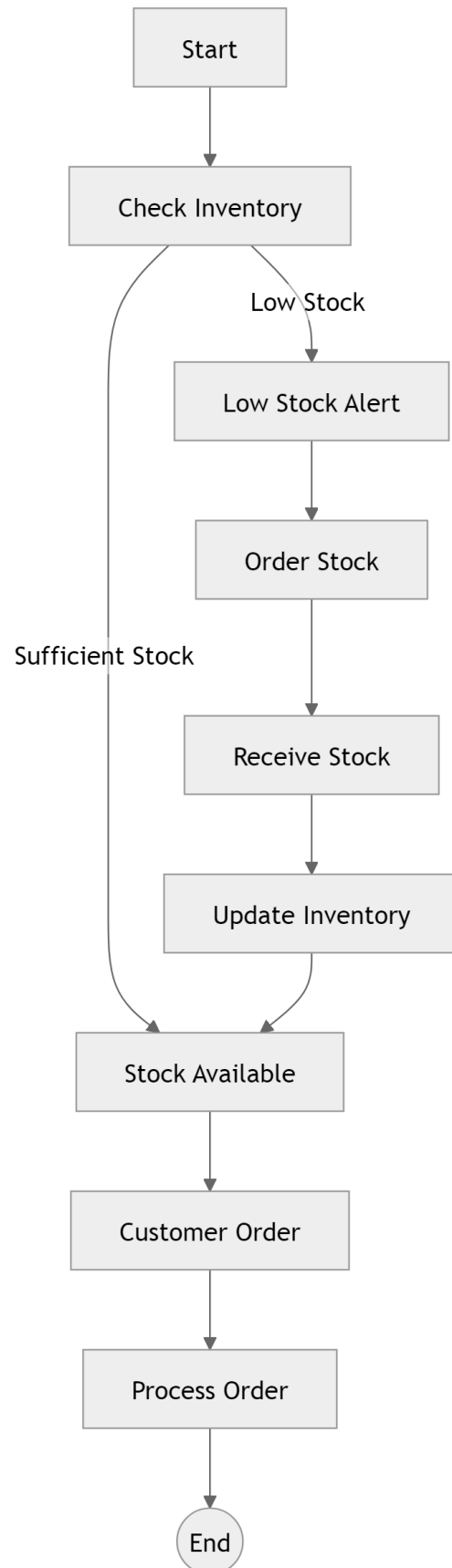
SYSTEM ARCHITECTURE

5.2 UML Diagram

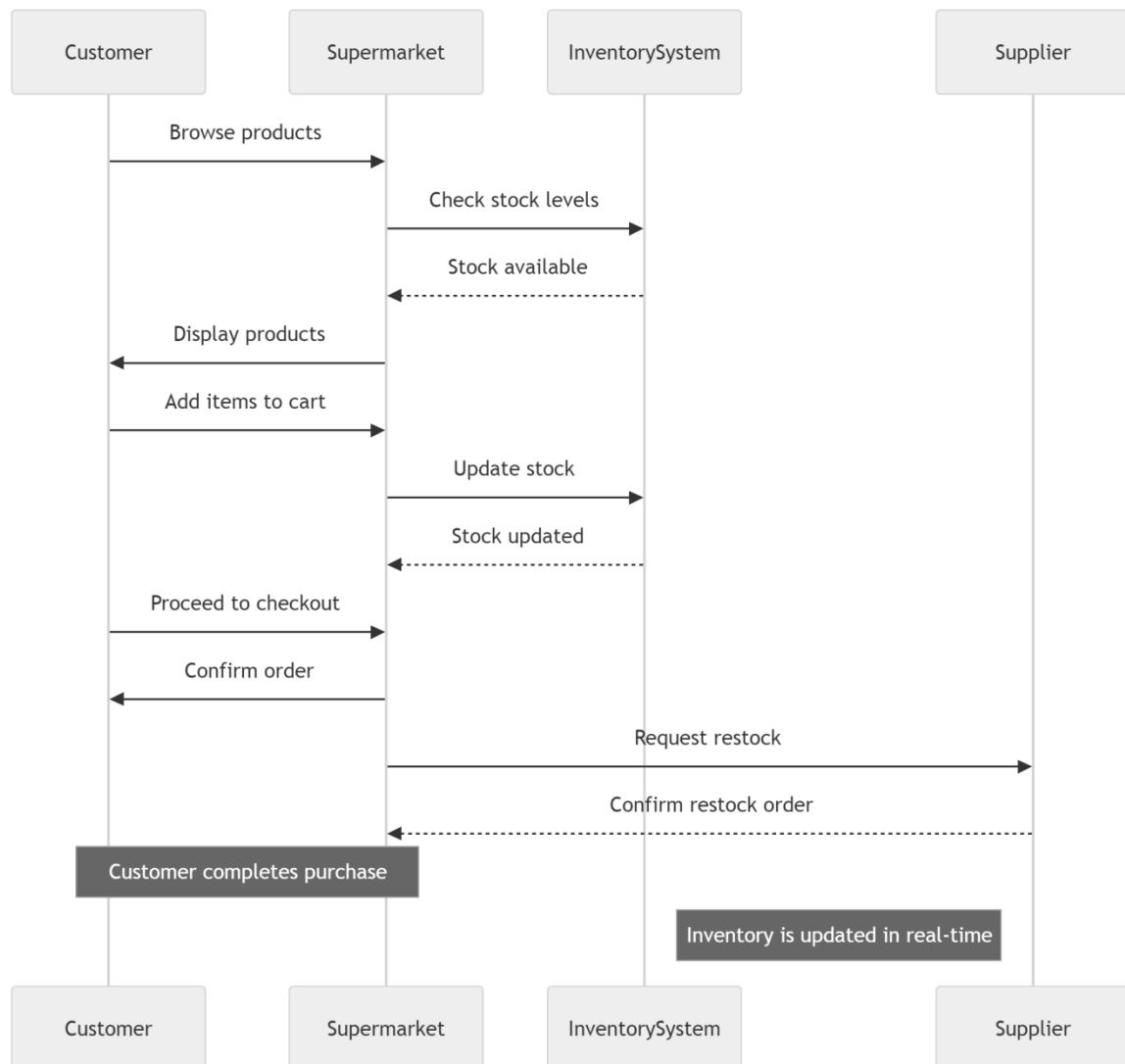
5.2.1 Class Diagram



5.2.2 Activity Diagram



5.2.3 Collaboration Diagram



MODULE DESCRIPTION

1. User Authentication Module

- **Purpose :** To ensure secure login and restricted access to administrative functionalities.
- **Key Functions:**
 - ✧ Login verification using predefined credentials.
 - ✧ Session management using Flask sessions.
 - ✧ Redirects unauthorized users to the login page.
- **Interaction :** This module interacts with all other modules, ensuring that only authenticated users can perform operations like reorder approval or viewing inventory.

2. Inventory Management Module

- **Purpose :** To handle product stock details, manage entries, and display live stock levels.
- **Key Functions:**
 - ✧ Fetching product details such as name, category, quantity, expiry date, and current stock from the MySQL database.
 - ✧ Evaluating stock against defined thresholds (understock or overstock).
 - ✧ Updating stock levels as per sales or restocking.
- **Interaction :** This module feeds into the **Notification Module** and **Reorder Module** by flagging items needing attention. It also uses the database connection module to fetch and update data.

3. Notification Module (AWS SNS Integration)

- **Purpose :** To alert the admin or supplier via SMS or email in case of critical stock levels or order approvals.

➤ **Key Functions:**

- ✧ Send alerts if a product is understocked or overstocked.
- ✧ Trigger messages when products are near expiry.

- **Interaction:** Receives input from the inventory and reorder modules and connects with AWS SNS to send messages to the subscribed users or suppliers.

4. Reorder Management Module

- **Purpose :** To manage the lifecycle of reorder requests including approval, rejection, and notifications.

➤ **Key Functions:**

- ✧ List understocked products for admin review.
- ✧ Provide options to approve or reject reorders.
- ✧ Update reorder status in the database.

- **Interaction :** This module relies on the Inventory Module to detect understocked items and on the Notification Module to send alerts to the supplier.

5. Expiry Monitoring Module

- **Purpose :** To identify and flag products that are near or past their expiry date.

➤ **Key Functions:**

- ✧ Scan all products daily.
- ✧ Check expiry dates against the current date + threshold days (e.g., 7 days).
- ✧ Generate alerts if expiry is approaching.

- **Interaction :** This module interacts with the Notification Module to send out expiry alerts and updates the dashboard for the admin to take necessary actions.

6. Dashboard and Reporting Module

- **Purpose :** To provide an admin overview and generate reports.
- **Key Functions:**
 - ✧ Visual dashboard displaying current stock, understocked/overstocked items, and expiry alerts.
 - ✧ Real-time product display.
 - ✧ Access to reorder requests, status, and approval options.
 - ✧ Basic reporting on product movement.
- **Interaction :** Acts as a front-end interface to all modules and gathers data from the database and internal logic of the other modules.

7. Database Connection Module

- **Purpose :** To securely manage MySQL connections for all modules.
- **Key Functions:**
 - ✧ Establish and return database connections.
 - ✧ Handle connection errors gracefully.
 - ✧ Centralized function for DB operations across modules.
- **Interaction :** Used by every other module that interacts with the product or reorder tables.

8. Logout and Session Termination Module

- **Purpose :** To allow secure session ending for the user.
- **Key Functions:**
 - ✧ Clear user session.
 - ✧ Redirect to the login page.
- **Interaction :** Ensures that after logout, the admin cannot access other modules unless logged in again.

SYSTEM IMPLEMENTATION

7.1 Source Code

Python

```
from flask import Flask, render_template, request, redirect, url_for, session
import mysql.connector
import boto3
from datetime import datetime, timedelta

app = Flask(__name__)
app.secret_key = 'SKaviyask@25'

# Define stock level thresholds
UNDERSTOCK_THRESHOLD = 10
OVERSTOCK_THRESHOLD = 100
EXPIRY_ALERT_DAYS = 7

# AWS Configuration for SNS
AWS_REGION = "ap-southeast-1"
SNS_TOPIC_ARN = "arn:aws:sns:ap-southeast-1:051826731666:kaviya"

# Database Configuration (AWS RDS)
DB_HOST = "database-2.cdmqo2swyhou.ap-southeast-1.rds.amazonaws.com"
DB_USER = "admin"
DB_PASSWORD = "SKaviyask25"
DB_NAME = "kaviya"

# Connect to MySQL
def get_db_connection():
    try:
        conn = mysql.connector.connect(
            host=DB_HOST,
            user=DB_USER,
            password=DB_PASSWORD,
            database=DB_NAME
        )
        return conn
    except mysql.connector.Error as err:
        print(f"Database connection error: {err}")
        return None

# Check stock levels
def check_inventory():
    conn = get_db_connection()
```

```

if conn is None:
    return [], []

cursor = conn.cursor(dictionary=True)
query = "SELECT Product, Category, Price, Quantity, Totalstock, Expirydate FROM Products"
cursor.execute(query)
products = cursor.fetchall()
cursor.close()
conn.close()

stock_issues = []
expiry_issues = []
today = datetime.now().date()
alert_date = today + timedelta(days=EXPIRY_ALERT_DAYS)

for product in products:
    if product["Totalstock"] < UNDERSTOCK_THRESHOLD or product["Totalstock"] >
OVERSTOCK_THRESHOLD:
        stock_issues.append(product)
    if product["Expirydate"] and product["Expirydate"] <= alert_date:
        expiry_issues.append(product)

return stock_issues, expiry_issues

# Send SMS Notification using AWS SNS
def send_sms(message):
    try:
        sns_client = boto3.client("sns", region_name=AWS_REGION)
        response = sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message
        )
        print("Notification sent:", response)
    except Exception as e:
        print("Failed to send notification:", e)

# Notify stock issues and expiry alerts
def notify_issues():
    stock_issues, expiry_issues = check_inventory()
    for product in stock_issues:
        message = f"Stock Alert: {product['Product']} has {product['Totalstock']} units."
        if product["Totalstock"] < UNDERSTOCK_THRESHOLD:
            message += " It is UNDERSTOCKED."
        elif product["Totalstock"] > OVERSTOCK_THRESHOLD:
            message += " It is OVERSTOCKED."
        send_sms(message)

```

```

for product in expiry_issues:
    message = f"Expiry Alert: {product['Product']} will expire on {product['Expirydate']}."
    send_sms(message)

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

@app.route('/login', methods=['POST'])
def do_login():
    username = request.form['username']
    password = request.form['password']
    if username == 'kaviya' and password == 'SKaviya@25':
        session['logged_in'] = True
        return redirect(url_for('dashboard'))
    else:
        return "Invalid Credentials"

@app.route('/dashboard')
def dashboard():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    conn = get_db_connection()
    if conn is None:
        return "Database connection error. Please try again later."

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT COUNT(*) AS Product_id FROM Products ")
    Product_id = cursor.fetchone()["Product_id"]

    # Fetch low stock products (Threshold: 10)
    cursor.execute("SELECT COUNT(*) AS low_stock_count FROM Products WHERE totalstock < 10")
    low_stock_count = cursor.fetchone()["low_stock_count"]

    # Fetch reorder pending items
    cursor.execute("SELECT COUNT(*) AS reorder_pending FROM reorders WHERE status = 'Pending'")
    reorder_pending = cursor.fetchone()["reorder_pending"]

    return render_template('dashboard.html', Product_id=Product_id, low_stock_count = low_stock_count, reorder_pending=reorder_pending)

@app.route('/example')
def products():
    if not session.get('logged_in'):

```

```

        return redirect(url_for('login'))

    conn = get_db_connection()
    if conn is None:
        return "Database connection error. Please try again later."

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT Product, Category, Price, Quantity, Totalstock, Expirydate FROM
Products")
    products = cursor.fetchall()
    cursor.close()
    conn.close()

    return render_template('example.html', Products=products,
                           understock_threshold=UNDERSTOCK_THRESHOLD,
                           overstock_threshold=OVERSTOCK_THRESHOLD )

@app.route('/reorders')
def reorders():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    conn = get_db_connection()
    if conn is None:
        return "Database connection error. Please try again later."

    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT r.reorder_id, p.Product, p.totalstock, r.status, s.email
        FROM reorders r
        JOIN Products p ON r.product_id = p.Product_id
        JOIN suppliers s ON s.supplier_id = s.supplier_id
WHERE p.totalstock < 10 AND r.status = 'pending'
        """)
    reorders = cursor.fetchall()
    cursor.close()
    conn.close()

    return render_template('reorders.html', reorders=reorders)

@app.route('/approve_reorder/<int:reorder_id>', methods=['POST'])
def approve_reorder(reorder_id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    cursor.execute("SELECT p.name, r.quantity, s.email FROM reorders r JOIN products p ON
r.product_id = p.product_id JOIN suppliers s ON s.Supplier_id WHERE r.reorder_id = %s",

```

```

(reorder_id,))
    reorder = cursor.fetchone()

    if reorder:
        cursor.execute("UPDATE reorders SET status = 'Approved' WHERE reorder_id = %s",
(reorder_id,))
        conn.commit()

        subject = "Reorder Approved"
        message = f"Order approved for {reorder['quantity']} units of {reorder['name']}"
        send_sns_email(subject, message)

    cursor.close()
    conn.close()
    return redirect(url_for('reorders'))

@app.route('/reject_reorder/<int:reorder_id>', methods=['POST'])
def reject_reorder(reorder_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE reorders SET status = 'Rejected' WHERE reorder_id = %s",
(reorder_id,))
    conn.commit()
    cursor.close()
    conn.close()
    return redirect(url_for('reorders'))

@app.route('/logout')
def logout():
    session['logged_in'] = False
    return redirect(url_for('login'))

@app.route('/reports')
def reports():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    conn = get_db_connection()
    if conn is None:
        return "Database connection error. Please try again later."

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT COUNT(*) AS low_stock_count FROM Products WHERE totalstock
< 10")
    low_stock_count = cursor.fetchone()["low_stock_count"]

    cursor.execute("SELECT COUNT(*) AS expiring_soon FROM Products WHERE ExpiryDate

```

```

<= '7")
    expiring_soon = cursor.fetchone()["expiring_soon"]

    return render_template ('reports.html', low_stock_count = low_stock_count, expiring_soon =
expiring_soon)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

Login Page

```

<!DOCTYPE html>
<html>
<head>
    <title>Login Page</title>
<link rel="stylesheet" href="{{ url_for('static' , filename='style.css') }}">
</head>
<body style="background-color: sky-blue; display: flex; justify-content: center; align-items:
center; height: 100vh;">
<div class="login-container">
    <form action="/login" method="post">
        <h1>Login</h1>
        <br>
        <br>
        <B>Username :</B> <input type = "text" name = "username" placeholder = "Enter
Username">
        <br>
        <br>
        <B>Password :</B> <input type = "password" name = "password" placeholder = "Enter
Password">
        <br>
        <br>
        <button type="submit">Login</button>
    </form>
</div>

```

</body>

</html>

Dashboard Page

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dashboard</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

</head>

<body>

<!-- Navbar -->

<div class="navbar">

<h1>Smart Inventory Management System</h1>

Logout *

</div>

<!-- Sidebar -->

<div class="sidebar">

<h3 style="text-align: center;">Dashboard</h3>

<i class="fas fa-home"></i> Home

<i class="fas fa-boxes"></i> Products

<i class="fas fa-shopping-cart"></i> Reorders

<i class="fas fa-chart-line"></i> Reports

</div>

<!-- Main Content -->

<div class="content">

 <h2>Welcome to SK Supermarket</h2>

<div class="card-container">

 <div class="card">

 <i class="fas fa-box"></i>

 <h4>Total Products</h4>

 <h3>{{ Product_id }}</h3>

 </div>

<div class="card">

 <i class="fas fa-exclamation-triangle"></i>

 <h4>Low Stock</h4>

 <h3>{{ low_stock_count }}</h3>

</div>

<div class="card">

 <i class="fas fa-shopping-cart"></i>

 <h4>Reorders Pending</h4>

 <h3>{{ reorder_pending }}</h3>

</div>

<div class="card">

 <i class="fas fa-indian-rupee-sign"></i>

 <h4>Sales Today</h4>

 <p>500</p>

</div>

```
<br>
```

```
<br>
```

```
<br>
```

```
</div>
```

```
<button class="btn" onclick="window.location.href='/example'">
```

```
  <i class="fas fa-list"></i> View Supermarket Products List
```

```
</button>
```

```
</div>
```

```
</body>
```

```
</html>
```

Products Page

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Supermarket Stock</title>
```

```
  <linkrel="stylesheet" href = "https://cdnjs.cloudflare.com/ajax/libs/fontawesome/6.0.0/css/all.min.css">
```

```
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

```
</head>
```

```
<body>
```

```
  <h1><i class="fas fa-store"></i>Supermarket Products List<i class="fas fa-store"></i></h1>
```

```
  <br>
```

```
  <br>
```

```
  <table id="stockTable">
```

```
    <tr>
```

```
      <th>Product</th>
```

```
      <th>Category</th>
```

```
      <th>Price</th>
```

```
      <th>Quantity</th>
```

```

    <th>Total Stock</th>
    <th>Expiry Date</th>
    <th>Status</th>
  </tr>
  {% for Product in Products %}
  <tr>
    <td>{{ Product.Product }}</td>
    <td>{{ Product.Category }}</td>
    <td>{{ Product.Price }}</td>
    <td>{{ Product.Quantity }}</td>
    <td class="total-stock">{{ Product.Totalstock }}</td>
    <td>{{ Product.Expirydate }}</td>
    <td class="status-cell"></td> <!-- JavaScript will add color and text here -->
  </tr>
  {% endfor %}
</table>

```

```

<a href="/dashboard" class="back-btn"><i class="fas fa-arrow-left"></i> Back to Dashboard
</a>

```

```

<script>
document.addEventListener("DOMContentLoaded", function() {
  let overstockThreshold = {{ overstock_threshold }};
  let understockThreshold = {{ understock_threshold }};

  let rows = document.querySelectorAll("#stockTable tr");
  rows.forEach((row, index) => {
    if (index === 0) return; // Skip header row

    let stockCell = row.querySelector(".total-stock");
    let statusCell = row.querySelector(".status-cell");

```

```

    if (stockCell && statusCell) {
        let stockValue = parseInt(stockCell.textContent.trim());

        if (stockValue > overstockThreshold) {
            statusCell.classList.add("status-overstock");
            statusCell.textContent = "Overstock";
        } else if (stockValue < understockThreshold) {
            statusCell.classList.add("status-understock");
            statusCell.textContent = "Understock";
        } else {
            statusCell.classList.add("status-regular");
            statusCell.textContent = "Regular Stock";
        }
    }
});
});
</script>
</body>
</html>

```

Reorder page

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Reorder Requests</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome/6.0.0/css/all.min.css">
    <link rel="stylesheet" href="{ { url_for('static' , filename='style.css') } }">
</head>
<body>

<div class="navbar">
    <h1>Reorder Requests</h1>

```

</div>

<div class="container">

<h2>Pending Reorders</h2>

<table>

<thead>

<tr>

<th>Reorder ID</th>

<th>Product ID</th>

<th>Total Stock</th>

<th>Status</th>

<th>Actions</th>

</tr>

{% for reorder in reorders %}

<tr>

<td>{{ reorder.reorder_id }}</td>

<td>{{ reorder.product_id }}</td>

<td>{{ reorder.total_stock }}</td>

<td>{{ reorder.status }}</td>

<td>

<button class="btn approve" onclick="approveReorder ({{ reorder.reorder_id }})"> Approve

</button>

<button class="btn reject" onclick="rejectReorder({{ reorder.reorder_id }})">Reject</button>

</td>

</tr>

{% endfor %}

</thead>

<tbody id="reordersTable">

<!-- Data will be inserted dynamically -->

</tbody>

</table>

<i class="fas fa-arrow-left"></i> Back to Dashboard

</div>

```

<script>
function approveReorder(reorderId) {
    fetch('/approve_reorder/' + reorderId, { method: 'POST' })
        .then(response => location.reload());
}

```

```

function rejectReorder(reorderId) {
    fetch('/reject_reorder/' + reorderId, { method: 'POST' })
        .then(response => location.reload());
}

```

```

</script>

```

```

</body>

```

```

</html>

```

Report Page

```

<!DOCTYPE html>

```

```

<html lang="en">

```

```

<head>

```

```

    <meta charset="UTF-8">

```

```

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

```

    <title>Sales & Purchase Reports</title>

```

```

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

```

```

    <link rel="stylesheet" href="{ { url_for('static' , filename='style.css') } }">

```

```

</head>

```

```

<body>

```

```

<div class="navbar">

```

```

    <h1>Sales & Purchase Reports</h1>

```

```

</div>

```

```

<br>

```

```

<br>

```

```

<br>

```

```

<div class="container">

```

```

    <h2>Overall Sales and Purchase Summary</h2>

```

```

<div class="report-section">
  <div class="report-box">
    <h3><i class="fas fa-shopping-cart"></i> Total Sales</h3>
    <p><i class="fas fa-indian-rupee-sign"></i> 50,000</p>
  </div>
  <div class="report-box">
    <h3><i class="fas fa-truck"></i> Total Purchases</h3>
    <p><i class="fas fa-indian-rupee-sign"></i> 30,000</p>
  </div>
</div>

<div class="report-section">
  <div class="report-box">
    <h3><i class="fas fa-dollar-sign"></i> Profit</h3>
    <p><i class="fas fa-indian-rupee-sign"></i> 20,000</p>
  </div>
  <div class="report-box">
    <h3><i class="fas fa-exclamation-triangle"></i> Low Stock Items</h3>
    <h3>{{ low_stock_count }}</h3>
  </div>
</div>

<div class="report-section">
  <div class="report-box">
    <h3><i class="fas fa-exclamation-triangle"></i> Expiring Soon Items</h3>
    <h3>{{ expiring_soon }}</h3>
  </div>
</div>

<a href="/dashboard" class="back-btn"><i class="fas fa-arrow-left"></i> Back to Dashboard</a>
</div>

</body>
</html>

```

CSS Style

<style>

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 20;
    padding: 30;
}
form {
    display: inline-block;
    text-align: left;
}
input {
    width: 90%;
    padding: 10px;
    margin: 10px 0;
    border-radius: 10px;
    border-bottom: 2px solid #ccc;
    transition: border-bottom 0.3s;
}
h1 {
    text-align: center;
}
table {
    width: 100%;
    border-collapse: collapse;
    margin: 20px auto;
}
table, th, td {
    border: 1px solid #ddd;
}
th, td {
    padding: 12px;
```



```

        text-align: center;
        border-bottom: 1px solid #ddd;
    }
    th {
        background-color: #4CAF50;
        border-radius: 6px;
        color: #333;
    }
    .approve-btn, .reject-btn {
        padding: 8px 12px;
        border: none;
        color: white;
        cursor: pointer;
        border-radius: 5px;
    }
    .approve-btn {
        background-color: #28a745;
    }
    .reject-btn {
        background-color: #dc3545;
    }
    .approve-btn:hover {
        background-color: #218838;
    }
    .reject-btn:hover {
        background-color: #c82333;
    }
    tr:nth-child(even) {
        background-color: #87CEEB;
    }
    tr:hover {
        background-color: #f1f1f1;
    }
    /* Status column styling */
    .status-overstock {

```

```

padding: 1px;
border-radius: 90%;
background-color: #FF0000;
color: #b30021;
font-weight: bold;
}
.status-understock {
padding: 1px;
border-radius: 90%;
background-color: #FFFF00;
color: #00008B;
font-weight: bold;
}
.status-regular {
padding: 1px;
border-radius: 90%;
background-color: #86e49d;
color: #006b21;
font-weight: bold;
}
.back-btn {
display: inline-block;
margin-top: 20px;
padding: 10px 15px;
background: #4CAF50;
color: white;
text-decoration: none;
border-radius: 5px;
text-align: center;
width: 150px;
}
button {
background-color: #008000;
color: white;
padding: 10px;

```

```

        border-radius: 10px;
        cursor: pointer;
    }
    .login-container {
        background: white;
        padding: 40px;
        border-radius: 30px;
        box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
        text-align: center;
        width: 350px;
    }
    .navbar {
        background-color: #4CAF50;
        padding: 15px;
        color: white;
        border-radius: 60px;
        text-align: center;
        font-size: 20px;
        position: fixed;
        width: 100%;
        top: 0;
        left: 0;
    }
    .container {
        margin: 80px auto;
        width: 90%;
        max-width: 1000px;
        background: white;
        padding: 20px;
        border-radius: 8px;
        box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
        text-align: center;

```

```

    color: #333;
}

.report-section {
    display: flex;
    justify-content: space-around;
    margin-top: 20px;
}

.report-box {
    background: #4CAF50;
    color: white;
    padding: 20px;
    border-radius: 8px;
    text-align: center;
    width: 45%;
    box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.2);
}

.report-box h3 {
    margin: 0;
    font-size: 22px;
}

.report-box p {
    font-size: 18px;
    margin-top: 5px;
}

.back-btn:hover {
    background: #388E3C;
}

.navbar a {
    float: right;

```

```
    color: white;
    margin-right: 20px;
    font-size: 16px;
}
```

```
.sidebar {
    width: 250px;
    height: 100vh;
    background: #343a40;
    color: white;
    border-radius: 8px;
    position: fixed;
    top: 50px;
    left: 0;
    padding-top: 20px;
}
```

```
.sidebar a {
    display: block;
    color: white;
    padding: 15px;
    text-decoration: none;
    font-size: 18px;
}
```

```
.sidebar a:hover {
    background: #4CAF50;
}
```

```
.content {
    padding: 20px;
    text-align: center;
    margin-top: 70px;
}
```

```
.card-container {
```

```
        display: flex;
    flex-wrap: wrap;
    gap: 30px;
    justify-content: right; /* Centers the cards */
    align-items: right;
}
```

```
.card {
    background: white;
    width: 200px;
    padding: 15px;
    text-align: center;
    border-radius: 8px;
    box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.1);
    transition: transform 0.2s;
}
```

```
.card:hover {
    transform: scale(1.05);
}
```

```
.card i {
    font-size: 2rem;
    color: #4CAF50;
    margin-bottom: 10px;
}
```

```
.btn {
    background-color: #4CAF50;
    color: white;
    padding: 12px 20px;
    border-radius: 10px;
    cursor: pointer;
    font-size: 16px;
    margin-top: 20px;
    display: inline-block;
```

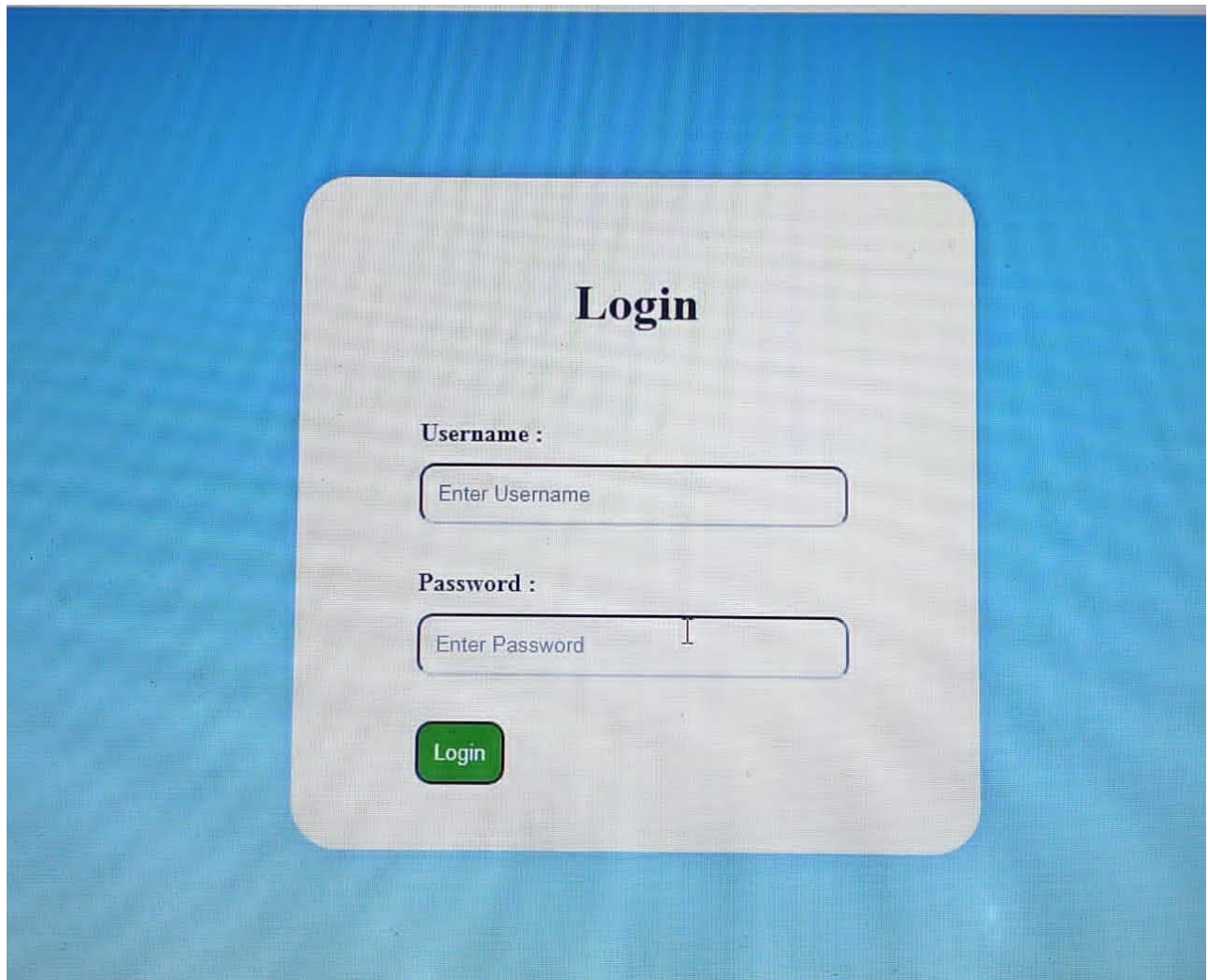
```
}

.btn:hover {
  background-color: #388E3C;
}

@media (max-width: 768px) {
  .sidebar {
    width: 100%;
    height: auto;
    position: relative;
  }
  .content {
    margin-left: 0;
    padding: 10px;
  }
  .card-container {
    flex-direction: column;
    align-items: center;
  }
</style>
```

7.2 Module Screen Shots

➤ Login page



The screenshot shows a login page with a blue background. In the center is a white rounded rectangle containing the login form. The form has the title "Login" at the top. Below it are two input fields: "Username :" and "Password :". The "Username :" field has a placeholder text "Enter Username". The "Password :" field has a placeholder text "Enter Password" and a cursor. Below the password field is a green "Login" button.

Login

Username :

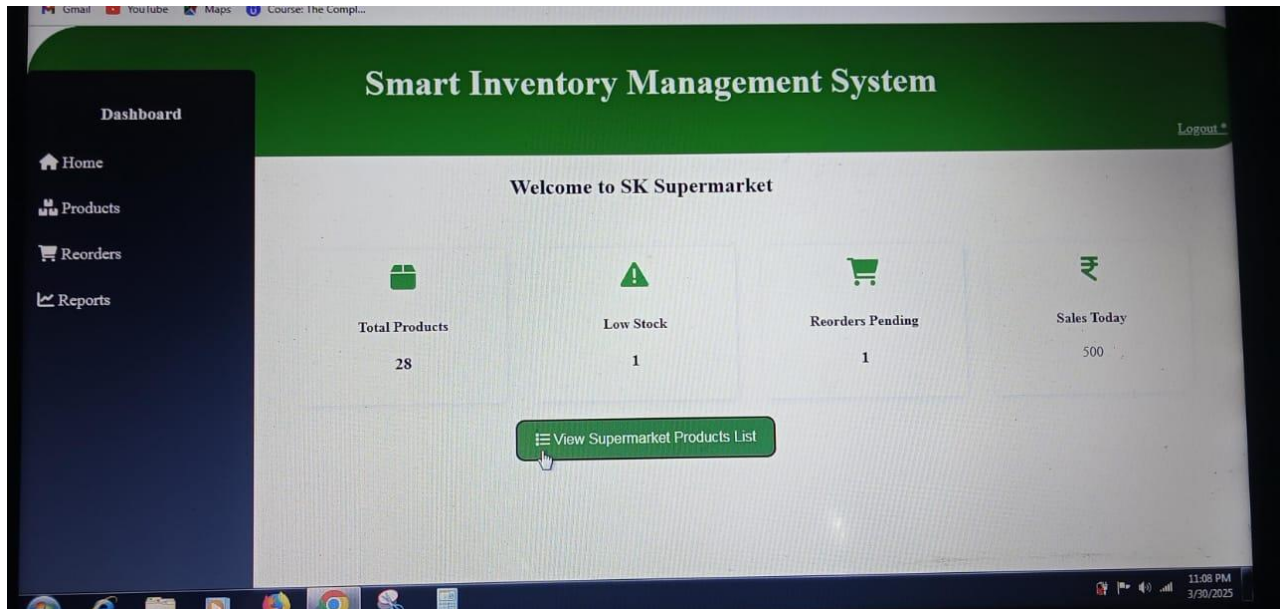
Enter Username

Password :

Enter Password

Login

➤ Dashboard page



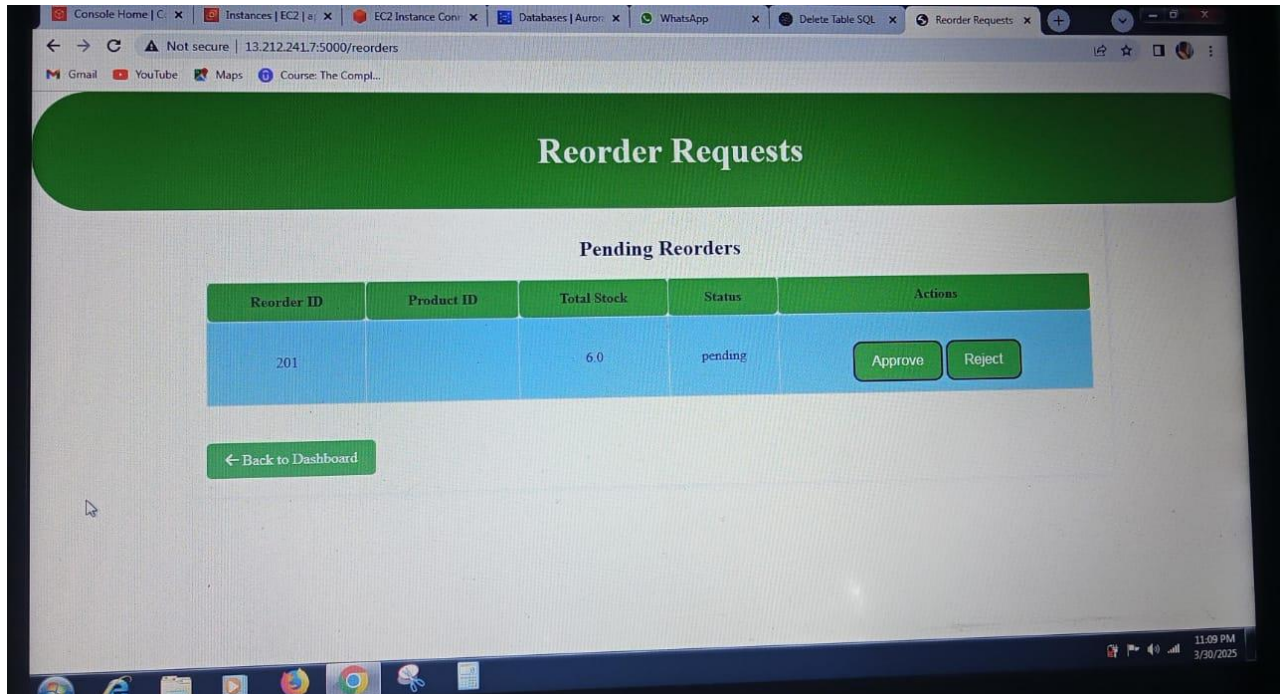
➤ Products Page

The screenshot displays the 'Supermarket Products List' page. It features a table with the following columns: Product, Category, Price, Quantity, Total Stock, Expiry Date, and Status. The table lists various dairy products, most of which are in 'Regular Stock', while 'Milk' is marked as 'Understock'.

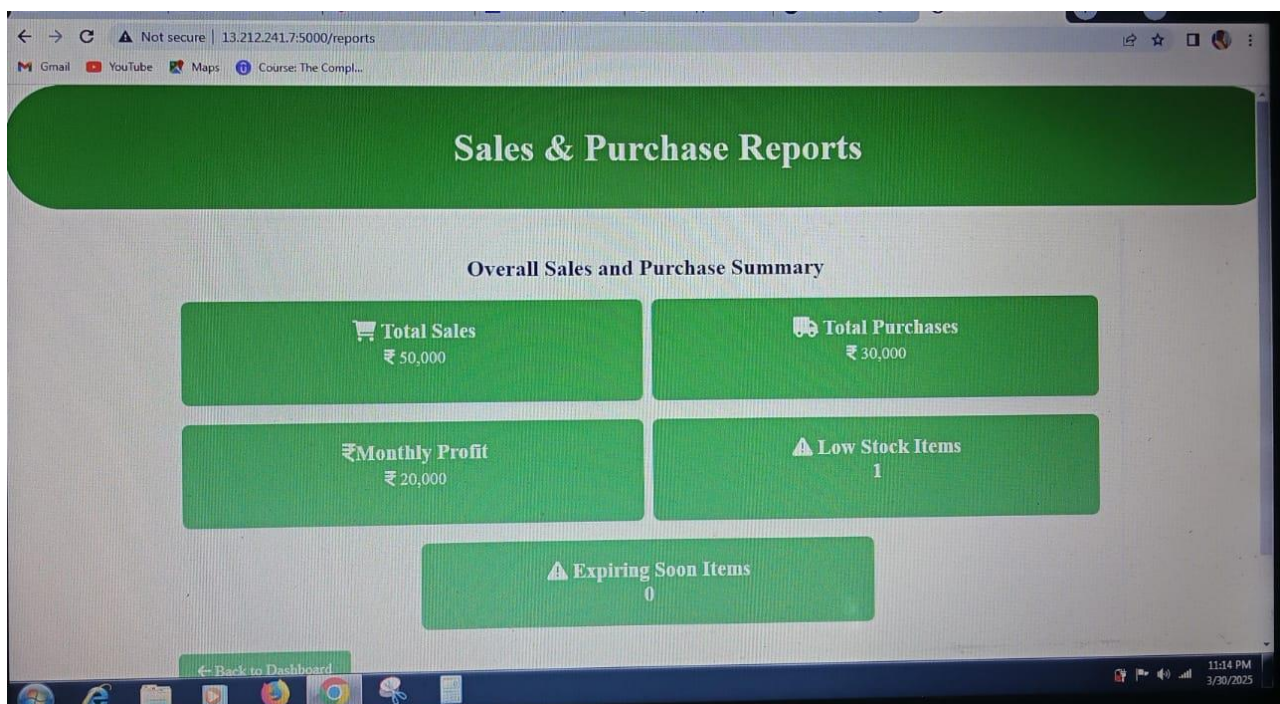
| Product | Category | Price | Quantity | Total Stock | Expiry Date | Status |
|----------------|----------|--------|----------|-------------|-------------|---------------|
| Cheese | Dairy | 125.00 | 250 g | 70.0 | 2025-04-10 | Regular Stock |
| Yogurt | Dairy | 50.00 | 250g | 80.0 | 2025-04-20 | Regular Stock |
| Butter | Dairy | 300.00 | 500g | 35.0 | 2025-04-25 | Regular Stock |
| Cream | Dairy | 80.00 | 250g | 50.0 | 2025-04-20 | Regular Stock |
| Ice Cream | Dairy | 195.00 | 1L | 60.0 | 2025-06-15 | Regular Stock |
| condensed Milk | Dairy | 130.00 | 1kg | 40.0 | 2026-01-01 | Regular Stock |
| Whipping Cream | Dairy | 190.00 | 1L | 30.0 | 2025-05-10 | Regular Stock |
| Ghee | Dairy | 125.00 | 250g | 30.0 | 2026-02-01 | Regular Stock |
| Butter Milk | Dairy | 50.00 | 1L | 50.0 | 2025-04-22 | Regular Stock |
| Milk | Dairy | 40.00 | 1L | 6.0 | 2025-04-20 | Understock |

The browser's taskbar at the bottom shows the system clock indicating 11:09 PM on 3/30/2025.

➤ Reorder Page



➤ Report page



TESTING

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy and usability. It mainly aims at measuring specification, functionality and performance of a software program or application. Software testing can be stated as the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by its design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases. Both verification and validation are necessary, but different components of any testing activity.

Testing Methods

1. Black Box Testing

The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without concerning with the internal logical structure of the software is known as black box testing. Black Box testing treats an application looks at the outputs that are produced by specific inputs into the application. The Black box tester does not need to understand why the code does what it does, and they should not have access to the source code of the application. Requirements are used to determine the correct outputs of black box testing, and these test cases are used to validate that the right software is being built.

Black Box Testing is testing the software without any knowledge of the inner working, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document. Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other. It is a testing in which the software under test is treated, as a black box. You cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

2. White-Box Testing

The technique of testing in which the tester is aware of the internal workings of the product, have access to its source code and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing. White Box Testing is tests that are run an application with the knowledge of the internal working of the code base. White box test cases should test different paths, decision points (both true and false decisions), execute loops, and check internal data structures of the application. The goal of white box testing is to cover testing as many of the statements, decision point, and branches in the code base as possible. White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from black box level.

Functional Testing

Function tests provide a systematic demonstration that functions tested are available as specified by the business and technical requirements, system documentation and user manuals. Functional testing is centered on the following items:

- ✧ Valid Input: Identified classes of valid input must be accepted.
- ✧ Invalid Input: Identified classes of invalid input must be rejected.
- ✧ Functions: Identified functions must be exercised.
- ✧ Output: Identified classes of application outputs must be exercised
- ✧ System/Procedures: Inter facing systems or procedures must be invoked.

1. Unit Testing

Testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed. It focuses on smallest unit of software design. It is often done by programmer by using sample input and observing its corresponding outputs. This testing performed on each module or block of code. Unit testing involves the design of test cases that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration.

This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests performs basic tests at component level and test a specific business process, application and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

2. Integration Testing

Testing process where individual units are combined and tested as a group. Integration testing is testing in which a group of components are combined to produce output. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model. Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

3. System Testing

Testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. In this software is tested such that it works fine for different operating system. It is covered under the black box testing technique. In this we just focus on required input and output without focusing on internal working. Testing is done by a professional testing agent on the completed software product before it is introduced to the market.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components sub assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement. System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable result.

4. Performance Testing

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test speed and effectiveness of program. Performance test was conducted to identify the bottlenecks involved in the system and to evolve the amount of execution time spent on various part of the unit. The response time of the activities performed by the system is verified and is found to be reasonable.

CONCLUSION AND FUTURE ENHANCEMENT

Conclusion

The **Smart Inventory Management System (SIMS)** has been developed to address the core challenges of traditional inventory handling in supermarkets, such as manual stock tracking, delayed reorder processes, and lack of real-time notifications. By leveraging modern technologies such as **Flask (Python), MySQL, and AWS Services (especially SNS)**, the system offers a fully functional and efficient solution that automates stock monitoring and ensures timely communication with suppliers and administrators.

This system provides **real-time stock updates**, tracks **expiry dates**, and automates **reorder approvals**, significantly reducing human error and improving operational efficiency. The integration of **AWS Simple Notification Service (SNS)** plays a crucial role in instantly alerting suppliers about understocked or overstocked items and notifying them upon reorder approvals. This ensures smooth stock replenishment and helps avoid product shortages or excesses. Through its **modular design**, the application ensures scalability and ease of maintenance. The admin-friendly **dashboard** offers a clear overview of current inventory status, pending reorders, and alerts related to stock and expiry, allowing the admin to make informed decisions quickly.

Future Enhancement

- Barcode/QR Code Integration : For automatic product scanning and faster inventory updates.
- Mobile Application Support : Android/iOS app for remote monitoring and reorder approvals.
- Predictive Analytics with Machine Learning : Forecast product demand and avoid overstock/understock situations.
- Supplier Management Module : Maintain supplier profiles, reorder history, and product pricing.
- Role-Based Access Control : Provide different access levels (Admin, Manager, Staff).
- Email Notifications : Send order updates and alerts to suppliers via email.
- Bulk Import/Export of Inventory : Upload or download product data using Excel or CSV files.
- Cloud Database Backup : Regular backups to prevent data loss using AWS services.
- POS System Integration : Sync inventory with point-of-sale for real-time stock updates.

REFERENCES

- Amazon Web Services (AWS) : <https://aws.amazon.com/>
- MySQL Documentation: <https://dev.mysql.com/doc/>
- Flask Web Framework : <https://flask.palletsprojects.com/>
- UML Diagram Guide : <https://diagrammingai.com/>
- Python : <https://www.hackerrank.com/>
- HTML, CSS, JS : <https://w3schools.com/html>