Inventory Management System For Retailers

Team ID: PNT2022TMID53428

Bachelor of Engineering

Computer Science and Engineering

Sri Venkateswara College of Engineering (SVCE)

Chennai - 602 117

Industry Mentor : Vasudeva hanush

Faculty Mentor : Dr N. Balamurugan

Team Members :

2127190501126 - Shri Rajamanikandan V

2127190501149- Yokesh Raj

2127190501147- Vishal R

2127190501138- Suriya S S

1. INTRODUCTION

1.1 Project Overview

Retail inventory management involves making sure you have the right amount of inventory, neither too little nor too much of the goods that customers want to buy. Retailers may meet client demand without running out of product by properly managing their inventory.

Effective retail inventory management reduces costs and improves knowledge of sales patterns in practice. Tools and techniques for retail inventory management provide merchants with more data on which to run their businesses. Applications have been created to assist shops in keeping track of and managing the supply of their own products. Retailers will be prompted by the System to register their accounts by providing necessary information. Retailers can log into the programme to access their accounts.Once retailers have successfully logged in to the programme, they can update the information on their inventory. Users can also add new goods by providing the necessary information regarding the item. They have access to the current inventory's specifics. If there is no stock found in the accounts of the retailers, the System will automatically send an email alert to them. In order for them to order new stock.

1.2 Purpose

The goal of retail inventory management is to maintain the right amount of desired product in stock. Retailers may meet client demand without running out of product by properly managing their inventory.

2. LITERATURE SURVEY

2.1 Existing problem

At an Engineering College, there is a student-run store called stationery shop that sells a range of items that students need, including safety boots, workshop jackets, stationery, and textbooks. It requires ongoing monitoring of stock-in and stock-out because it is a well-established store. The shop's inventory management, however, is not practical or effective enough to handle all the stock in the inventory because they continue to use a manual inventory system. This causes a number of issues, one of which is that the members of the shop always have to manually gather and calculate their stock at the start of each semester.

2.2 References

1.   A. Dahbi and H. T. Mouftah, "Supply chain efficient inventory management as a service offered by a cloud-based platform," *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1-7, doi: 10.1109/ICC.2016.7510722.
2.   https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0259284

3. W. Intayoad and P. Temdee, "Inventory cloud service for local SME: A scenario study for Ice cream factory," 2013 13th International Symposium on Communications and Information Technologies (ISCIT), 2013, pp. 741-745, doi: 10.1109/ISCIT.2013.6645952.

4. R. F. Olanrewaju, A. Irham Dollah and B. A. Ajayi, "Cloud-Based Inventory System for Effective Management of Under and Over-stock Hazards," 2021 8th International Conference on Computer and Communication Engineering (ICCCE), 2021, pp. 274-278, doi: 10.1109/ICCCE50029.2021.9467138.

2.3 Problem Statement Definition

Retail inventory management involves making sure you have the right amount of inventory that customers want to buy. Retailers may meet client demand without running out of product by properly managing their inventory.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

3.2 Ideation & Brainstorming

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕐 10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Akshara

Fine tune forecasting	Track stock levels	Inventory accounting
Use the FIFO approach	Forecast your inventory accurately	Create a centralized record of all products!

Samyuktha

Set up a point-of-sale technology	Use cloud based inventory management software	Decreases Inventory cost
Employ a safety stock inventory	Economic order quantity model	Physical Inventory Audit

Dhanavardhana

Audit stock levels	Identify low turn stock	Open-to-buy Inventory planning
Use the just in time inventory management	Batch Tracking	Spot Checking

Saranya

Implement a tracking system	Hire a stock controller	Minimizes out of stock
Use data and analytics	Integrate with mobile technology	Retail Inventory Method

3

Group Ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

⏱ 20 minutes

Tracking

Track stock levels	Imlpement a tracking system	Minimizes out of stock	Identify low turn stock	Batch Tracking

TIP

Add customisable tags to sticky notes to make it easier to find, browse, organise, and categorise important ideas as themes within your mural.

Audit

Audit stock levels	Physical Inventory Audit	Decreases Inventory cost	Inventory accounting

Maintaining Record

Create a centralized record of all products	Forecast your Inventory accurately	Employ a safety stock Inventory	Retail Inventory Method	Use data and analytics	Fine tune forecasting	Spot Checking

Management

Use the FIFO approach	Use the just in time inventory management	Open-to-buy Inventory planning	Set up a point-of-sale technology	Use cloud based inventory management software	Economic order quantity model

Hire a stock controller	Integrate with mobile technology

5

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⏱ 20 minutes

Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Batch Tracking

Use the FIFO approach

Track stock levels

Retail Inventory Method

Economic order quantity model

Forecast your inventory accurately

Use cloud based inventory management software

Use the just in time inventory management

Open-to-buy inventory planning

Minimizes out of stock

Inventory accounting

Integrate with mobile technology

Employ a safety stock inventory

Identify low turn stock

Use data and analytics

Fine tune forecasting

Set up a point-of-sale technology

Audit stock levels

Implement a tracking system

Decreases inventory cost

Hire a stock controller

Physical Inventory Audit

Spot Checking

Create a centralized record of all products!

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

6

3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To develop an inventory management system to add new stocks, update the existing stocks, view the stock details and to notify retailers when there is out of stock.
2.	Idea / Solution description	The inventory management system allows the retailers to log in to the application to perform necessary actions. Once logged in, the user can add new stocks into the applications. In case there are some changes in the stock, the user can update it accordingly. Our application allows the user to view the stock details. The retailers will be notified if there is no stock in their accounts. Our application will keep the retailers about their stock details and help them to meet the customer demands accordingly.
3.	Novelty / Uniqueness	As this application is hosted in the cloud,it can be shared among multiple retailers to maintain and manage their inventory at the same time.Also this application has no downtime unlike the web servers thus monitoring the retailer's stocks constantly and notifying them when needed.
4.	Social Impact / Customer Satisfaction	Our application will be very useful for retailers in their business. It helps them to keep track of their stocks which almost reduce their workload. The retailers do not need to remember all the stock details as our application notifies them regularly. Retailers do not need to appoint a person to manage the stock details. By using our application their money, time and workload will be reduced.

5.	Business Model (Revenue Model)	As retailers will be informed about the out of stock, it will be helpful for them to add new stocks. By notifying the retailers, they can prevent the loss which might occur due to out of stock. Retailers will be very much satisfied when they do not lose any of their customers.
6.	Scalability of the Solution	This application can handle request from a large number of users at the same time thus making it highly scalable.It also allows users to add more stocks into the system without concerning about storage.

3.4 Problem Solution fit

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) `CS`

Who is your customer?
i.e. working parents of 0-5 m.o. kids

The customers of this project are retailers who would like to have their inventory managed automatically

6. CUSTOMER CONSTRAINTS `CC`

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, unreliable devices.

- Budget
- Reliabilty
- Network connection

5. AVAILABLE SOLUTIONS `AS`

Which solutions are available to the customers when they face the problem?

or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

Retailers can manage their inventory manually

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS `J&P`

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Alerting the retailer via email when there there is out of stock to meet the customer demands

Providing cost - effective scalable management of stocks with less maintainance

9. PROBLEM ROOT CAUSE `RC`

What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations.

Manually managing the stocks leds to human error and may fail to meet to customer demands as it is unscalable

7. BEHAVIOUR `BE`

What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

As the number of stocks increases, the number of staff managing the inventory can be increased to provide scalability.

Focus on J&P, tap into BE, understand RC

---

3. TRIGGERS `TR`

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

- Seeing other retailers move to a more efficient solution
- Learning that softwares are more economical and scalable than managing manually

4. EMOTIONS: BEFORE / AFTER `EM`

How do customers feel when they face a problem or a job and afterwards? i.e. fed, insecure > confident, in control - use it in your communication strategy & devise.

Before :
- Retailers might be frustrated that the existing manual method incurs more cost in terms of time and effort and also worried the human prone errors
- Retailers might not be able to completely focus on the business growth

After :
- Retailers need no staff for stock management which saves money and thus makes them happy
- No human prone errors and retailers alerted where is there is out of stock making them focus on the main business functions and thus making retailers more confident about the success of their business

10. YOUR SOLUTION `SL`

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

- A cloud hoster inventory management system which is highly scalable and can be accessed by the retailers to manage their stock effectively
- Retailers need to create an account in the system and then can login to update their inventory details.
- Allso users will be able to add new stock by submitting essential details related to the stock.
- They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.

8. CHANNELS of BEHAVIOUR `CH`

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

- To manage the stocks,retailers can hire staff by using the social media for advertising
- Can search for effective ideas/techniques to manage the stocks and to reduce human error in management

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

- Can hire more staff by using posters for advertising
- Can get help from other retailers nearby to effectively manage more stocks without incurring much cost and maintainance

9

## 4. REQUIREMENT ANALYSIS

4.1 Functional requirement

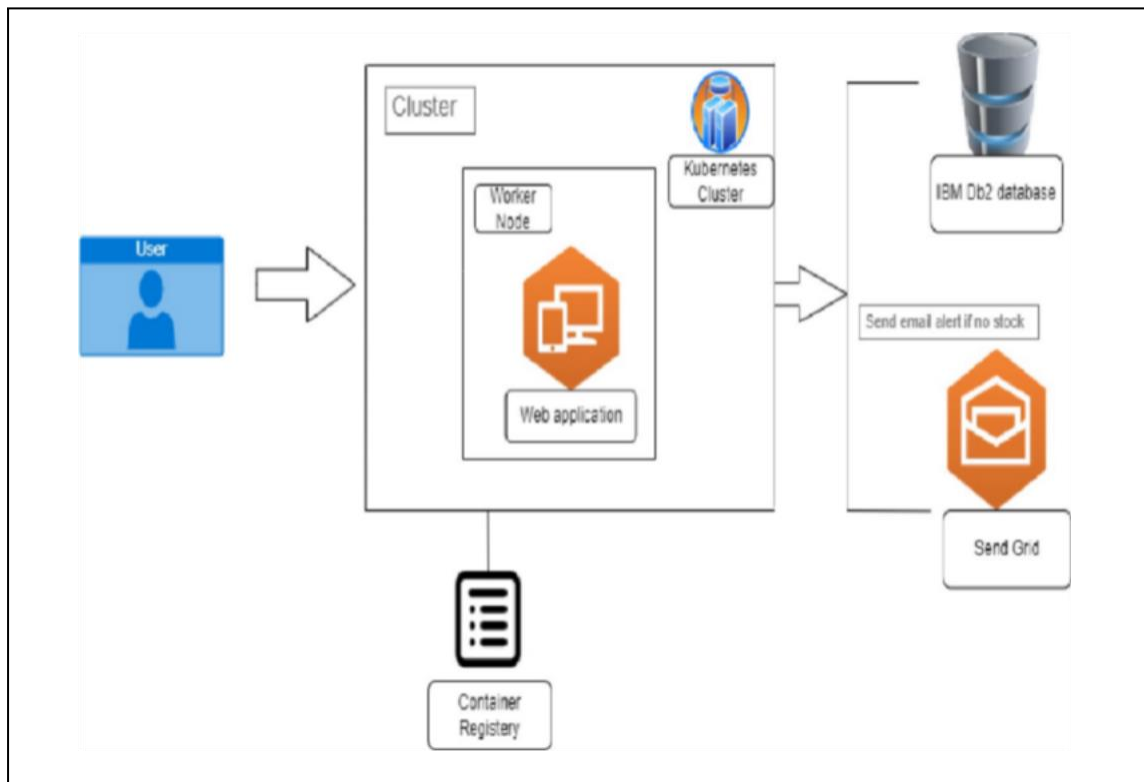| FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Successful Log in	Notification through Email
FR-4	Update inventory details	Notification through Email
FR-5	Add new stock	Notification through Email
FR-6	Unavailability of stock	Alert notification through Email

4.2 Non-Functional requirements

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Easy to use and learn
NFR-2	Security	Customer data integrity
NFR-3	Reliability	Reliable transport of messages
NFR-4	Performance	Less response time
NFR-5	Availability	No downtime
NFR-6	Scalability	Can handle large no of customers at the same time

5. PROJECT DESIGN

5.1 Data Flow Diagrams
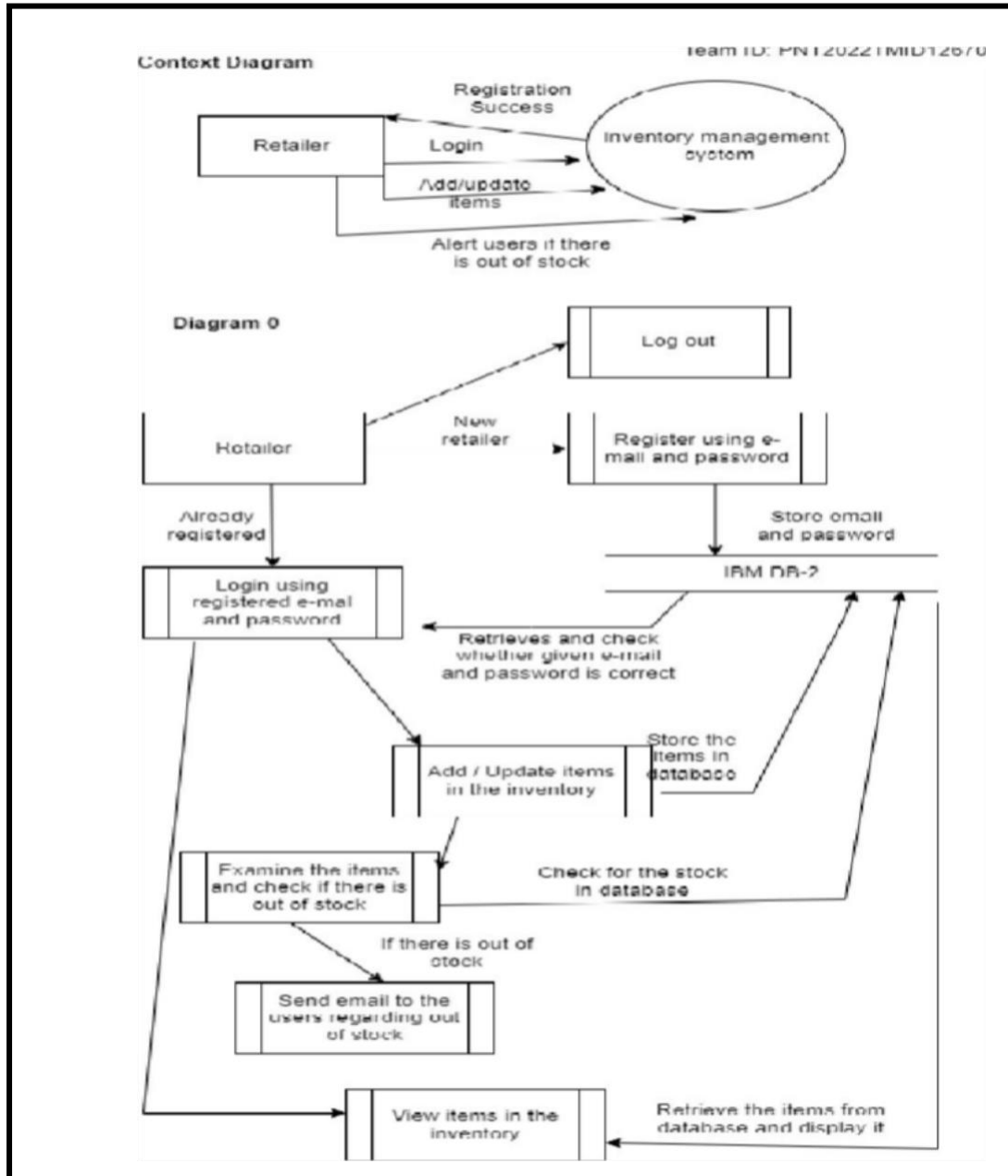
5.2 Solution & Technical Architecture

5.3 User Stories

| User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
	Login	USN-2	As a user, I can log into the application by entering email & password	I can access my account / dashboard	High	Sprint-1
	Add/delete items in the inventory	USN-3	As a user, I can add or delete items in the inventory	I can store the stock items and its details in the database	High	Sprint-2
	View items	USN-4	As a user, I can view the items in the inventory	I can view items which is stored in database	Medium	Sprint-2
	Update items	USN-5	As a user, I can update the existing items details	The database will be updated	High	Sprint-2
	Alert users	USN-6	As a user, I can alert the users if there is a out of stock or low stock	I can refill the stock details	Medium	Sprint-3
	Log out	USN-7	As a user, I can log out the web application	I will be redirected to login page again once I log out	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	4
Sprint-1	Login	USN-2	As a user, I can log into the application by entering email & password	1	High	4
Sprint-2	Add/delete items in the inventory	USN-3	As a user, I can add or delete items in the inventory	2	High	4
Sprint-2	View items	USN-4	As a user, I can view the items in the inventory	1	Medium	4
Sprint-3	Update items	USN-5	As a user, I can update the existing items details	2	High	4
Sprint-4	Alert users	USN-6	As a user, I can alert the users if there is a out of stock or low stock	1	Medium	4
Sprint-1	Log out	USN-7	As a user, I can log out the web application	1	High	4

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	7	6 Days	24 Oct 2022	29 Oct 2022	7	29 Oct 2022
Sprint-2	9	6 Days	31 Oct 2022	05 Nov 2022	9	05 Nov 2022
Sprint-3	5	6 Days	07 Nov 2022	12 Nov 2022	5	12 Nov 2022
Sprint-4	10	6 Days	14 Nov 2022	19 Nov 2022	10	19 Nov 2022

6.3 Reports from JIRA
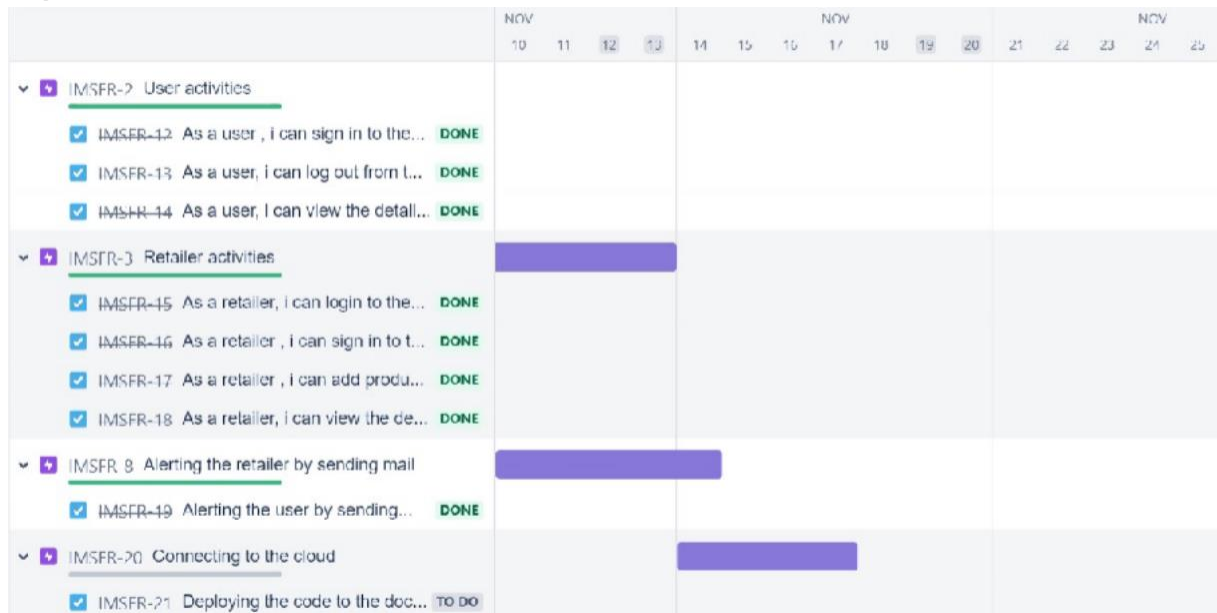
7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Login

Users are able to login to the system by filling in the details like their username and password, if the entered details are correct it will leave the message like ' Logged in successfully' ; if not it will leave the message like ' incorrect username / password'.

```python
@app.route('/')
def hello():
    return render_template('home.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ''
    if request.method == 'POST' :
        username = request.form['username']
        pd = request.form['password']
        sql = "SELECT * FROM people WHERE username =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,pd)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            session['loggedin'] = True
            session['id'] = account['USERNAME']
            userid=  account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

            return render_template('dashboard.html', msg = msg)
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg = msg)
```

7.2 Sign up

Users can sign up into the system by entering their details. If the account already exists it will leave the message like 'Account already exists' / ' Enter the valid email address' / 'name must contain only characters and numbers'. If users entered the correct details then it will send the message like ' You are successfully registered click signin !'.

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    mg=''
    if request.method == "POST":
        username=request.form['name']
        email=request.form['email']
        pw=request.form['password']
        address=request.form['address']
        phone=request.form['phone']
        sql='SELECT * FROM people WHERE email =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,email)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg='Account already exits!!'

        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            mg='Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms='name must contain only character and number'
        else:
            insert_sql='INSERT INTO people VALUES (?,?,?,?,?,?)'
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,username)
            ibm_db.bind_param(pstmt,2,address)
            ibm_db.bind_param(pstmt,3,phone)
            ibm_db.bind_param(pstmt,4,email)
            ibm_db.bind_param(pstmt,5,pw)
            ibm_db.execute(pstmt)
            mg='You have successfully registered click signin!!
            return render_template("login.html")
```

```python
    else:
        msg="fill out the form first!"
        return render_template('signup.html',meg=mg)
```

7.3 Log out

Users can log out from the system by clicking the log out option, it will send the message like ' You are now logged out , success '.

```python
@app.route('/logout')
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return render_template("home.html")
```

7.4 Add stock

Users are able to insert the products into the system by using the Add stock function. After adding the products it will send the message like ' You have successfully added the products'. If the product already exists it will send the message like ' Product already exists !! '.

```python
def add_stock():
    mg=''
    if request.method == "POST":
        prodname=request.form['prodname']
        quantity=request.form['quantity']
        warehouse_location=request.form['warehouse_location']
        sql='SELECT * FROM product WHERE prodname =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,prodname)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg='Product already exits!!'
        else:
            insert_sql='INSERT INTO product VALUES (?,?,?)'
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,prodname)
            ibm_db.bind_param(pstmt,2,quantity)
            ibm_db.bind_param(pstmt,3,warehouse_location)
            ibm_db.execute(pstmt)
            mg='You have successfully added the products!!'
            return render_template("dashboard.html")

    else:
        msg="fill out the form first!"
    return render_template('add_stock.html',neg=mg)
```

16

7.5 Delete stock

Users can delete the products from the system using the Delete stock function. After deleting the products it will send the message like ' Product deleted, Success'.

```python
@app.route('/delete_stock',methods=['GET','POST'])
def delete_stock():
    if(request.method=="POST"):
        prodname=request.form['prodname']
        sql2="DELETE FROM product WHERE prodname=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,prodname)
        ibm_db.execute(stmt2)


        flash("Product Deleted", "success")


        return render_template("dashboard.html")
```

7.6 Update stock

Users can Update the details of the products from the system using the update stock function. After updating the products it will send the message like ' You have Successfully updated the products! !'.

```python
def update_stock():
    mg=''
    if request.method == "POST":
        prodname=request.form['prodname']
        quantity=request.form['quantity']
        quantity=int(quantity)
        print(quantity)
        print(type(quantity))
        warehouse_location=request.form['warehouse_location']
        sql='SELECT * FROM product WHERE prodname =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,prodname)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)


        if acnt:
            insert_sql='UPDATE product SET  quantity=?,warehouse location=? WHERE prodname=? '
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,quantity)
            ibm_db.bind_param(pstmt,2,warehouse_location)
            ibm_db.bind_param(pstmt,3,prodname)
            ibm_db.execute(pstmt)
            mg='You have successfully updated the products!!'
            limit=5
            print(type(limit))
            if(quantity<=limit):
                alert("Please update the quantity of the product {}, Atleast {} number of pieces must be added!".format(prodname,10))
            return render_template("dashboard.html",meg=mg)
```

```
    else:
        msg-'Product not found!!'


else:
    msg-"fill out the form first!"
    return render_template('update_stock.html',msg=msg)
```

7.7 View stock

Users can View the products in the system using the View stock function. If the product is not found in the system it will send the message like ' No products found'.

```
def view_stock():

    sql = "SELECT * FROM product"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)
    print(result)

    products=[]
    row = ibm_db.fetch_assoc(stmt)
    print(row)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt)
        print(row)
    products=tuple(products)
    print(products)

    if result>0:
        return render_template('view.html', products = products)
    else:
        msg='No products found'
        return render_template('view.html', msg=msg)
```

7.8 Send grid

An account was created in sendgrid and apikey was generated for smtp connection. Using the api key generated, we accessed the smtp port with the help of python code and send an alert email. An alert email will be sent if the product quantity is less than 5 to the retailers.

```python
import smtplib
import os
from dotenv import load_dotenv
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase

def alert(main_msg):
    mail_from = '19z311@psgtech.ac.in'
    mail_to = 'dhanavandhanakannan3@gmail.com'
    msg = MIMEMultipart()
    msg['From'] = mail_from
    msg['To'] = mail_to
    msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
    mail_body = main_msg
    msg.attach(MIMEText(mail_body))
    print(msg)

    try:
        server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)
        server.ehlo()
        server.login('apikey',os.environ.get('SENDGRID_API_KEY'))
        server.sendmail(mail_from, mail_to, msg.as_string())
        server.close()
        print("Mail sent successfully!")
    except:
        print("Some Issue, Mail not Sent :(")
```

7.9 Database Schema

Table 1 -people
- This people table used to store users details while registering
- Used to store details of users while login
- This table contains users details so it is used to verify the user credentials

Table 2- product
- This product table used to store the details of the product present in the stock
- This table is used to know the amount of product present in the stock

8. TESTING

8.1 Test Cases

	Test Scenarios
	Login
1	Verify user is able to see login page
2	Verify user is able to loginto application or not?
3	Verify login page elements
	Register
1	Verify if user is able to enter all the details and register
2	Verify if user is redirected to login page once registered.
	Add products
1	Verify user is able to add products in ad product page
2	Verify whether added products are correctly added in the table
	View products
1	Verify whether products can be viewed in view page
2	Verify whether products can be retrieved correctly from database
	Update Products
1	Verify user is able to update products in update page
2	Verify whether updated product details are correctly updated in the table
3	Verify if the product quantity is less than 5
4	Verify if an alert email has been sent to retailer if the product quantity is less than 5
	Delete Products
1	Verify user is able to delete product movements
2	Verify whether deleted product gets deleted from the table

8.2 User Acceptance Testing

The project has been tested extensively with a number of users. The users found the interface very easy to use. The Web pages were colorful and attractive. There were no unnecessary details in the web page. It was clean and simple that any new user could master. The data input format was also simple.The user can login to the system by entering username and password and they can add stock,delete stock,update stock and view stock. Various inputs have been given by the users to test the consistency of the model. The model proved itself and all the users accepted the model as reliable and convenient.

9. RESULTS

9.1 Performance Metrics

The application that we have developed has better performance in speed.An alert email will be sent if the product quantity is less than 5 to the retailers.. The application developed also performed well with no glitches or lag found during the testing phase.

10. ADVANTAGES & DISADVANTAGES

Advantages:

- It helps to maintain the right amount of stocks
- It leads to a more organized warehouse
- It saves time and money
- A well-structured inventory management system leads to improved customer retention

Disadvantages:

- Increased space is need to hold the inventory
- Complexity
- High implementation costs

11. CONCLUSION

It is obvious that the manual inventory system needs to be replaced with an automated one that can shorten the time required, minimize human error, and increase inventory efficiency. The software of choice is also able to meet the specified demands intended to address the issue. The system can be made better in the future by including a place to store receipts and generate receipts for each transaction. Additionally, a profile area for each user may be created to store additional information about the users. Lastly, there is a feature that can tell the user when an item is getting close to its expiration date.

12. FUTURE SCOPE

There will be even more technological advancements in the field of inventories. To boost sales and draw customers, innovations ranging from artificial intelligence to inventory-free businesses are always being produced.

13. APPENDIX

Source Code

Backend code:

app.py:

```python
from flask import Flask, redirect, url_for, request, render_template, session, flash
import sqlite3 as sql
import re
import ibm_db
from sendgrid import *
conn
    =ibm_db.connect("DATABASE=bludb;HOSTNAME=ba99a9e6-d59e-4883-8fc0-d6a8c9f
    7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=31321;SECURITY=SS
    L;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=xwz94276;PWD=0HI7kIeAaMG
    SpL03","","")
app  =  Flask(  name  )
app.secret_key="dhana"


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


@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ''
    if request.method == 'POST' :
        username = request.form['username']
        pd = request.form['password']
        sql = "SELECT * FROM people WHERE username =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,pd)
        ibm_db.execute(stmt)
```

```python
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            session['loggedin'] = True
            session['id'] = account['USERNAME']
            userid=  account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

            return render_template('dashboard.html', msg = msg)
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg = msg)


@app.route('/signup', methods=['GET', 'POST'])
def signup():
    mg=''
    if request.method == "POST":
        username=request.form['name']
        email=request.form['email']
        pw=request.form['password']
        address=request.form['address']
        phone=request.form['phone']
        sql='SELECT * FROM people WHERE email =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,email)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg='Account already exits!!'
```

```python
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            mg='Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms='name must contain only character and number'
        else:
            insert_sql='INSERT INTO people VALUES (?,?,?,?,?)'
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,username)
            ibm_db.bind_param(pstmt,2,address)
            ibm_db.bind_param(pstmt,3,phone)
            ibm_db.bind_param(pstmt,4,email)
            ibm_db.bind_param(pstmt,5,pw)
            ibm_db.execute(pstmt)
            mg='You have successfully registered click signin!!'
            return render_template("login.html")


    else:
        msg="fill out the form first!"
        return render_template('signup.html',meg=mg)


@app.route('/logout')
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return render_template("home.html")


@app.route('/add_stock',methods=['GET','POST'])
def add_stock():
    mg=''
    if request.method == "POST":
        prodname=request.form['prodname']
        quantity=request.form['quantity']
        warehouse_location=request.form['warehouse_location']
        sql='SELECT * FROM product WHERE prodname =?'
```

```python
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,prodname)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg='Product already exits!!'
        else:
            insert_sql='INSERT INTO product VALUES (?,?,?)'
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,prodname)
            ibm_db.bind_param(pstmt,2,quantity)
            ibm_db.bind_param(pstmt,3,warehouse_location)
            ibm_db.execute(pstmt)
            mg='You have successfully added the products!!'
            return render_template("dashboard.html")

    else:
        msg="fill out the form first!"
        return render_template('add_stock.html',meg=mg)

@app.route('/delete_stock',methods=['GET','POST'])
def delete_stock():
    if(request.method=="POST"):
        prodname=request.form['prodname']
        sql2="DELETE FROM product WHERE prodname=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,prodname)
        ibm_db.execute(stmt2)

        flash("Product Deleted", "success")

        return render_template("dashboard.html")
```

```python
@app.route('/update_stock',methods=['GET','POST'])
def update_stock():
    mg=''
    if request.method == "POST":
        prodname=request.form['prodname']
        quantity=request.form['quantity']
        quantity=int(quantity)
        print(quantity)
        print(type(quantity))
        warehouse_location=request.form['warehouse_location']
        sql='SELECT * FROM product WHERE prodname =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,prodname)
        ibm_db.execute(stmt)
        acnt=ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
                insert_sql='UPDATE product SET quantity=?,warehouse_location=? WHERE
    prodname=? '
            pstmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt,1,quantity)
            ibm_db.bind_param(pstmt,2,warehouse_location)
            ibm_db.bind_param(pstmt,3,prodname)
            ibm_db.execute(pstmt)
            mg='You have successfully updated the products!!'
            limit=5
            print(type(limit))
            if(quantity<=limit):
                    alert("Please update the quantity of the product {}, Atleast {} number of pieces
    must be added!".format(prodname,10))
            return render_template("dashboard.html",meg=mg)
```

```python
        else:
            mg='Product not found!!'


    else:
        msg="fill out the form first!"
        return render_template('update_stock.html',meg=msg)


@app.route('/view_stock')
def view_stock():

    sql = "SELECT * FROM product"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)
    print(result)

    products=[]
    row = ibm_db.fetch_assoc(stmt)
    print(row)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt)
        print(row)
    products=tuple(products)
    print(products)

    if result>0:
        return render_template('view.html', products = products)
    else:
        msg='No products found'
        return render_template('view.html', msg=msg)
```

```python
@app.route('/delete')
def delete():
    return render_template('delete_stock.html')


@app.route('/update')
def update():
    return render_template('update_stock.html')
if __name__ == "_main_":
    app.run(debug=True)
```

sendgrid.py:

```python
import smtplib
import os
from dotenv import load_dotenv
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase


def alert(main_msg):
    mail_from = '19z311@psgtech.ac.in'
    mail_to = 'dhanavandhanakannan3@gmail.com'
    msg = MIMEMultipart()
    msg['From'] = mail_from
    msg['To'] = mail_to
    msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
    mail_body = main_msg
    msg.attach(MIMEText(mail_body))
    print(msg)

    try:
        server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)
        server.ehlo()
        server.login('apikey',os.environ.get('SENDGRID_API_KEY'))
        server.sendmail(mail_from, mail_to, msg.as_string())
        server.close()
```

```python
        print("Mail sent successfully!")
    except:
        print("Some Issue, Mail not Sent :(")
```

Front end code:

Home.html:

```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 by Tufayel Ahmed</title>
    <!-- Bootstrap CSS -->
     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzK
gwra6" crossorigin="anonymous">

<link
rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css">
    <style>
        body {
            background-color: rgb(221, 221, 221);
            color: black;
        }
    </style>
</head>

<body>
    <div class="header text-center">
        <h1 class="my-5 mx-3">Welcome to Inventory Management System</h1>
```

```html
    </div>
    <div class="container text-center mb-5">

<button class="btn btn-primary" onclick="location.href='/login'"><i class="bi-door-open"></i>
Login</button>

<button class="btn btn-info text-white" onclick="location.href='/signup'">
<i class="bi-person-plus"></i>
Sign Up</button>
    </div>
    <div class="accordion m-3" id="accordionExample">
        <div class="accordion-item">
            <h2 class="accordion-header" id="headingOne">

<button
class="accordion-button"
type="button" data-bs-toggle="collapse"
 data-bs-target="#collapseOne"
 aria-expanded="true" aria-controls="collapseOne">
                Sign Up
    </button>
    </h2>

<div
id="collapseOne"
class="accordion-collapse collapse show"
 aria-labelledby="headingOne"
 data-bs-parent="#accordionExample">
            <div class="accordion-body">
                Register to use the inventory management system
            </div>
        </div>
    </div>
    <div class="accordion-item">
```

```html
        <h2 class="accordion-header" id="headingTwo">

<button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"

data-bs-target="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">
            Login
          </button>
        </h2>

<div id="collapseTwo" class="accordion-collapse collapse" aria-labelledby="headingTwo"
          data-bs-parent="#accordionExample">
          <div class="accordion-body">
            Login and access the inventory
          </div>
        </div>
      </div>

  </div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vl5D
kLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
</html>
```

Signup.html:

```html
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
```

```css
body, html
  { height:
  100%;
  font-family: Arial, Helvetica, sans-serif;
  background-color: #9f9da7;


}

* {
  box-sizing: border-box;
}

.bg-img {

   background-color: #9f9da7;

  /* Center and scale the image nicely */
  background-position: center;
  background-repeat: no-repeat;
  background-size: cover;
  position: relative;
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  min-width: 50%;
  min-height: 50%;
}

/* Add styles to the form container */
.container
  { position:
  absolute;right: 0;
```

```css
    margin: 0px;
    max-width: 500px;
    padding: 16px;
    vertical-align: center;
    background-color: white;
    width:500px;
    height: 600px;
    margin:auto;
    left:  70px;
    top: 10px;


}

.button {
  background-color:#00cccc;
  padding: 15px 25px;
  text-align: center;
  text-decoration: none;
  display: block;
  margin: auto;
  cursor: pointer;
  border-radius: 16px;
  background-color: white;
  color: black;
  width:190;
  border: 2px solid #00cccc;

}
.button2:hover
  { background-
  color:#00cccc;color: white;
}
```

```
/* Set a style for the submit button */



</style>
</head>
<body >
</br></br></br>
<div class="bg-img">

  <form class="container"  action="{{url_for('signup') }}" method="POST">
    <h1>Sign-up/Register</h1>
    <br>


    <label for="name" style="font-size: 24">Name:</label>
      <input  type="text"  id="name"  name="name"  placeholder="Enter name"  style="width:
100%;height: 30;padding: 9px;margin: 5px 0 22px 0;border: none;background: #f1f1f1;"><br>
    <label for="addr" style="font-size: 24">Address:</label>
        <input  type="text"  id="address"  name="address"  placeholder="Enter  Address"
style="width: 100%;height: 30;padding: 9px;margin: 5px 0 22px 0;border: none;background:
#f1f1f1;"><br>


    <label for="phone" style="font-size: 24">Phone:</label>
      <input type="text" id="phone" name="phone" placeholder="Enter Phone" style="width:
100%;height: 30;padding: 9px;margin: 5px 0 22px 0;border: none;background: #f1f1f1;"><br>


    <label for="mail" style="font-size: 24">Email-id:</label>
      <input type="email" id="email" name="email" placeholder="Enter Email" style="width:
100%;height: 30;padding: 9px;margin: 5px 0 22px 0;border: none;background: #f1f1f1;"><br>
```

```html
    <label for="password" style="font-size: 24">Password:</label>
     <input type="password" id="password" name="password" placeholder="Enter Password"
style="width: 100%;height: 30;padding: 9px;margin: 5px 0 22px 0;border: none;background:
#f1f1f1;"><br>
    <input type="submit" class="button button2" name="submit" value="Register">

</form>
</div>
</body>
</html>
```

Login.html:
```html
<!DOCTYPE html>
<html lang="en" >
<head>
  <meta charset="UTF-8">
  <title>HTML5 Login Form with validation Example</title>

<link
rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normalize.min.css">
  <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='style.css') }}">

</head>
<body>
<!-- partial:index.partial.html -->
<div id="login-form-wrap">
  <h2>Login</h2>
  <form id="login-form"  action="{{url_for('login') }}" method="POST">
   <p>
        <input    type="text"   id="username" name="username" placeholder="Username"
required><i class="validation"><span></span><span></span></i>
   </p>
   <p>
```

```html
    <input type="password" id="password" name="password" placeholder="Password"
required>
<i class="validation"><span></span><span></span></i>
    </p>
    <p>
    <input type="submit" id="login" value="Login">
    </p>
  </form>
  <div id="create-account-wrap">
    <p>Not a member? <a href="#">Create Account</a><p>
  </div><!--create-account-wrap-->
</div><!--login-form-wrap-->
<!-- partial -->

</body>
</html>
```

Dashboard.html:
```html
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link                                                        rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

<style>
body {
  font-family: "Lato", sans-serif;
  background-image: url("");
  background-attachment: fixed;
  background-position: 70% 50%; background-repeat: no-repeat;
}

/* Fixed sidenav, full height */
```

```css
.sidenav
 { height:
 100%; width:
 300px;
 position: fixed;
 z-index: 1;
 top: 0;
 left: 0;
 background-color: #0059b3;
 overflow-x: hidden;
 padding-top: 20px;
}

/* Style the sidenav links and the dropdown button */
.sidenav a{
 padding: 6px 8px 6px 16px;
 text-decoration: none;
 font-size: 20px;
 color: rgb(239, 239, 239);
 display: block;
 border: none;
 background: none;
 width: 100%;
 text-align: left;
 cursor: pointer;
 outline: none;
}

/* On mouse-over */
.sidenav
 a:hover{color:
 #111;
}
```

```
/* Some media queries for responsiveness */
@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}
</style>
</head>
<body>

<div class="sidenav">

  <a href="{{url_for('add_stock') }}"><strong>Add stock<strong></a>
  <a href="{{url_for('update') }}"><strong>Update stock details<strong></a>
  <a href="{{url_for('view_stock') }}"><strong>View stock<strong></a>
   <a href="{{url_for('delete')}}"><strong>Delete stock<strong></a>
  <a href="{{url_for('logout') }}"><strong>Log out<strong></a>




 </body>
 </html>
```

Add_stock.html:
```
<! Doctype html>
   <html lang="en">
   <head>
   <title>Add stock</title>

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <style>
    label{
    display: inline-block;
      width: 150px;
```

```css
      text-align: right;
      }



/* Add padding to containers */
#form1 {
  padding: 16px;
}


/* Full-width input fields */
input[type=text], input[type=number] input[type=float]  input[type=date] {

    height: 30px;
    width: 735px;
    text-align: center;
    clear: both;
    margin-bottom:0px;
    padding-top:5px;
}


input[type=text]:focus,
input[type=number]:focus
input[type=float]:focus
input[type=date]:focus
{ background-color:
#ddd; outline: none;
}

/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}
```

```css
/* Set a style for the submit/register button */
.registerbtn {
  background-color: darkblue;
  border-radius: 5px;
  -webkit-border-radius: 5px;
  padding: 4px 14px;
  height:34px;
  width:120px;
  cursor: pointer;
  font-size: 1em;
  border: 1px solid #b5b5b5;
  text-transform:uppercase;
  color:white;
  font-weight:bold;
}

.registerbtn:hover
  {opacity:1;
}

/* Set a grey background color and center the text of the "sign in" section */
#form1 {
  background-color: #f1f1f1;
  text-align: center;
}

div {
    margin-bottom: 10px;
  }

  .sidenav
{ height:
100%; width:
200px;
```

position: fixed;

```css
  z-index: 1;
  top: 0;
  left: 0;
  background-color: #0059b3;
  overflow-x: hidden;
  padding-top: 20px;
}

/* Style the sidenav links and the dropdown button */
.sidenav a{
  padding: 6px 8px 6px 16px;
  text-decoration: none;
  font-size: 20px;
  color: rgb(239, 239, 239);
  display: block;
  border: none;
  background: none;
  width: 100%;
  text-align: left;
  cursor: pointer;
  outline: none;
}

/* On mouse-over */
.sidenav a:hover, .dropdown-btn:hover
  {color: #111;
}

/* Some media queries for responsiveness */
@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}
```

```html
</style>
<center>
</head>
<body style="background-color:white;">
<h1 style="
text-align: center;
color: #fff;
font-size: 1.6em;
background: #616161;
display: block;
border-radius: 5px;
margin-bottom: 0px;
width: 700px;
padding: 30px 0px 30px 0px;
line-height: 0px !important;">  <br />ADD STOCK</h1>
<div class="sidenav">

  <a href="{{url_for('add_stock') }}"><strong>Add stock<strong></a>
  <a href="{{url_for('update') }}"><strong>Update stock details<strong></a>
  <a href="{{url_for('view_stock') }}"><strong>View stock<strong></a>
   <a href="{{url_for('delete')}}"><strong>Delete stock<strong></a>
 <a href="{{url_for('logout') }}"><strong>Log out<strong></a>
  </div>
<form method="post" action="{{url_for('add_stock') }}" style="
 background:#f7f7f7;
 width: 700px;
 border:1px solid #ccc;
 padding:30px 0px 30px 0px;
 box-shadow: 0px 5px 5px rgba(0,0,0,0.2);
 margin-top:10px;">

    <label> Product name:   </label> <input type="string" name="prodname"
style="width: 150px" autocomplete="off" >
```

```html
<br> <br>
    <label> Item quantity:   </label> <input type="number" name="quantity"
style="width: 150px" autocomplete="off" >

    <br> <br>
    <label
style="width:250px;margin-left:-175px;"> Warehouse location:   </label>
    <select name="warehouse_location">
        <option value="Chennai">Chennai</option>
        <option value="Banglore">Banglore</option>

    </select>
  <br> <br>
  <input class="registerbtn" type="submit" name="submit" value="Submit">
  </form>
  </center>
  </body>
  </html>
```

Delete_stock.html:

```html
<! Doctype html>
  <html lang="en">
  <head>
  <title>Delete stock</title>

  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <style>
  /*label{
      cursor: pointer;
      display: inline-block;
      padding: 3px 6px;
      text-align: right;
      width: 150px;
```

```css
        vertical-align: top;
    }
    .error {
            color: white;
            font-family: lato;
            color:red;
            display: inline-block;
            padding: 2px 10px;
    }
    input{


    } */
    label{
    display: inline-block;
      width: 150px;
      text-align: right;
      }

/* Add padding to containers */
#form1 {
  padding: 16px;
}
/* Full-width input fields */
input[type=text], input[type=number] input[type=float]  input[type=date] {
  /*width:  20%;
  padding: 15px;
  margin: 5px 0 22px 0;
  display: inline-block;
  border: none;
  background: #f1f1f1;*/

  height: 30px;
  width: 735px;
  text-align: center;
```

```css
    clear: both;
    margin-bottom:0px;
    padding-top:5px;
}
input[type=text]:focus,
input[type=number]:focus
input[type=float]:focus
input[type=date]:focus
{ background-color:
#ddd; outline: none;
}

/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* Set a style for the submit/register button */
.registerbtn {
  /*font-size:130%;
  color: white;
  padding: 16px 20px;
  margin-left: 10% ;
  border: none;
  cursor: pointer;
  width: 10%;
  opacity: 0.9;*/

  background-color: darkblue;
  border-radius: 5px;
  -webkit-border-radius: 5px;
  padding: 4px 14px;
  height:34px;
```

```css
  width:120px;
  cursor: pointer;
  font-size: 1em;
  border: 1px solid #b5b5b5;
  text-transform:uppercase;
  color:white;
  font-weight:bold;
}

.registerbtn:hover
  {opacity:1;
}

/* Set a grey background color and center the text of the "sign in" section */
#form1 {
  background-color: #f1f1f1;
  text-align: center;
}

div {
      margin-bottom: 10px;
    }

    .sidenav
  { height:
  100%; width:
  300px;
  position: fixed;
  z-index: 1;
  top: 0;
  left: 0;
  background-color: #0059b3;
  overflow-x: hidden;
  padding-top: 20px;
}
```

```css
/* Style the sidenav links and the dropdown button */
.sidenav a{
  padding: 6px 8px 6px 16px;
  text-decoration: none;
  font-size: 20px;
  color: rgb(239, 239, 239);
  display: block;
  border: none;
  background: none;
  width: 100%;
  text-align: left;
  cursor: pointer;
  outline: none;
}

/* On mouse-over */
.sidenav a:hover, .dropdown-btn:hover
  {color: #111;
}

/* Some media queries for responsiveness */
@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}
    </style>
    <center>
    </head>
    <body style="background-color:white;">
      <div class="sidenav">

        <a href="{{url_for('add_stock') }}"><strong>Add stock<strong></a>
        <a href="{{url_for('update') }}"><strong>Update stock details<strong></a>
        <a href="{{url_for('view_stock') }}"><strong>View stock<strong></a>
```

```html
      <a href="{{url_for('delete')}}"><strong>Delete stock<strong></a>
     <a href="{{url_for('logout') }}"><strong>Log out<strong></a>
      </div>
  <h1 style="
  text-align: center;
  color: #fff;
  font-size: 1.6em;
  background: #616161;
  display: block;
  border-radius: 5px;
  margin-bottom: 0px;
  width: 700px;
  padding: 30px 0px 30px 0px;
  line-height: 0px !important;">  <br />Delete stock</h1>
  <form method="post" action="{{url_for('delete_stock') }}" style="
   background:#f7f7f7;
   width: 700px;
   border:1px solid #ccc;
   padding:30px 0px 30px 0px;
   box-shadow: 0px 5px 5px rgba(0,0,0,0.2);
   margin-top:10px;">
   <label> Product name:   </label> <input type="text" name="prodname"
style="width: 150px" autocomplete="off" >

   <br> <br>
   <input class="registerbtn" type="submit" name="submit" value="Delete">
  </form>
  </center>
  </body>
  </html>
```

delete_stock.html(after deleting a stock):

```html
<!DOCTYPE html>
<html>
<style>
.registerbtn {
 /*font-size:130%;
 color: white;
 padding: 16px 20px;
 margin-left: 10% ;
 border: none;
 cursor: pointer;
 width: 10%;
 opacity: 0.9;*/

 background-color: darkblue;
 border-radius: 5px;
 -webkit-border-radius: 5px;
 padding: 4px 14px;
 height:34px;
 width:320px;
 cursor: pointer;
 font-size: 1em;
 border: 1px solid #b5b5b5;
 text-transform:uppercase;
 color:white;
 font-weight:bold;
}
</style>
<body>
    <center>
        <h3>Record deleted successfully!</h3>
            <form action="" method="POST">
            <input type="submit" value="Redirect to Dashboard" class="registerbtn"/>
            </form>
```

```html
        </center>
</body>

</html>
```

Update_stock.html:
```html
<! Doctype html>
    <html lang="en">
    <head>
    <title>Add stock</title>

      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
      <style>
      /*label{
          cursor: pointer;
          display: inline-block;
          padding: 3px 6px;
          text-align: right;
          width: 150px;
          vertical-align: top;
       }
      .error {
              color: white;
              font-family: lato;
              color:red;
              display: inline-block;
              padding: 2px 10px;
      }
      input{


      } */
      label{
      display: inline-block;
```

```css
            width: 150px;
            text-align: right;
            }


/* Add padding to containers */
#form1 {
  padding: 16px;
}


/* Full-width input fields */
input[type=text], input[type=number] input[type=float]  input[type=date] {
 /*width:  20%;
 padding: 15px;
 margin: 5px 0 22px 0;
 display: inline-block;
 border: none;
 background: #f1f1f1;*/


   height: 30px;
   width: 735px;
   text-align: center;
   clear: both;
   margin-bottom:0px;
   padding-top:5px;
}

input[type=text]:focus,
input[type=number]:focus
input[type=float]:focus
input[type=date]:focus
{ background-color:
#ddd; outline: none;
}
```

```css
/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* Set a style for the submit/register button */
.registerbtn {
  /*font-size:130%;
  color: white;
  padding: 16px 20px;
  margin-left: 10% ;
  border: none;
  cursor: pointer;
  width: 10%;
  opacity: 0.9;*/

  background-color: darkblue;
  border-radius: 5px;
  -webkit-border-radius: 5px;
  padding: 4px 14px;
  height:34px;
  width:120px;
  cursor: pointer;
  font-size: 1em;
  border: 1px solid #b5b5b5;
  text-transform:uppercase;
  color:white;
  font-weight:bold;
}
.registerbtn:hover
  {opacity:1;
}
```

```css
/* Set a grey background color and center the text of the "sign in" section */
#form1 {
  background-color: #f1f1f1;
  text-align: center;
}

div {
     margin-bottom: 10px;
   }

   .sidenav
 { height:
 100%; width:
 300px;
 position: fixed;
 z-index: 1;
 top: 0;
 left: 0;
 background-color: #0059b3;
 overflow-x: hidden;
 padding-top: 20px;
}

/* Style the sidenav links and the dropdown button */
.sidenav a{
  padding: 6px 8px 6px 16px;
  text-decoration: none;
  font-size: 20px;
  color: rgb(239, 239, 239);
  display: block;
  border: none;
  background: none;
  width: 100%;
  text-align: left;
  cursor: pointer;
```

outline: none;

```
}

/* On mouse-over */
.sidenav a:hover, .dropdown-btn:hover
  {color: #111;
}

/* Some media queries for responsiveness */
@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}
    </style>
    <center>
    </head>
    <body style="background-color:white;">
      <div class="sidenav">


        <a href="{{url_for('add_stock') }}"><strong>Add stock<strong></a>
        <a href="{{url_for('update') }}"><strong>Update stock details<strong></a>
        <a href="{{url_for('view_stock') }}"><strong>View stock<strong></a>
         <a href="{{url_for('delete')}}"><strong>Delete stock<strong></a>
      <a href="{{url_for('logout') }}"><strong>Log out<strong></a>
        </div>
    <h1 style="
    text-align: center;
    color: #fff;
    font-size: 1.6em;
    background: #616161;
    display: block;
    border-radius: 5px;
    margin-bottom: 0px;
    width: 700px;
    padding: 30px 0px 30px 0px;
```

```
line-height: 0px !important;">  <br />UPDATE STOCK</h1>
<form method="post" action="{{url_for('update_stock') }}" style="
 background:#f7f7f7;
 width: 700px;
 border:1px solid #ccc;
 padding:30px 0px 30px 0px;
 box-shadow: 0px 5px 5px rgba(0,0,0,0.2);
 margin-top:10px;">


    <label> Product name:   </label> <input type="string" name="prodname"
style="width: 150px" autocomplete="off" >


   <br> <br>
     <label> Item quantity:   </label> <input type="number" name="quantity"
style="width: 150px" autocomplete="off" >


   <br> <br>


   <label
style="width:250px;margin-left:-175px;"> Warehouse location:   </label>
    <select name="warehouse_location">
        <option value="Chennai">Chennai</option>
        <option value="Banglore">Banglore</option>
       </select>
    <br> <br>

 <input class="registerbtn" type="submit" name="submit" value="Update">
 </form>
 </center>
 </body>
 </html>
```

View_stock.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
      table {
          margin: 0 auto;
          font-size: large;
          border: 1px solid black;
      }

      h1 {
          text-align: center;
          color: #006600;
          font-size: xx-large;
          font-family: 'Gill Sans', 'Gill Sans MT',
          ' Calibri', 'Trebuchet MS', 'sans-serif';
      }

      td {
          background-color: #E4F5D4;
          border: 1px solid black;
      }

      th,
      td {
          font-weight: bold;
          border: 1px solid black;
          padding: 10px;
          text-align: center;
      }

      td {
          font-weight: lighter;
```

```css
      }
      .sidenav
{          height:
 100%;    width:
 300px;
 position: fixed;
 z-index: 1;
 top: 0;
 left: 0;
 background-color: #0059b3;
 overflow-x: hidden;
 padding-top: 20px;
}

/* Style the sidenav links and the dropdown button */
.sidenav a{
  padding: 6px 8px 6px 16px;
  text-decoration: none;
  font-size: 20px;
  color: rgb(239, 239, 239);
  display: block;
  border: none;
  background: none;
  width: 100%;
  text-align: left;
  cursor: pointer;
  outline: none;
}

/* On mouse-over */
.sidenav a:hover, .dropdown-btn:hover
  {color: #111;
}

/* Some media queries for responsiveness */
```

```
@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}



    </style>

</head>
<body>
   <div class="sidenav">

      <a href="{{url_for('add_stock') }}"><strong>Add stock<strong></a>
      <a href="{{url_for('update') }}"><strong>Update stock details<strong></a>
      <a href="{{url_for('view_stock') }}"><strong>View stock<strong></a>
       <a href="{{url_for('delete')}}"><strong>Delete stock<strong></a>
     <a href="{{url_for('logout') }}"><strong>Log out<strong></a>
      </div>

    <div class="container" >

     <table class="table table-hover">
       <thead>
        <tr>
            <th>Product name</th>
            <th>Item quantity</th>
            <th>Warehouse location</th>
        </tr>
       </thead>

      {% for item in products %}

      <tr>
        <td>{{item.PRODNAME}}</td>
```

```
        <td>{{item.QUANTITY}}</td>
        <td>{{item.WAREHOUSE_LOCATION}}</td>


    </tr>
    {% endfor %}
  </table>
    </table>


 <body>
 </html>
```

GitHub link :

  https://github.com/IBM-EPBL/IBM-Project-14285-1659548197

Demo link : **https://www.dropbox.com/s/akjv781mitiqmx6/Demo%20Video%20link?dl=0**