INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

1.INTRODUCTION

1.1 PROJECT OVERVIEW

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.

Once retailers successfully log in to the application they can update their inventory details, also 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.

1.2 PURPOSE

The main purpose of inventory management is to help businesses easily and efficiently manage the ordering, stocking, storing, and using of inventory. By effectively managing your inventory, you'll always know what items are in stock, how many of them there are, and where they are located. Plus, practicing strong inventory management allows you to understand how you use your inventory–and how demand changes for it–over time. You can zero in on exactly what you need, what's not so important, and what's just a waste of money. That's using inventory management to practice inventory control. By the way, inventory control is the balancing act of always having enough stock to meet demand, while spending as little as possible on ordering and carrying inventory.

The over and under the stock of inventory usually create the requirements of different types of inventory, a period of the stock, and cost associated with it. The main objectives of inventory management are:

1. To supply the required materials continuously: The main objective of inventory management is to maintain the required inventory to run the production and sales process smoothly.
2. To minimize the risk of under and overstocking of material: Inventory management manages to minimize the risk caused due to under and overstocking of the inventory.
3. To reduce losses damages and misappropriation of materials: Inventory management aims to reduce or remove the losses of materials and stock, done by maintaining the proper stock of materials with utmost care.

2.LITERATURE SURVEY

2.1 EXISTING PROBLEM

Pastore and Martin (2012) study was to examine students' perceptions of designing and developing mobile based instructions by interviewing and surveying of graduate students. Results of the survey and qualitative data analysis indicated that usability was a key issue on the mobile device. Users enjoyed quick access, good organization, user control, single column layouts, and large links/buttons. These findings contribute to the literature base on the design and development of mobile based instruction.

Norman E (2012) discusses, while existing factors identified in the literature were found to be present in the context of today's design program, the critical perspective of this study recontextualized these factors, along with the identification of new or underrepresented factors. Taking on the perspective of a student's experience of pedagogy foregrounds issues of uncertainty and ambiguity, highlighting the social interactions between fellow students, and the role of communication and individual effort in learning to think in a more designerly way.

Agnelo and Fernandes(2012) aims to analyze, through a case study called Researching the Value of Project Management, the relations of the constructs of this conceptual model and to show how they interfere with the organizational values, possibly in programs conducted by a government agency, from the perspective of the senior management directly involved. The term emerging technology (ET) has been frequently used by IT professionals and academics. However, little research has shed light on this term and specified its characteristics and what it means. Halaweh M (2013) aims to define and conceptualize the characteristics of ET. These characteristics are uncertainty, network effect, unseen social and ethical concerns, cost, limitation to particular countries, and a lack of investigation and research.

Antonelli and et al (2013) aims to identify Information Technology benefits in individual work. With technologies fully implemented, greater satisfaction was observed for all constructs of the survey, with statistically significant differences. When comparing age, it was found that younger users were more satisfied with the benefits of technology. Concerning the number of employees, small business users were less satisfied with Information Technology.

Alderete (2013) presents an econometric model to determine whether an SME (Small and Medium Sized Enterprise)'s probability of outsourcing depends on their levels of innovation and information and communication technology use. The model predicts that the level of innovation of an SME will significantly influence its probability of outsourcing. Besides, it stresses the negative incidence of the information and communication technologies (ICT) access on the outsourcing decision.

Didonet and Díaz, (2012) explains, the supply chain management studies have verified that integration and collaboration in the supply chain can provide important benefits to the companies involved. Among these benefits are added value, the creation of efficiencies and client, which are represented by the reduction in inventories, improvements in service delivery and quality and shorter product development cycles.

Zabala (2012) investigates whether decisions considered as common in new product development literature are also valid in a region characterized by traditional industries. The author aims to link the theoretical and empirical fields in the context of new product development and product innovation management.

Leber (2014) reports the results of a survey on the use of innovation management techniques with the potential to improve effectiveness of new product development, and customer satisfaction. Failure mode and effects analysis was found as the most applied IMT in Slovene firms with the highest perceived utility potential to reduce development costs and improve customer satisfaction.

2.2 REFERENCES

1. R. Ishfaq, C. C. Defee, B. J. Gibson and U. Raja, "Realignment of the physical distribution process in omni-channel fulfillment", International Journal of Physical Distribution & Logistics Management, vol. 46, no. 6/7, pp. 543-561, jul. 2016.

2. J. Kembro and A. Norrman, "Exploring trends implications and challenges for logistics information systems in omni-channels : Swedish retailers' perception", International Journal of Retail and Distribution Management, vol. 47, no. 4, pp. 384-411, 2019.

3. G. Hançerlioğulları, A. Şen and E. A. Aktunç, "Demand uncertainty and inventory turnover performance: an empirical analysis of the US retail industry", International Journal of Physical Distribution and Logistics Management, vol. 46, no. 6–7, pp. 681-708, 2016.

4. J. D. Sterman and G. Dogan, "'I'm not hoarding i'm just stocking up before the hoarders get here.': Behavioral causes of phantom ordering in supply chains", Journal of Operations Management, vol. 39, pp. 6-22, 2015.

5. Y. Wang, S. W. Wallace, B. Shen and T.-M. Choi, "Service supply chain management: A review of operational models", European Journal of Operational Research, vol. 247, no. 3, pp. 685-698, 2015.

6. S. Mahar and P. D. Wright, "The value of postponing online fulfillment decisions in multi-channel retail/e-tail organizations", Computers & operations research, vol. 36, no. 11, pp. 3061-3072, 2009.

7. A. Hübner, A. Holzapfel and H. Kuhn, "Operations management in multi-

channel retailing: an exploratory study", Operations Management Research, vol. 8, no. 3–4, pp. 84-100, 2015.

8. A. Fink, "Conducting research literature reviews: From the internet to paper" in , Sage publications, 2019.

2.3 PROBLEM STATEMENT DEFINITION

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.

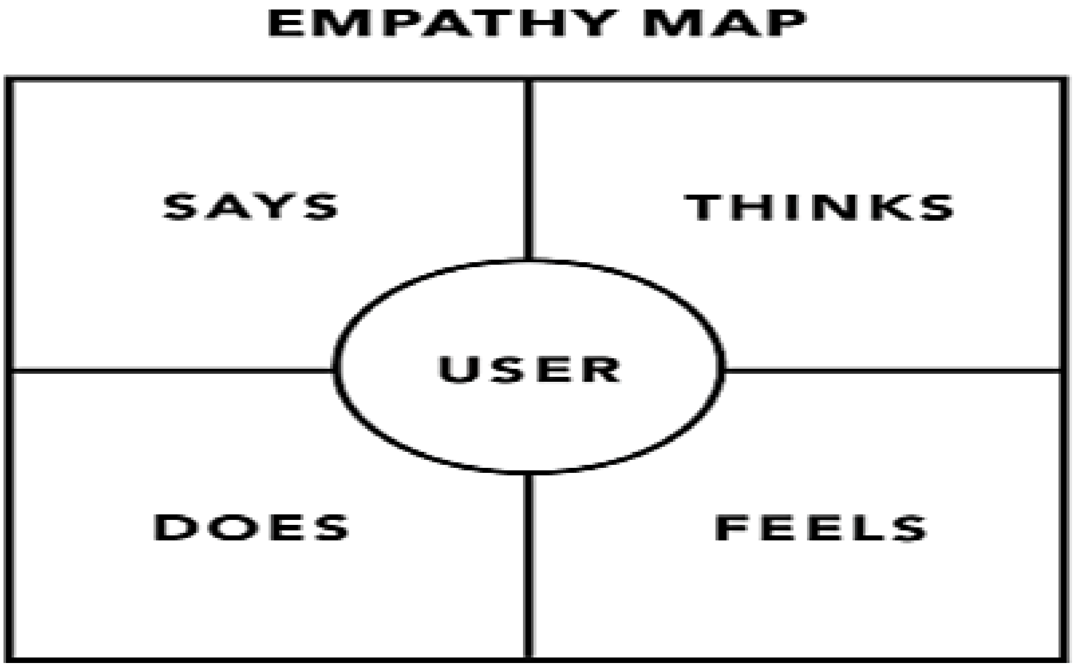
Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.

3. IDEATION & PROPOSED SOLUTION

3.1EMPATHY MAP

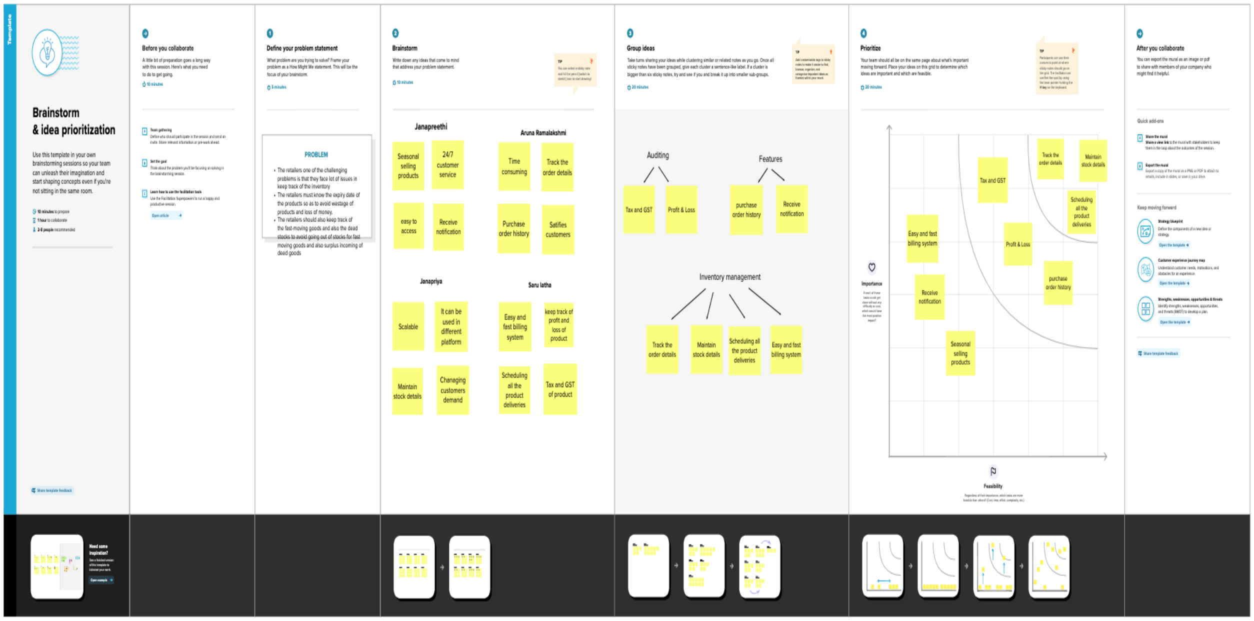
An empathy map is a collaborative visualization used to articulate what to know about a particular type of user. It externalizes knowledge about users in order to 1) create a shared understanding of user needs, and 2) aid in decision making.

Traditional empathy maps are split into 4 quadrants (Says, Thinks, Does, and Feels), with the user or persona in the middle. Empathy maps provide a glance into who a user is as a whole and are not chronological or sequential. The figure 3.1 Empathy Map Format shows the basic Structure of the empathy



3.2 IDEATION & BRAINSTORMING

As this is an open ended problem statement, We can start with initial brainstorming. By this process we’d have an idea of overall app structure & features.



Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

- The retailers one of the challenging problems is that they face lot of issues in keep track of the inventory
- The retailers must know the expiry date of the products so as to avoid wastage of products and loss of money.
- The retailers should also keep track of the fast-moving goods and also the dead stocks to avoid going out of stocks for fast moving goods and also surplus incoming of dead goods

2 Brainstorm

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

10 minutes

Janapreethi

Seasonal selling products

24/7 customer service

easy to access

Receive notification

Aruna Ramalakshmi

Time consuming

Track the order details

Purchase order history

Satifies customers

Janapriya

Scalable

It can be used in different platform

Maintain stock details

Chanaging customers demand

Saru latha

Easy and fast billing system

keep track of profit and loss of product

Scheduling all the product deliveries

Tax and GST of product

3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, 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

Auditing

Tax and GST

Profit & Loss

Features

purchase order history

Receive notification

Inventory management

Track the order details

Maintain stock details

Scheduling all the product deliveries

Easy and fast billing system

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

Tax and GST

Track the order details

Maintain stock details

Scheduling all the product deliveries

Easy and fast billing system

Profit & Loss

Receive notification

purchase order history

Seasonal selling products

Importance

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

Feasibility

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

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

A Share the mural
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

B Export the mural
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

Share template feedback

3.3 PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem tobe solved)	● The retailer's one of the challenging problems is that they face lot of issues in keep track of the inventory ● The retailers must know the expiry date of the products so as to avoid wastage of products and loss of money ● The retailers shouldalso keep trackof the fast-moving goods and also the dead stocks to avoid going out of stocks for fast moving goods and also surplus incoming of dead goods

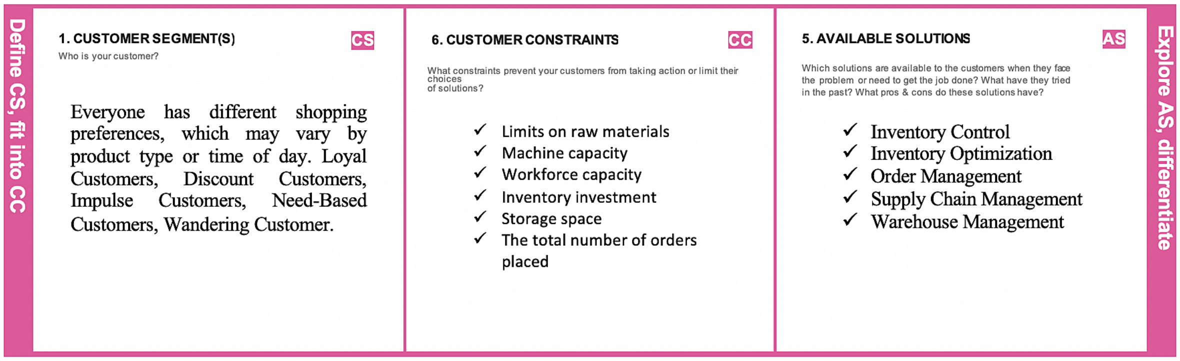
2.	Idea / Solution description	● Applications have been developedto help retailers track and manage stocks relatedto their own products. The System will ask retailers to create theiraccounts by providing essential details. Retailers can access theiraccounts by logginginto the application. ● Once retailers successfully login to the application they can update their inventory details, also 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 ordernew stock.

3.	Novelty / Uniqueness	● User can trackthe record of goods available using the application. Inventory tracking helpsto improve inventory management and ensuresthat having optimal stock available tofulfill orders. ● Reduces manpower, cost and saves time. Emails will be sent automatically While stocks are not available.Makes the business processmore efficient.Improves organizations performance.

4.	Social Impact / Customer Satisfaction	● Customer satisfaction is the keyfor success of a business. The availability of product is just one way in which an inventory management system creates customer satisfaction. ● Inventory management systemsare designed to monitor productavailability, determine purchasing schedules for better customer interaction.
5.	Business Model (Revenue Model)	● Hereby we can provide a robust and most reliable inventory management system by using: ○ Providing user friendly platforms for the user ○ Preparing and implementing strategies ○ for no or minimal loss ○ Using attractive advertisements which are closely related to the lifestyle of people. ○ Providing attractive discounts and usersto the customers. ○ An application to monitor all the orders ○ Shrinkage or expansion of stock ○ database as per requirement

6.	Scalability of the Solution	● This proposed systemfor inventory management system can accommodate expansion without restricting the existing workflow and ensure an increase in the output or efficiency of the process.

3.4 PROBLEM SOLUTON FIT



1. CUSTOMER SEGMENT(S) CS

Who is your customer?

Everyone has different shopping preferences, which may vary by product type or time of day. Loyal Customers, Discount Customers, Impulse Customers, Need-Based Customers, Wandering Customer.

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions?

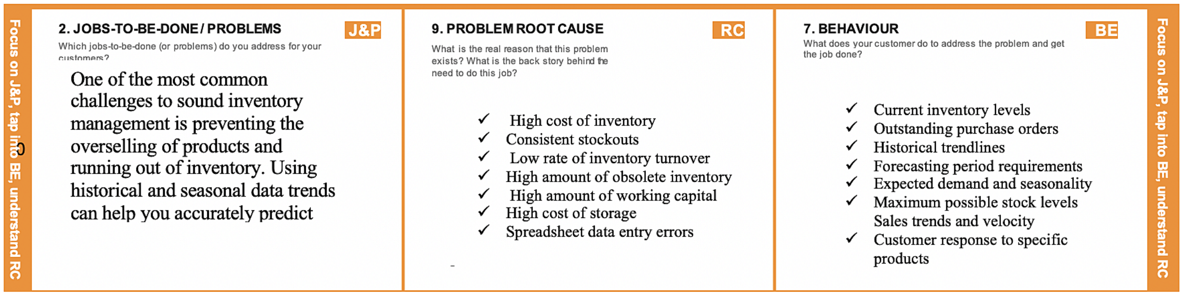
- ✓ Limits on raw materials
- ✓ Machine capacity
- ✓ Workforce capacity
- ✓ Inventory investment
- ✓ Storage space
- ✓ The total number of orders placed

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?

- ✓ Inventory Control
- ✓ Inventory Optimization
- ✓ Order Management
- ✓ Supply Chain Management
- ✓ Warehouse Management

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?

One of the most common challenges to sound inventory management is preventing the overselling of products and running out of inventory. Using historical and seasonal data trends can help you accurately predict

9. PROBLEM ROOT CAUSE RC

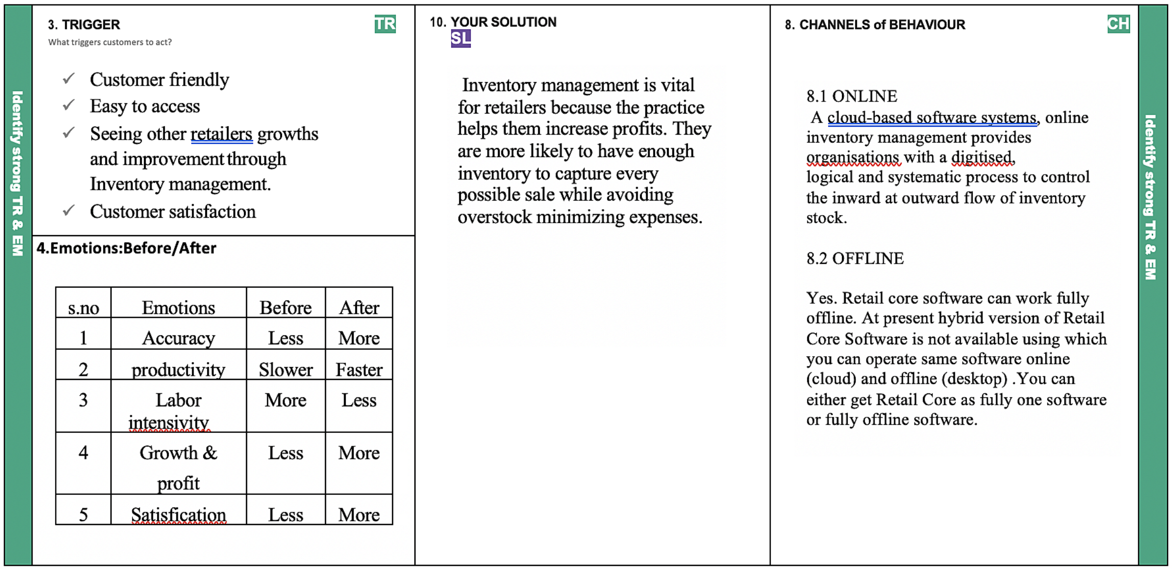
What is the real reason that this problem exists? What is the back story behind the need to do this job?

- ✓ High cost of inventory
- ✓ Consistent stockouts
- ✓ Low rate of inventory turnover
- ✓ High amount of obsolete inventory
- ✓ High amount of working capital
- ✓ High cost of storage
- ✓ Spreadsheet data entry errors

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?

- ✓ Current inventory levels
- ✓ Outstanding purchase orders
- ✓ Historical trendlines
- ✓ Forecasting period requirements
- ✓ Expected demand and seasonality
- ✓ Maximum possible stock levels Sales trends and velocity
- ✓ Customer response to specific products



Identify strong TR & EM

3. TRIGGER TR

What triggers customers to act?

- ✓ Customer friendly
- ✓ Easy to access
- ✓ Seeing other retailers growths and improvement through Inventory management.
- ✓ Customer satisfaction

4.Emotions:Before/After

s.no	Emotions	Before	After
1	Accuracy	Less	More
2	productivity	Slower	Faster
3	Labor intensivity	More	Less
4	Growth & profit	Less	More
5	Satisfication	Less	More

10. YOUR SOLUTION SL

Inventory management is vital for retailers because the practice helps them increase profits. They are more likely to have enough inventory to capture every possible sale while avoiding overstock minimizing expenses.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE

A cloud-based software systems, online inventory management provides organisations with a digitised, logical and systematic process to control the inward at outward flow of inventory stock.

8.2 OFFLINE

Yes. Retail core software can work fully offline. At present hybrid version of Retail Core Software is not available using which you can operate same software online (cloud) and offline (desktop) .You can either get Retail Core as fully one software or fully offline software.

Identify strong TR & EM

4.REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIRMENT

FR No.	Functional Requirement (Epic)	Sub Requirement (Story/ Sub-Task)
FR-1	Account Creation	Creation through Google Creation through Email Creation through LinkedIn Creation through Github
FR-2	User Confirmation	Confirmation via Email Confirmation viaOTP
FR-3	Successful Log in	Notification through Email Notification through Phonenumber
FR-4	Update inventory details	Notification through Email Notification throughPhone number

FR-5	Addnew stock	Notification through Email Notification through Phonenumber
FR-6	Unavailability of stock	Alert notificationthrough Email, PhoneNumber

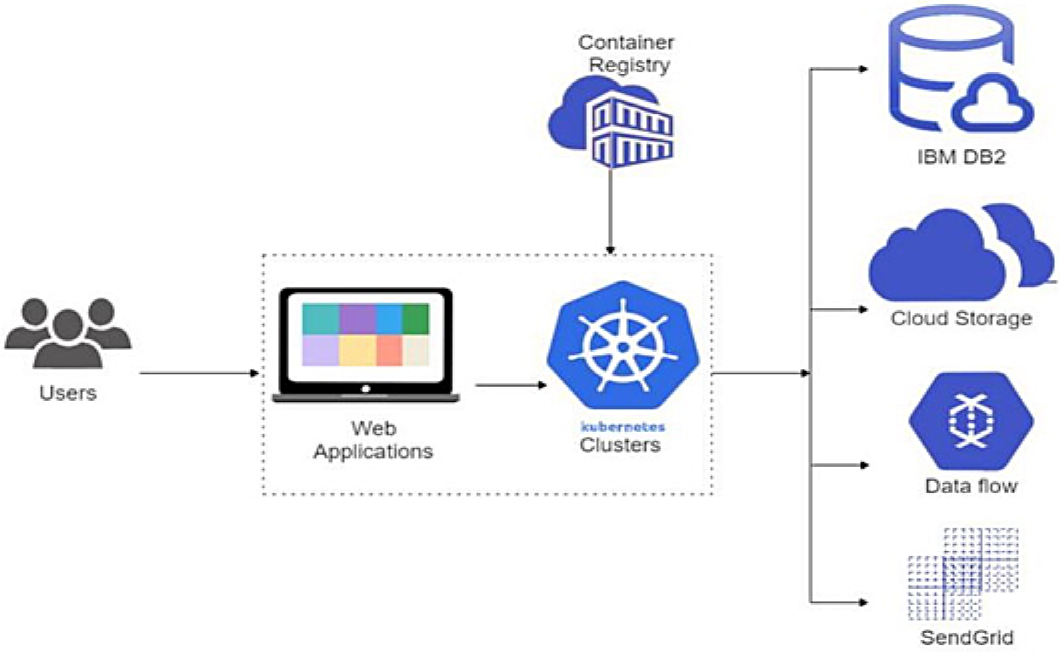
4.2 NON-FUNCTIONAL REQUIREMENTS

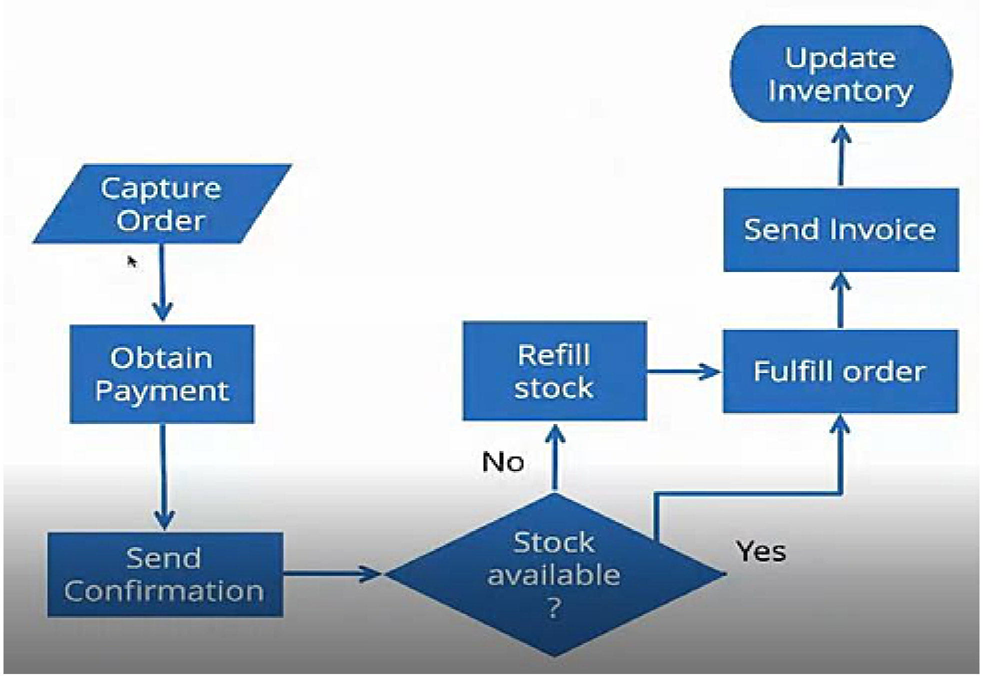
FR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the currentinventory. The Systemwill automatically sendan email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.
NFR-2	**Security**	Applications have been developed to help retailers trackand manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. With Registered Mailid only retailers can log intothe application. So it provide authentication.

NFR-3	**Performance**	User can track the record of goods available using the application. Inventory tracking helps to improve inventory management and ensures that having optimal stock available to fulfill orders.Reduces manpower , cost and savestime. Emails willbe sent automatically While stocks are not available.Makes the business process more efficient.Improves organizations performance.
NFR-4	**Availability**	The availability of product is just one way in which an inventory management system creates customersatisfaction. Inventory management systems are designed to monitor product availability, determine purchasing schedules for better customer interaction.
NFR-5	**Scalability**	Scalability is an aspect or rather a functional quality of asystem, software or solution.This proposed system for inventory management system can accommodate expansion withoutrestricting the existing workflow and ensure an increase in the output or efficiency of theprocess.

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAM





5.2 SOLUTION & TECHNICAL ARCHITECTURE

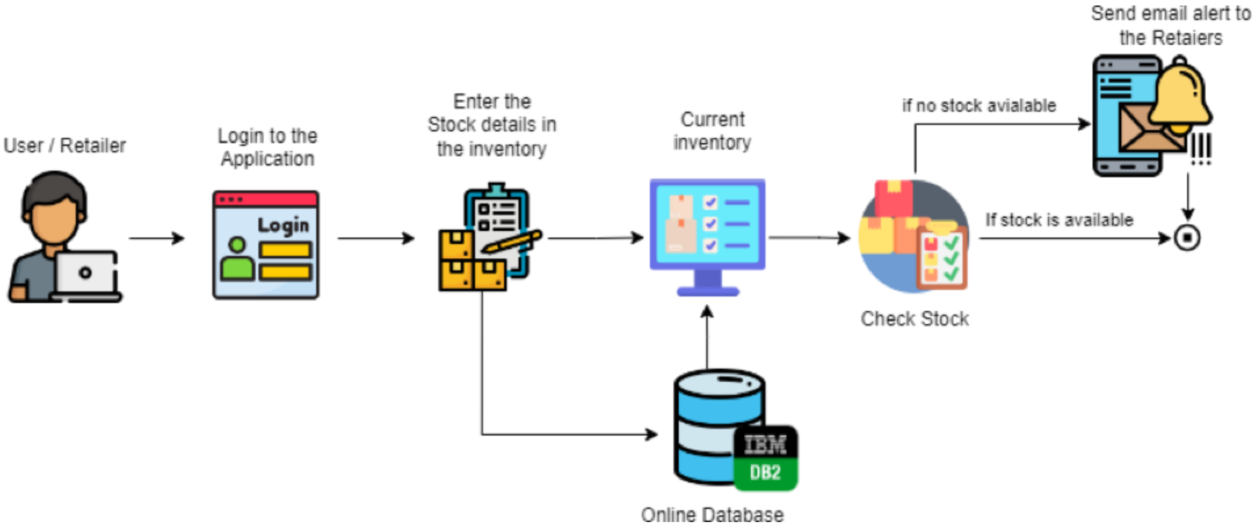


Table-1 : Components & Technologies:

S.NO	Component	Description	Technology
	User Interface	How user interacts with application, e.g. Mobile App,Chatbot.	HTML, CSS, Python, MySQL, Flask
1.	Application Logic-1	Logic for a process in the application	Python
2.	Application Logic-2	Logic for a process in the application	SendGrid
3.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
4.	Database	Data Type,Configurations etc.	MySQL

5.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloud.
6.	File Storage	File storage requirements	IBM ObjectStorage

Table-2: Application Characteristics:

S. No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Micro web framework, written in Python	Flask
2.	Security Implementations	a set of cryptographic hashfunctions	SHA-256
3.	Scalable Architecture	Kubernetes is an open-source container orchestration system for automating softwaredeployment, scaling, and management.	Kubernetes
4.	Availability	To customize settings for the dockerCLI. The configuration file usesJSON formatting.	Docker CLI
5.	Performance	To send alerts to users on based on their stock	Sendgrid

5.3 USER STORIES:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile 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
		USN-2	As a user, I will receive confirmation emailonce I have registered for the application	I canreceive confirmation email& click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register &access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I canreceive confirmation email& click confirm	Medium	Sprint-1
	Login	USN-5	As a user,I can log into the application by entering email & password	I can enter intomy account	High	Sprint-1
	Dashboard	USN-6	As a user, it displays all top	I can avail the recent offerin	High	Sprint-2

			brands and offers of product	sale		
Customer (Webuser)	Application	USN-7	As a user, I can register, login and shop the products easily	I can access my shopping very soon	High	Sprint-3
Customer CareExecutive	Update Inventory details	USN-8	To keeptrack of order and availability of stockin inventory	I cancontrol the inventory stock correctly	High	Sprint-4
Administrator	Add new stock	USN-9	To add new products intothe application	I can provide new trend products in an application for customer needs	High	Sprint-3
Customer CareExecutive	Verify customer feedback	USN-10	To design application that meets user’sdesires	I can satisfy the customer expectations	High	Sprint-4
Customer CareExecutive	Inventory Control	USN-11	To refill the unavailability of stock in inventory	I can alert notification through email	Medium	Sprint-2
Administrator	Performance of Application	USN-12	To make the business process more efficient	I can save time, cost to fulfil orders by improving the inventory management	High	Sprint-4

6. PROJECT PLANNING & SCHEDULING

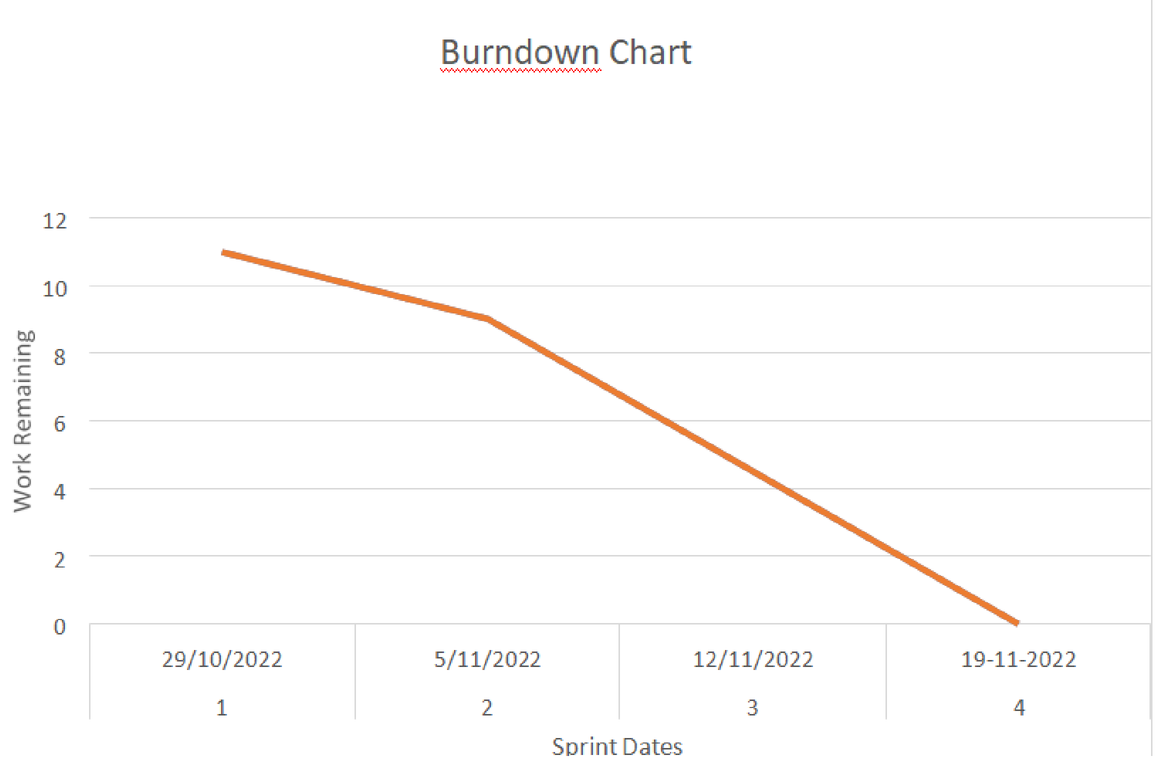
6.1 Sprint Planning & Estimation

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 using my email & password andconfirming my logincredentials.	3	High	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-1		USN-2	As a user, I canlogin through my E-mail.	3	Medium	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-1	Confirmation	USN-3	As a user, I can receivemy confirmation emailonce I have registered for the application.	2	High	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-1	Login	USN-4	As a user, I can log in to the authorized accountby entering the	3	Medium	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M

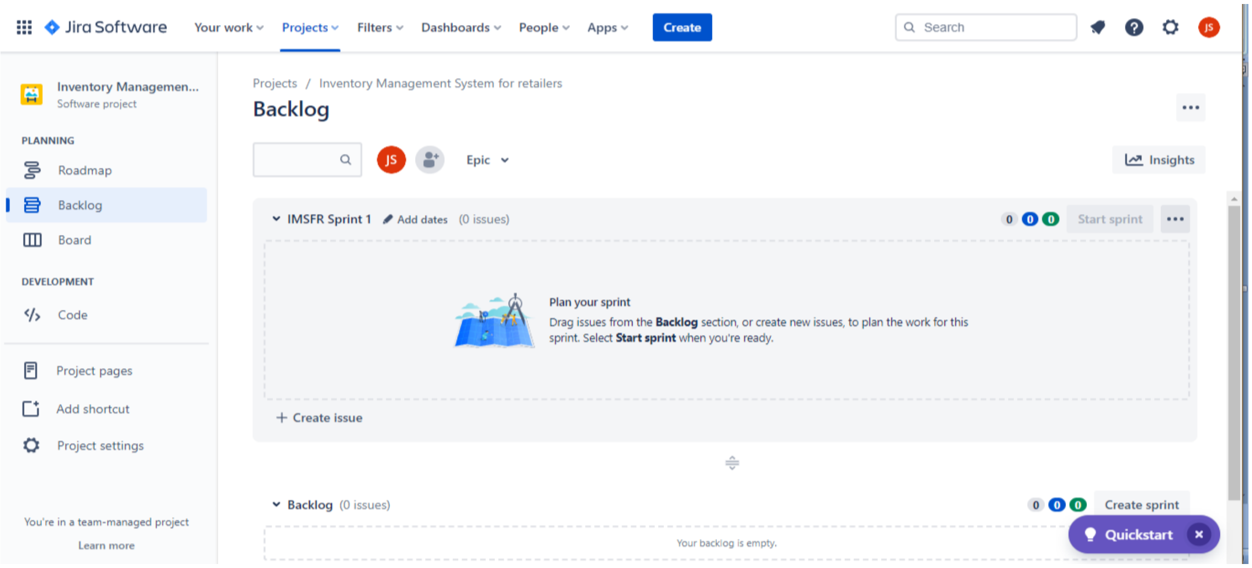
			registered email and password.			
Sprint-2	Dashboard	USN-5	As a user, I can viewthe products thatare available currently.	4	High	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-2	Stocks update	USN-6	As a user, I can add products which are not available in the inventory and restock the products.	3	Medi um	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-3	Sales prediction	USN-7	As a user, I can get access to sales prediction tool whichcan help me to predict better restock management of product.	6	Medi um	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M
Sprint-4	Request for customer care	USN-8	As a user, I am ableto request customer care to get in touch with the administrators and enquire the doubts andproblems.	4	Medi um	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M

Sprint-4	Giving feedback	USN-9	As a user, I am able to send feedback forms reporting any ideas for improving or resolving anyissues I am facingto get it resolved.	3	Medi um	Janapreethi S Janapriya S Aruna Ramalakshmi P SaruLatha M

6.2 Sprint Delivery Schedule



6.3 Reports from JIRA



IMSFR Sprint 1 Add dates (0 issues) 0 0 0

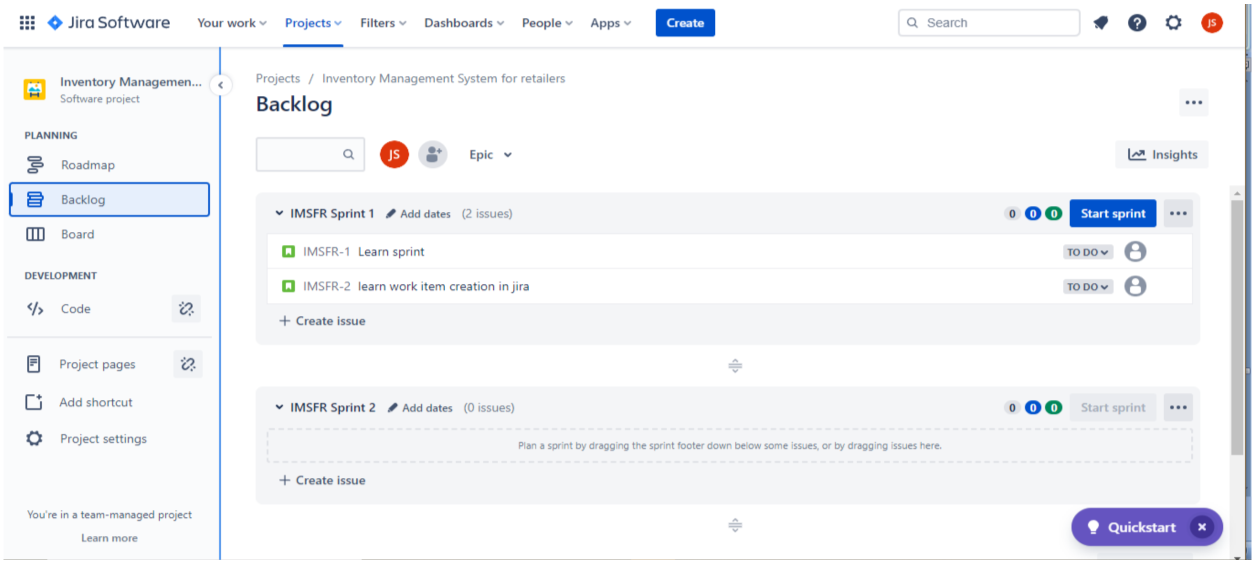
Plan your sprint

Drag issues from the **Backlog** section, or create new issues, to plan the work for this sprint. Select **Start sprint** when you're ready.

+ Create issue

Backlog (0 issues) 0 0 0

Your backlog is empty.



IMSFR Sprint 1 Add dates (2 issues) 0 0 0

IMSFR-1 Learn sprint TO DO

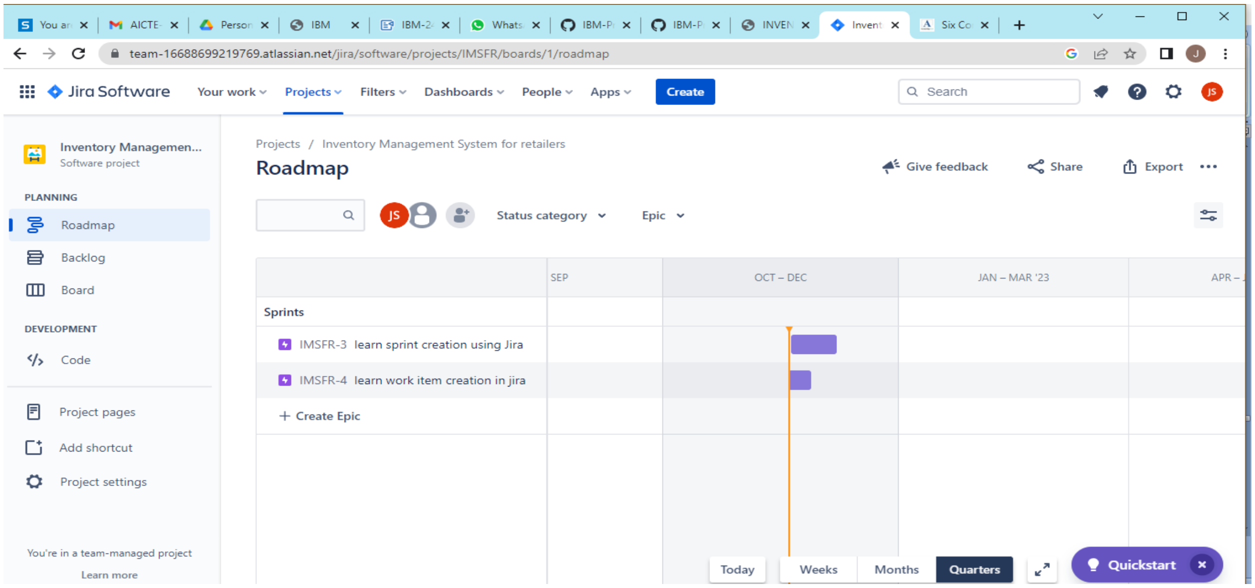
IMSFR-2 learn work item creation in jira TO DO

+ Create issue

IMSFR Sprint 2 Add dates (0 issues) 0 0 0

Plan a sprint by dragging the sprint footer down below some issues, or by dragging issues here.

+ Create issue



7. CODING & SOLUTIONING

7.1 Feature 1

APP.PY

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField, IntegerField
import ibm_db
from passlib.hash import sha256_crypt
from functools import wraps

from sendgrid import *

#creating an app instance
app = Flask(_name_)
```

```
app.secret_key='a'
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=824dfd4d-99de-440d-
9991-
629c01b3832d.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30119;S
ECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=tyt27664;
PWD=hrAJhoiC3znla3rh;",",")

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

#Products
@app.route('/products')
def products():
    sql = "SELECT * FROM products"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)

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

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

```
#Locations
@app.route('/locations')
def locations():

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

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

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

#Product Movements
@app.route('/product_movements')
def product_movements():

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

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

    if result>0:
        return render_template('product_movements.html', movements = movements)
    else:
        msg='No product movements found'
        return render_template('product_movements.html', msg=msg)

#Register Form Class
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
    username = StringField('Username', [validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6, max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')

#user register
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password = sha256_crypt.encrypt(str(form.password.data))

        sql1="INSERT INTO users(name, email, username, password)
VALUES(?,?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,name)
        ibm_db.bind_param(stmt1,2,email)
        ibm_db.bind_param(stmt1,3,username)
        ibm_db.bind_param(stmt1,4,password)
        ibm_db.execute(stmt1)
        #for flash messages taking parameter and the category of message to be
flashed
        flash("You are now registered and can log in", "success")
        #when registration is successful redirect to home
        return redirect(url_for('login'))
    return render_template('register.html', form = form)


#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        #Get form fields
        username = request.form['username']
        password_candidate = request.form['password']

        sql1="Select * from users where username = ?"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,username)
        result=ibm_db.execute(stmt1)
        d=ibm_db.fetch_assoc(stmt1)
        if result > 0:
            #Get the stored hash
            data = d
            password = data['PASSWORD']

            #compare passwords
            if sha256_crypt.verify(password_candidate, password):
                #Passed
                session['logged_in'] = True
                session['username'] = username

                flash("you are now logged in","success")
                return redirect(url_for('dashboard'))
            else:
                error = 'Invalid Login'
                return render_template('login.html', error=error)
            #Close connection
            cur.close()
        else:
            error = 'Username not found'
            return render_template('login.html', error=error)
    return render_template('login.html')

#check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login','danger')
            return redirect(url_for('login'))
    return wrap
```

```
#Logout
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return redirect(url_for('login'))

#Dashboard
@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql2="SELECT product_id, location_id, qty FROM product_balance"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)


    products=[]
    row = ibm_db.fetch_assoc(stmt2)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products=tuple(products)

    locations=[]
    row2 = ibm_db.fetch_assoc(stmt3)
    while(row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations=tuple(locations)

    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

    if result>0:
        return render_template('dashboard.html', products = products, locations = locs)
    else:
        msg='No products found'
        return render_template('dashboard.html', msg=msg)

#Product Form Class
class ProductForm(Form):
    product_id = StringField('Product ID', [validators.Length(min=1, max=200)])
    product_cost = StringField('Product Cost', [validators.Length(min=1, max=200)])
    product_num = StringField('Product Num', [validators.Length(min=1, max=200)])

#Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
    form = ProductForm(request.form)
    if request.method == 'POST' and form.validate():
        product_id = form.product_id.data
        product_cost = form.product_cost.data
        product_num = form.product_num.data
        sql1="INSERT INTO products(product_id, product_cost, product_num)
VALUES(?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,product_id)
        ibm_db.bind_param(stmt1,2,product_cost)
        ibm_db.bind_param(stmt1,3,product_num)
        ibm_db.execute(stmt1)

        flash("Product Added", "success")

        return redirect(url_for('products'))

    return render_template('add_product.html', form=form)

#Edit Product
@app.route('/edit_product/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_product(id):
    sql1="Select * from products where product_id = ?"
    stmt1 = ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1,1,id)
    result=ibm_db.execute(stmt1)
    product=ibm_db.fetch_assoc(stmt1)
    print(product)
    #Get form
    form = ProductForm(request.form)

    #populate product form fields
    form.product_id.data = product['PRODUCT_ID']
    form.product_cost.data = str(product['PRODUCT_COST'])
    form.product_num.data = str(product['PRODUCT_NUM'])
    if request.method == 'POST' and form.validate():
        product_id = request.form['product_id']
        product_cost = request.form['product_cost']
        product_num = request.form['product_num']

        sql2="UPDATE products SET product_id=?,product_cost=?,product_num=? WHERE product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,product_id)
        ibm_db.bind_param(stmt2,2,product_cost)
        ibm_db.bind_param(stmt2,3,product_num)
        ibm_db.bind_param(stmt2,4,id)
        ibm_db.execute(stmt2)

        flash("Product Updated", "success")

        return redirect(url_for('products'))

    return render_template('edit_product.html', form=form)

#Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in
def delete_product(id):

    sql2="DELETE FROM products WHERE product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Product Deleted", "success")
    return redirect(url_for('products'))

#Location Form Class
class LocationForm(Form):
    location_id = StringField('Location ID', [validators.Length(min=1, max=200)])

#Add Location
@app.route('/add_location', methods=['GET', 'POST'])
@is_logged_in
def add_location():
    form = LocationForm(request.form)
    if request.method == 'POST' and form.validate():
        location_id = form.location_id.data

        sql2="INSERT into locations VALUES(?)"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
        ibm_db.execute(stmt2)

        flash("Location Added", "success")

        return redirect(url_for('locations'))

    return render_template('add_location.html', form=form)

#Edit Location
@app.route('/edit_location/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_location(id):
    sql2="SELECT * FROM locations where location_id = ?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    result=ibm_db.execute(stmt2)
    location=ibm_db.fetch_assoc(stmt2)
    #Get form
    form = LocationForm(request.form)
    print(location)

    #populate article form fields
    form.location_id.data = location['LOCATION_ID']

    if request.method == 'POST' and form.validate():
        location_id = request.form['location_id']

        sql2="UPDATE locations SET location_id=? WHERE location_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
        ibm_db.bind_param(stmt2,2,id)
        ibm_db.execute(stmt2)

        flash("Location Updated", "success")

        return redirect(url_for('locations'))

    return render_template('edit_location.html', form=form)

#Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in
def delete_location(id):
    sql2="DELETE FROM locations WHERE location_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Location Deleted", "success")

    return redirect(url_for('locations'))

#Product Movement Form Class
class ProductMovementForm(Form):
    from_location = SelectField('From Location', choices=[])
    to_location = SelectField('To Location', choices=[])
    product_id = SelectField('Product ID', choices=[])
    qty = IntegerField('Quantity')

class CustomError(Exception):
    pass

#Add Product Movement
@app.route('/add_product_movements', methods=['GET', 'POST'])
@is_logged_in
def add_product_movements():
    form = ProductMovementForm(request.form)

    sql2="SELECT product_id FROM products"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)


    products=[]
```

```
row = ibm_db.fetch_assoc(stmt2)
while(row):
    products.append(row)
    row = ibm_db.fetch_assoc(stmt2)
products=tuple(products)

locations=[]
row2 = ibm_db.fetch_assoc(stmt3)
while(row2):
    locations.append(row2)
    row2 = ibm_db.fetch_assoc(stmt3)
locations=tuple(locations)

prods = []
for p in products:
    prods.append(list(p.values())[0])
locs = []
for i in locations:
    locs.append(list(i.values())[0])

form.from_location.choices = [(l,l) for l in locs]
form.from_location.choices.append(("Main Inventory","Main Inventory"))
form.to_location.choices = [(l,l) for l in locs]
form.to_location.choices.append(("Main Inventory","Main Inventory"))
form.product_id.choices = [(p,p) for p in prods]

if request.method == 'POST' and form.validate():
    from_location = form.from_location.data
    to_location = form.to_location.data
    product_id = form.product_id.data
    qty = form.qty.data
```

```
        if from_location==to_location:
            raise CustomError("Please Give different From and To Locations!!")


        elif from_location=="Main Inventory":
            sql2="SELECT * from product_balance where location_id=? and
product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,to_location)
            ibm_db.bind_param(stmt2,2,product_id)
            result=ibm_db.execute(stmt2)
            result=ibm_db.fetch_assoc(stmt2)
            print("-----------------")
            print(result)
            print("-----------------")
            app.logger.info(result)
            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity + qty

                    sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)

                    sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"
```

```
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,from_location)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.bind_param(stmt2,4,qty)
                    ibm_db.execute(stmt2)
            else:

                sql2="INSERT into product_balance(product_id, location_id, qty)
values(?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,product_id)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,qty)
                ibm_db.execute(stmt2)

                sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,from_location)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.bind_param(stmt2,4,qty)
                ibm_db.execute(stmt2)


            sql = "select product_num from products where product_id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt,1,product_id)
            current_num=ibm_db.execute(stmt)
            current_num = ibm_db.fetch_assoc(stmt)
            sql2="Update products set product_num=? where product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-qty)
            ibm_db.bind_param(stmt2,2,product_id)
            ibm_db.execute(stmt2)

            alert_num=current_num['PRODUCT_NUM']-qty

            if(alert_num<=0):
                alert("Please update the quantity of the product {}, Atleast {} number of
pieces must be added to finish the pending Product
Movements!".format(product_id,-alert_num))
        elif to_location=="Main Inventory":
            sql2="SELECT * from product_balance where location_id=? and
product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,product_id)
            result=ibm_db.execute(stmt2)
            result=ibm_db.fetch_assoc(stmt2)

            app.logger.info(result)
            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity - qty

                    sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)

                    sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,from_location)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.bind_param(stmt2,4,qty)
                    ibm_db.execute(stmt2)

                    flash("Product Movement Added", "success")


                    sql = "select product_num from products where product_id=?"
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt,1,product_id)
                    current_num=ibm_db.execute(stmt)
                    current_num = ibm_db.fetch_assoc(stmt)

                    sql2="Update products set product_num=? where product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']+qty)
                    ibm_db.bind_param(stmt2,2,product_id)
                    ibm_db.execute(stmt2)

                    alert_num=q
                    if(alert_num<=0):
                        alert("Please Add {} number of {} to {} warehouse!".format(-
q,product_id,from_location))
            else:
                raise CustomError("There is no product named {} in {}.".format(product_id,from_location))


        else: #will be executed if both from_location and to_location are specified
            f=0
            sql = "SELECT * from product_balance where location_id=? and product_id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt,1,from_location)
            ibm_db.bind_param(stmt,2,product_id)
            result=ibm_db.execute(stmt)
            result = ibm_db.fetch_assoc(stmt)

            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity - qty

                    sql2="UPDATE product_balance set qty=? where location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,from_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)
                    f=1

                    alert_num=q
                    if(alert_num<=0):
                        alert("Please Add {} number of {} to {} warehouse!".format(-q,product_id,from_location))
            else:
                raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))

            if(f==1):
                sql = "SELECT * from product_balance where location_id=? and
product_id=?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt,1,to_location)
                ibm_db.bind_param(stmt,2,product_id)
                result=ibm_db.execute(stmt)
                result = ibm_db.fetch_assoc(stmt)

                if result!=False:
                    if(len(result))>0:
                        Quantity = result["QTY"]
                        q = Quantity + qty

                        sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                        stmt2 = ibm_db.prepare(conn, sql2)
                        ibm_db.bind_param(stmt2,1,q)
                        ibm_db.bind_param(stmt2,2,to_location)
                        ibm_db.bind_param(stmt2,3,product_id)
                        ibm_db.execute(stmt2)

                else:
                    sql2="INSERT into product_balance(product_id, location_id, qty)
values(?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,product_id)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,qty)
                    ibm_db.execute(stmt2)
                sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,from_location)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.bind_param(stmt2,4,qty)
                ibm_db.execute(stmt2)

                flash("Product Movement Added", "success")

        render_template('products.html',form=form)



        return redirect(url_for('product_movements'))

    return render_template('add_product_movements.html', form=form)

#Delete Product Movements
@app.route('/delete_product_movements/<string:id>', methods=['POST'])
@is_logged_in
def delete_product_movements(id):

    sql2="DELETE FROM productmovements WHERE movement_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Product Movement Deleted", "success")
    return redirect(url_for('product_movements'))

if _name_ == '_main_':
    app.secret_key = "secret123"
    #when the debug mode is on, we do not need to restart the server again and
again
    app.run(debug=True)
```

7.2 Feature 2

Login

```
{% extends 'layout.html' %}

{% block body %}
<h1>Login</h1>
<style>
    body {
      background-color:#dcdcdc ;
    }
    </style>
<form method="POST" action="">
    <div class="form-group">
        <label>Username</label>
        <input type="text" name="username" class="form-control"
value={{request.form.username}}>
    </div>
    <div class="form-group">
            <label>Password</label>
            <input type="password" name="password" class="form-control"
value={{request.form.password}}>
        </div>
    <p><button type="submit" class="btn btn-primary"
```

```
value="Submit">Submit</button></p>
</form>
{% endblock %}
```

DASHBOARD

```
identifier{% extends 'layout.html' %}

{% block body %}
    <h1>Dashboard <small>Welcome {{session.username}}</small></h1>
    <hr>
      {% for location in locations %}
      <div>
       <h3 class="mt-4 text-primary" >{{location}}</h3>
       <table class="table table-striped">
           <thead>
               <tr>
                   <th>Product</th>
                   <th>Warehouse</th>
                   <th>Qty</th>
               </tr>
           </thead>
           <tbody>
                   {% for product in products %}
                      {% if product.LOCATION_ID == location %}
                  <tr>
                   <td>{{product.PRODUCT_ID}}</td>
                   <td>{{product.LOCATION_ID}}</td>
                   <td>{{product.QTY}}</td>
                  </tr>
                      {% endif %}
                  {% endfor %}
```

```
            </tbody>
        </table>
        <hr>
        </div>
    {% endfor %}
{% endblock %}
```

8. TESTING

The purpose of software testing is to access or evaluate the capabilities or attributes of a software program’s ability to adequately meet the applicable standards and application need. Testing does not ensure quality and the purpose of testing is not to find bugs. Testing can be verification and validation or reliability estimation. The primary objective if testing includes:

- To identifying defects in the application.
- The most important role of testing is simply to provide information to check the proper working of the application while inserting updating and deleting the entry of the products.

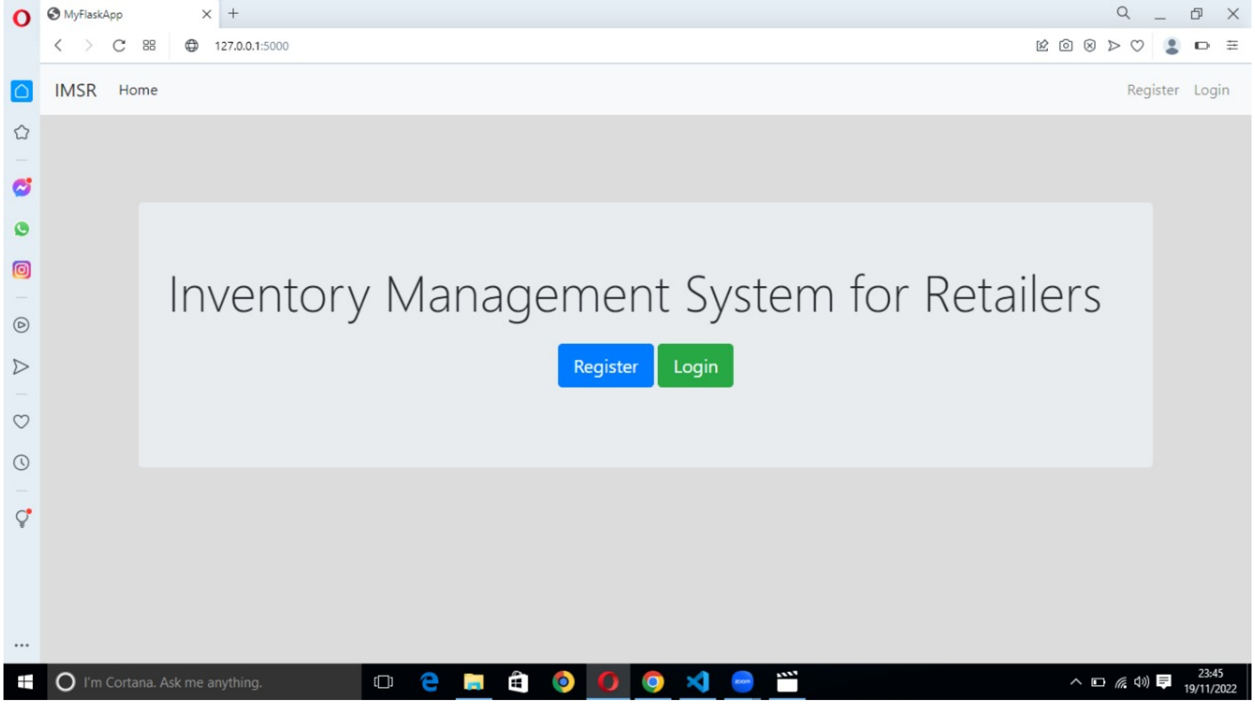
8.1 Type of Testing

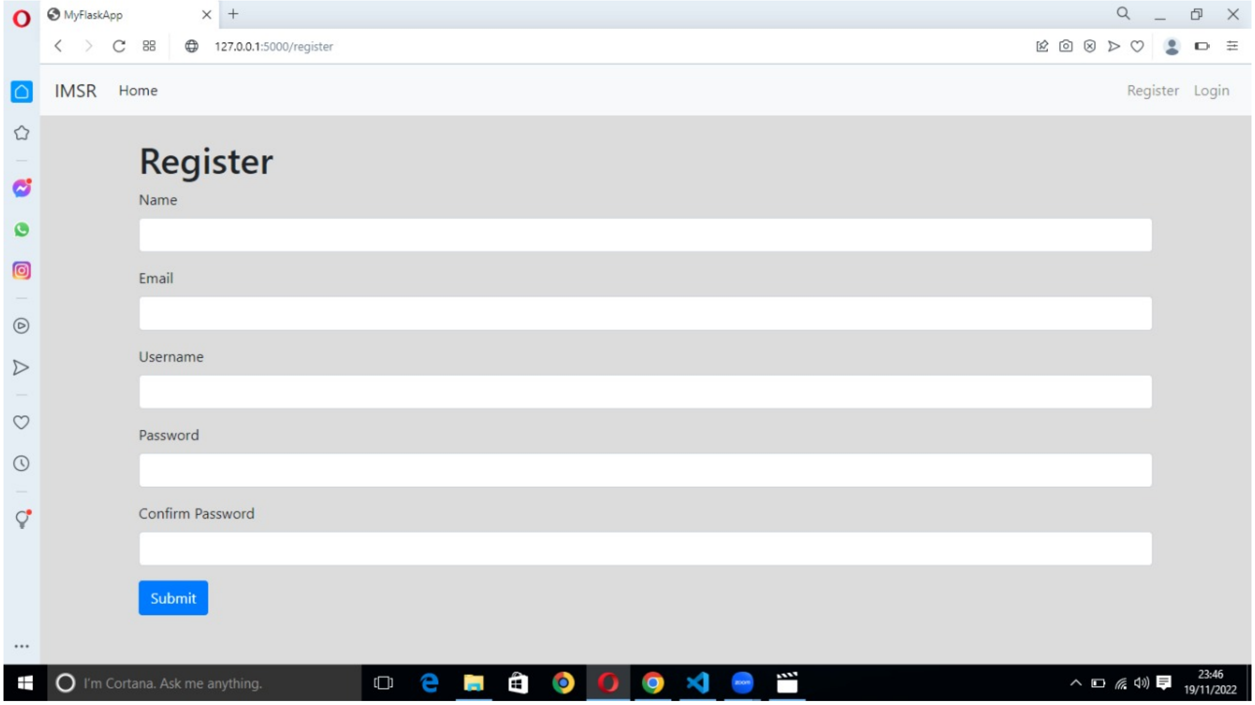
We have used one type of testing to ensure the error free features of our software application:

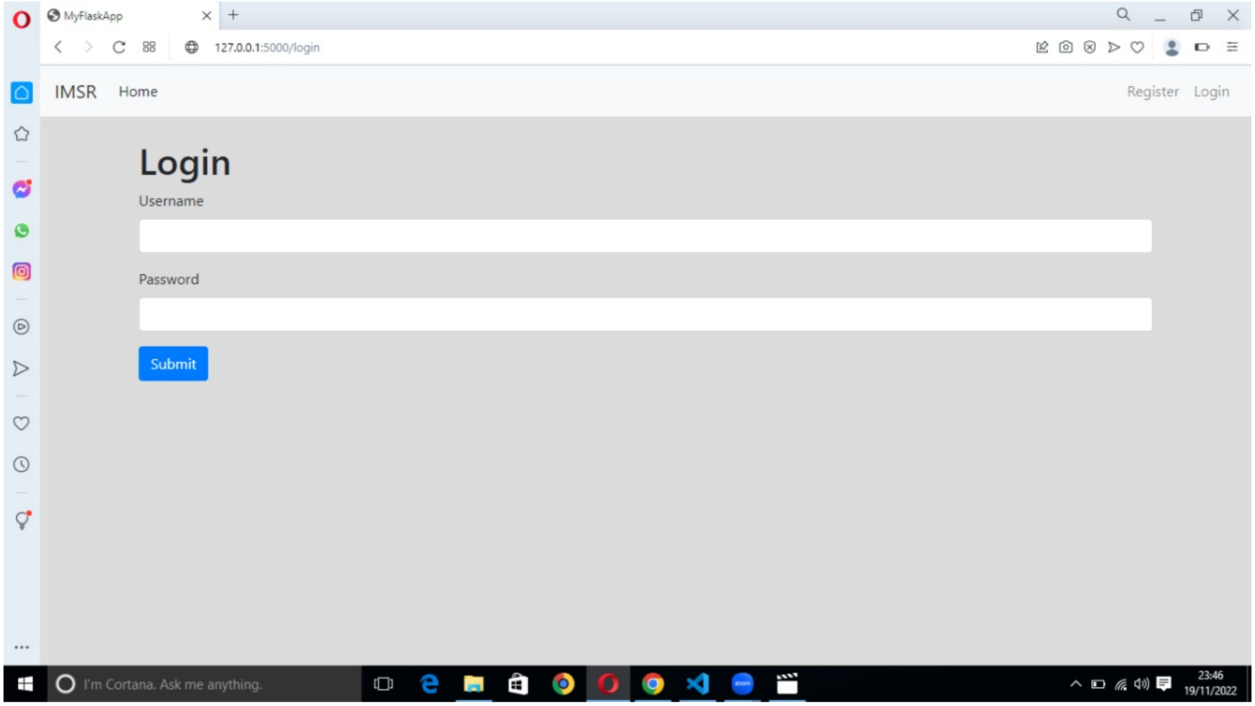
8.2 User Acceptance Testing

This type of testing is the testing of individual software components. It is typically done by the programmer and not by the testers. It requires details information and knowledge about the internal program design and code to perform this. During unit testing, we carried out various testing task such as the reflection of the unit data on database and its interface. Various types of bugs associated with the component were identified and fixed. We use various functional keys to test our software. In our software unit testing is concerned with the stock units, opening stock units and product units validation as well as the validation of product units.

9. RESULTS



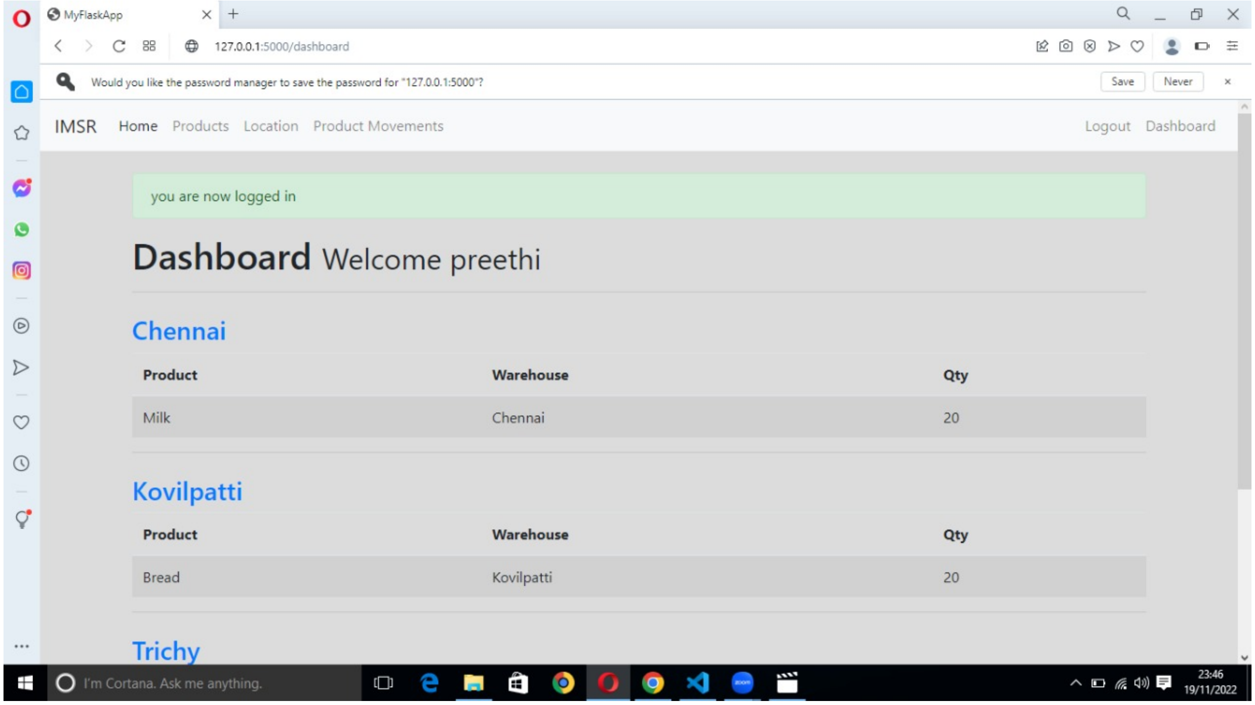




Login

Username

Password



Would you like the password manager to save the password for "127.0.0.1:5000"?

you are now logged in

Dashboard Welcome preethi

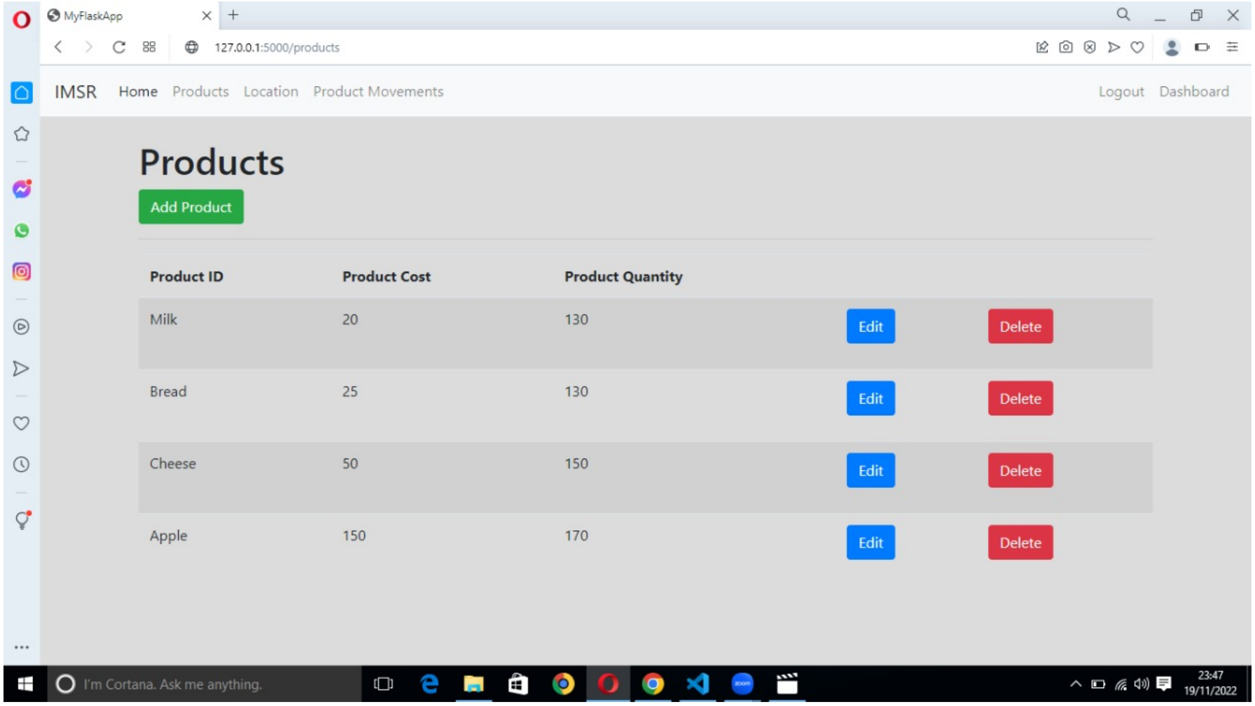
Chennai

Product	Warehouse	Qty
Milk	Chennai	20

Kovilpatti

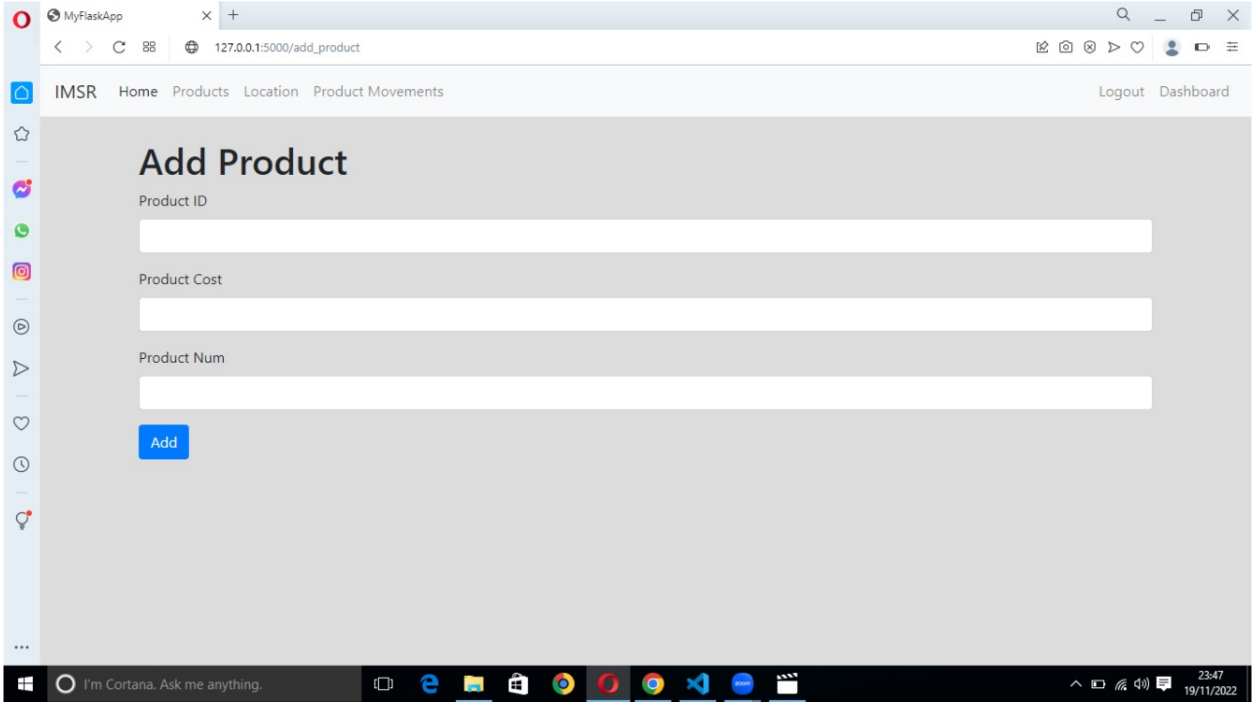
Product	Warehouse	Qty
Bread	Kovilpatti	20

Trichy



Products

Product ID	Product Cost	Product Quantity
Milk	20	130
Bread	25	130
Cheese	50	150
Apple	150	170

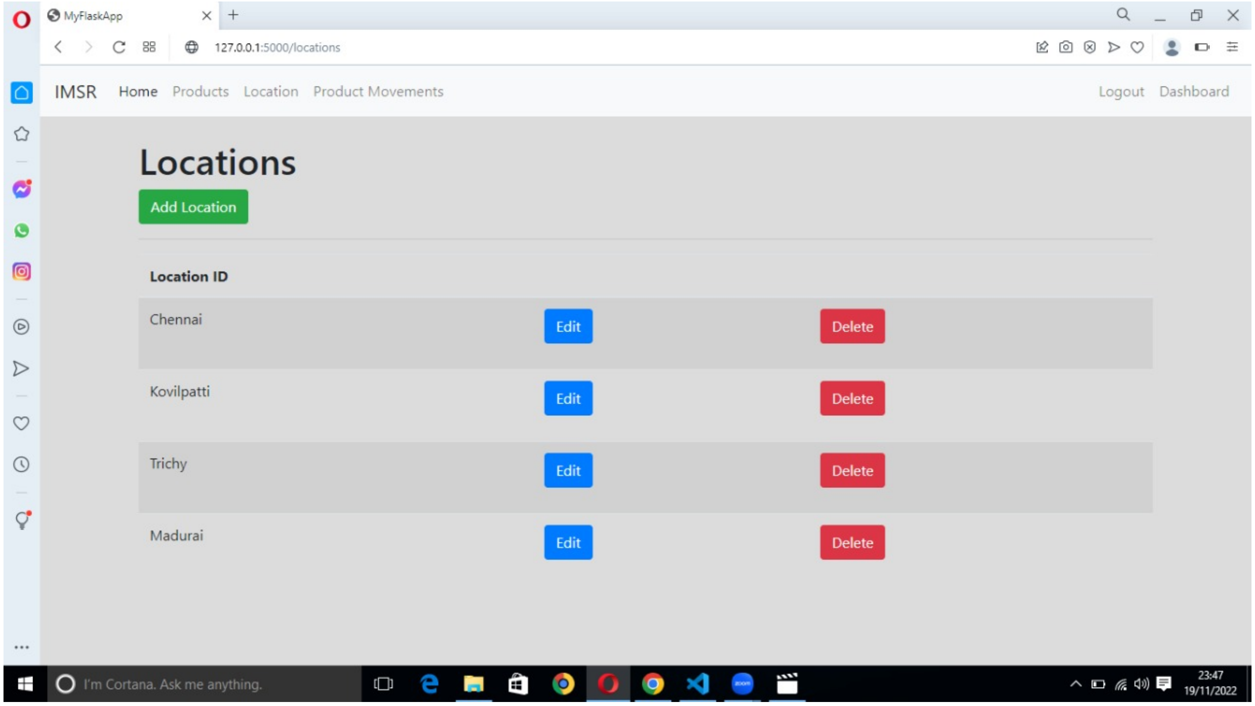


Add Product

Product ID

Product Cost

Product Num



MyFlaskApp

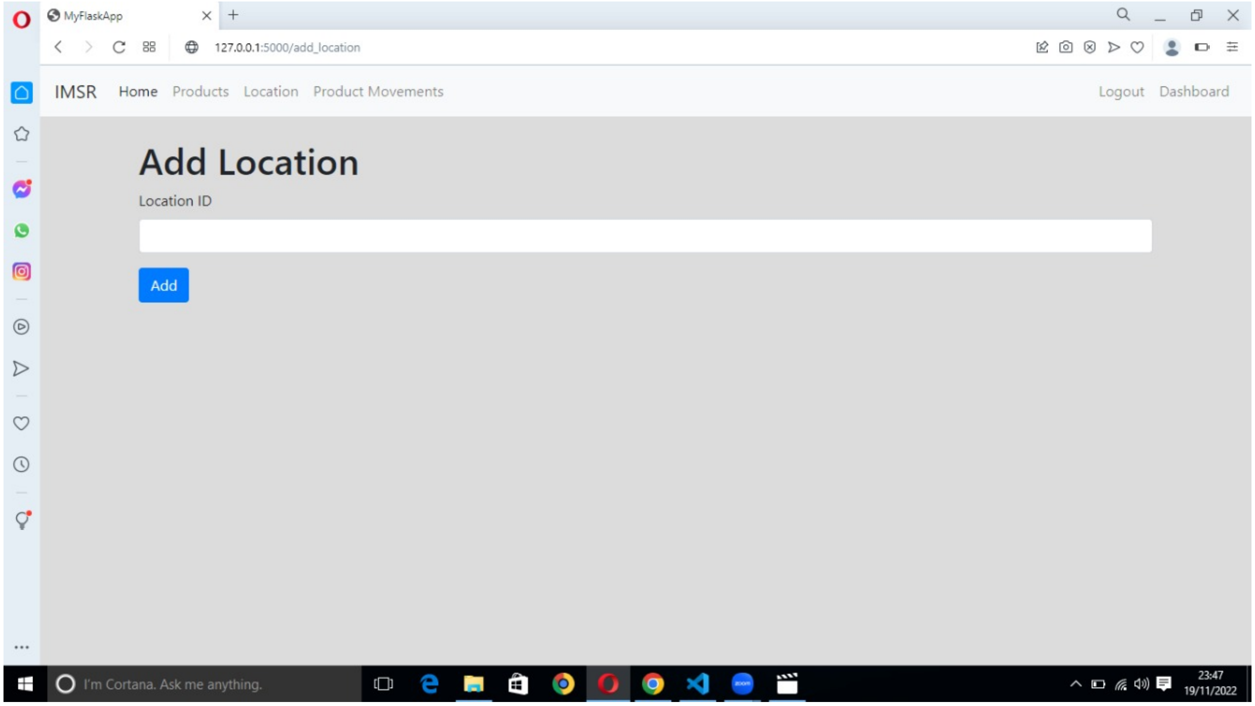
127.0.0.1:5000/locations

IMSR Home Products Location Product Movements Logout Dashboard

Locations

Add Location

Location ID		
Chennai	Edit	Delete
Kovilpatti	Edit	Delete
Trichy	Edit	Delete
Madurai	Edit	Delete



MyFlaskApp

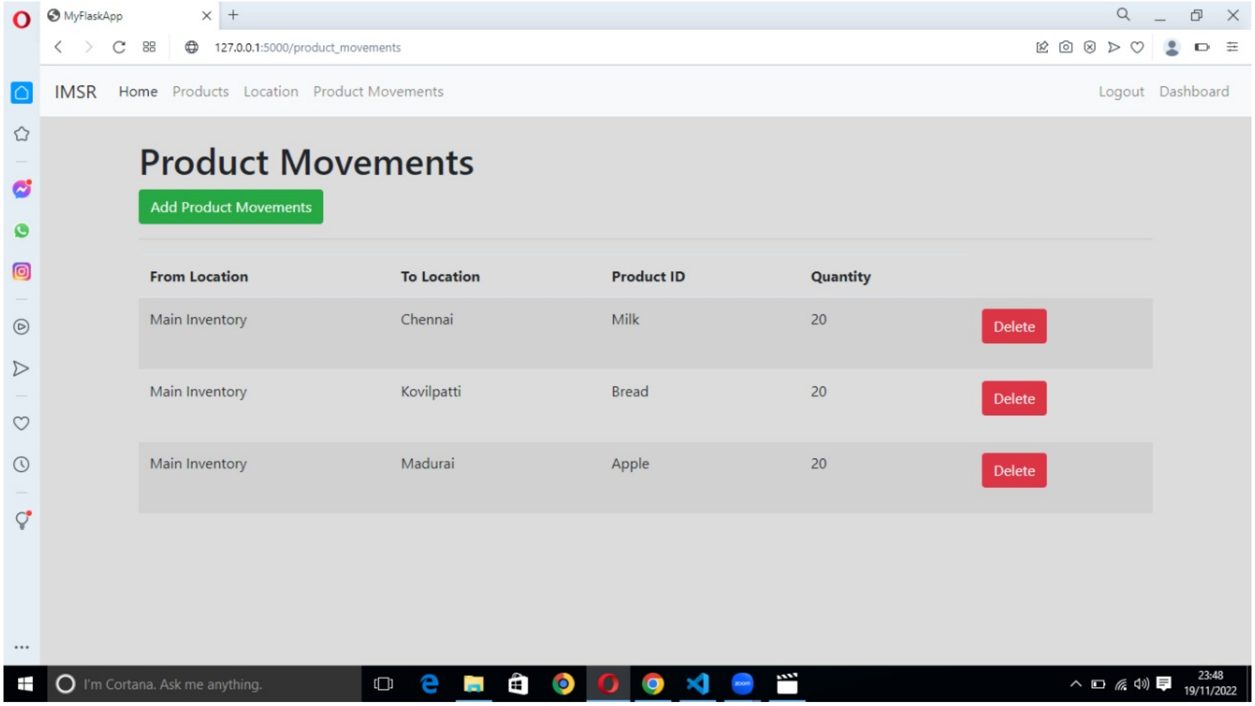
127.0.0.1:5000/add_location

IMSR Home Products Location Product Movements Logout Dashboard

Add Location

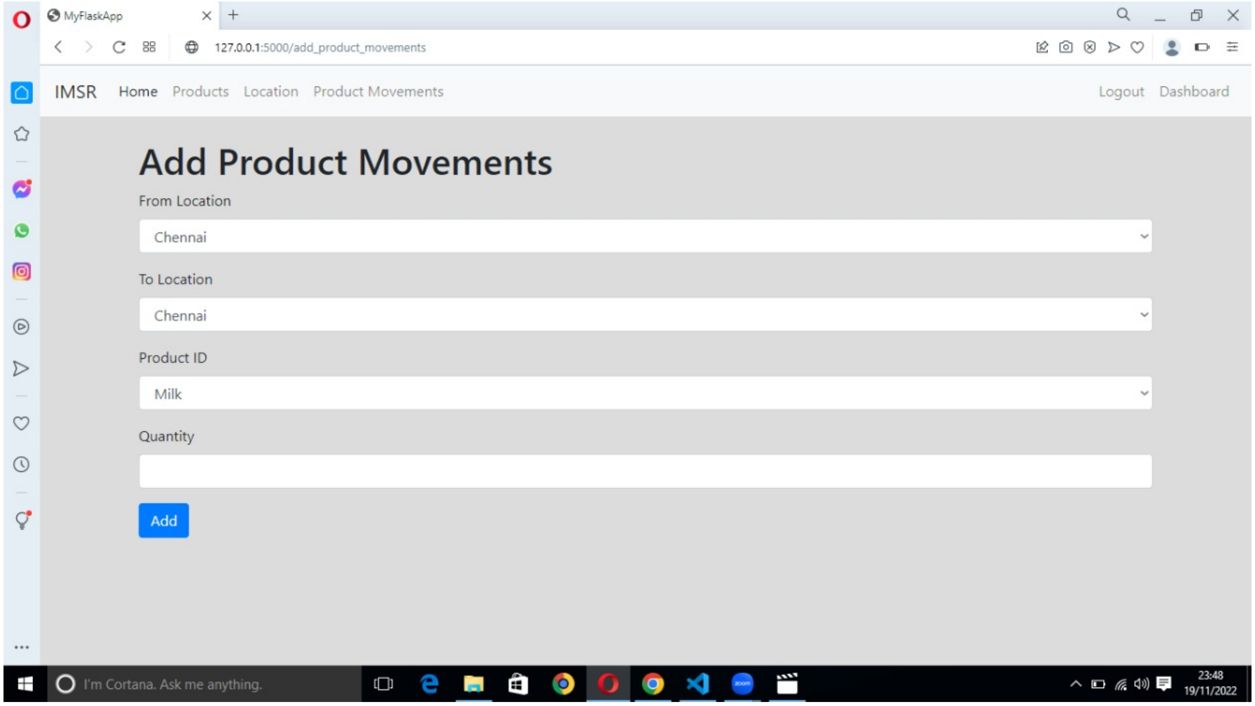
Location ID

Add



Product Movements

From Location	To Location	Product ID	Quantity	
Main Inventory	Chennai	Milk	20	Delete
Main Inventory	Kovilpatti	Bread	20	Delete
Main Inventory	Madurai	Apple	20	Delete



Add Product Movements

From Location: Chennai

To Location: Chennai

Product ID: Milk

Quantity:

9.1 Performance Metrics

User satisfaction

For the ease of use for the end users, we have developed this website with a modular approach and clean UI. The end user can easily use the website with the modules. Also this website is responsive which makes it easier to use across all the devices.

Average response time

Since the website was made using Flask(Python) and it uses the ibm cloud to get the details as go, The response time is negligible.

Average Request time

The user inputs the data needed to process in the ibm cloud and then the api needs to fetch data, the processing speed at the ibm cloud will determine the request time or waiting time, The request time is low.

Error rate

All the data was taken from ibm datasets and we used comprehensive data pre-processing to avoid and eliminate the errors. This app is almost error free.

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES OF INVENTORY MANAGEMENT

1. It helps to maintain the right amount of stocks: It seeks to maintain an equilibrium point where your inventory is working at a maximum efficiency and you do not have to have many stocks or too few stocks at hand at any particular point in time.

2.Increased information transparency: a good inventory management helps to keep the flow of information transparent.

3. It leads to a more organized warehouse: with the aid of a good inventory management system, you can easily organize your warehouse.

4. It saves time and money: an effective inventory management system can translate to time and money saved on the part of the business.

5. Improves efficiency and productivity: inventory management devices like bar code scanners and inventory management software can help to greatly increase the efficiency and productivity of a business.

6. Schedule maintenance: once you get hold of a new appliance, you can begin to schedule routine and preventative maintenance, issue work order to your staff and track that the maintenance was actually carried out.

7. Flexibility: a good inventory management strategy will allow the manager to be flexible and adapt to situations as they arise.

DISADVANTAGE:

System crash

One of the biggest problems with any computerized system is the potential for a system crash. A corrupt hard drive, power outages and other technical issues can result in the loss of needed data. At the least, businesses are interrupted when they are unable to access data they need. Business owners should back up data regularly to protect against data loss.

Malicious Hacks

Hackers look for any way to get company or consumer information. An inventory system connected to point-of-sale devices and accounting is a valuable resource to hack into in search of potential financial information or personal details of owners, vendors or clients. Updating firewalls and anti-virus software can mitigate this potential issue.

Reduced Physical Audits

When everything is automated, it is easy to forego time-consuming physical inventory audits. They may no longer seem necessary when the computers are doing their work. However, it is important to continue to do regular audits to identify loss such as spoilage or breakage. Audits also help business owners identify potential internal theft and manipulation of the computerized inventory system.

11. CONCLUSION

To conclude, Inventory Management System is a simple desktop based application basically suitable for small organization. It has every basic items which are used for the small organization. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also provides a simple report on daily basis to know the daily sales and purchase details. This application matches for small organization where there small limited if godowns. Through it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organization.

12. FUTURE SCOPE

The scope of an inventory system can cover many needs, including valuing the inventory, measuring the change in inventory and planning for future inventory levels. The value of the inventory at the end of each period provides a basis for financial reporting on the balance sheet. Measuring the change in inventory allows the company to determine the cost of inventory sold during the period. This allows the company to plan for future inventory needs.

13. APPENDIX

Source Code

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField,
IntegerField
import ibm_db
from passlib.hash import sha256_crypt
from functools import wraps
from sendgrid import *

#creating an app instance
app = Flask(_name_)

app.secret_key='a'
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=824dfd4d-99de-440d-9991-
629c01b3832d.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30119;SECURITY=S
SL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=tyt27664;PWD=hrAJhoiC3znla3rh;",
",")

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

#Products
@app.route('/products')
def products():
    sql = "SELECT * FROM products"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)

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

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

```
#Locations
@app.route('/locations')
def locations():

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

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

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

#Product Movements
@app.route('/product_movements')
def product_movements():

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

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

    if result>0:
        return render_template('product_movements.html', movements = movements)
    else:
        msg='No product movements found'
        return render_template('product_movements.html', msg=msg)

#Register Form Class
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
    username = StringField('Username', [validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6, max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')

#user register
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password = sha256_crypt.encrypt(str(form.password.data))

        sql1="INSERT INTO users(name, email, username, password) VALUES(?,?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,name)
        ibm_db.bind_param(stmt1,2,email)
        ibm_db.bind_param(stmt1,3,username)
        ibm_db.bind_param(stmt1,4,password)
        ibm_db.execute(stmt1)
        #for flash messages taking parameter and the category of message to be flashed
        flash("You are now registered and can log in", "success")
        #when registration is successful redirect to home
        return redirect(url_for('login'))
    return render_template('register.html', form = form)


#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        #Get form fields
        username = request.form['username']
        password_candidate = request.form['password']

        sql1="Select * from users where username = ?"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,username)
        result=ibm_db.execute(stmt1)
        d=ibm_db.fetch_assoc(stmt1)
        if result > 0:
            #Get the stored hash
            data = d
            password = data['PASSWORD']

            #compare passwords
            if sha256_crypt.verify(password_candidate, password):
                #Passed
                session['logged_in'] = True
                session['username'] = username

                flash("you are now logged in","success")
                return redirect(url_for('dashboard'))
            else:
                error = 'Invalid Login'
                return render_template('login.html', error=error)
            #Close connection
            cur.close()
        else:
            error = 'Username not found'
            return render_template('login.html', error=error)
    return render_template('login.html')

#check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login','danger')
            return redirect(url_for('login'))
    return wrap

#Logout
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return redirect(url_for('login'))

#Dashboard
@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql2="SELECT product_id, location_id, qty FROM product_balance"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)
    products=[]
    row = ibm_db.fetch_assoc(stmt2)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products=tuple(products)

    locations=[]
    row2 = ibm_db.fetch_assoc(stmt3)
    while(row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations=tuple(locations)

    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

    if result>0:
        return render_template('dashboard.html', products = products, locations = locs)
    else:
        msg='No products found'
        return render_template('dashboard.html', msg=msg)

#Product Form Class
class ProductForm(Form):
    product_id = StringField('Product ID', [validators.Length(min=1, max=200)])
    product_cost = StringField('Product Cost', [validators.Length(min=1, max=200)])
    product_num = StringField('Product Num', [validators.Length(min=1, max=200)])

#Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
    form = ProductForm(request.form)
    if request.method == 'POST' and form.validate():
        product_id = form.product_id.data
        product_cost = form.product_cost.data
        product_num = form.product_num.data

        sql1="INSERT INTO products(product_id, product_cost, product_num) VALUES(?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,product_id)
        ibm_db.bind_param(stmt1,2,product_cost)
        ibm_db.bind_param(stmt1,3,product_num)
        ibm_db.execute(stmt1)

        flash("Product Added", "success")

        return redirect(url_for('products'))

    return render_template('add_product.html', form=form)

#Edit Product
@app.route('/edit_product/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_product(id):
    sql1="Select * from products where product_id = ?"
    stmt1 = ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1,1,id)
    result=ibm_db.execute(stmt1)
    product=ibm_db.fetch_assoc(stmt1)
    print(product)
    #Get form
    form = ProductForm(request.form)

    #populate product form fields
    form.product_id.data = product['PRODUCT_ID']
    form.product_cost.data = str(product['PRODUCT_COST'])
    form.product_num.data = str(product['PRODUCT_NUM'])

    if request.method == 'POST' and form.validate():
        product_id = request.form['product_id']
        product_cost = request.form['product_cost']
        product_num = request.form['product_num']

        sql2="UPDATE products SET product_id=?,product_cost=?,product_num=? WHERE product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,product_id)
        ibm_db.bind_param(stmt2,2,product_cost)
        ibm_db.bind_param(stmt2,3,product_num)
        ibm_db.bind_param(stmt2,4,id)
        ibm_db.execute(stmt2)

        flash("Product Updated", "success")

        return redirect(url_for('products'))

    return render_template('edit_product.html', form=form)

#Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in
def delete_product(id):

    sql2="DELETE FROM products WHERE product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Product Deleted", "success")

    return redirect(url_for('products'))

#Location Form Class
class LocationForm(Form):
    location_id = StringField('Location ID', [validators.Length(min=1, max=200)])

#Add Location
@app.route('/add_location', methods=['GET', 'POST'])
```

```
@is_logged_in
def add_location():
    form = LocationForm(request.form)
    if request.method == 'POST' and form.validate():
        location_id = form.location_id.data

        sql2="INSERT into locations VALUES(?)"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
        ibm_db.execute(stmt2)

        flash("Location Added", "success")

        return redirect(url_for('locations'))

    return render_template('add_location.html', form=form)

#Edit Location
@app.route('/edit_location/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_location(id):
    sql2="SELECT * FROM locations where location_id = ?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    result=ibm_db.execute(stmt2)
    location=ibm_db.fetch_assoc(stmt2)
    #Get form
    form = LocationForm(request.form)
    print(location)

    #populate article form fields
    form.location_id.data = location['LOCATION_ID']

    if request.method == 'POST' and form.validate():
        location_id = request.form['location_id']

        sql2="UPDATE locations SET location_id=? WHERE location_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
        ibm_db.bind_param(stmt2,2,id)
        ibm_db.execute(stmt2)

        flash("Location Updated", "success")

        return redirect(url_for('locations'))

    return render_template('edit_location.html', form=form)

#Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in
def delete_location(id):
    sql2="DELETE FROM locations WHERE location_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Location Deleted", "success")

    return redirect(url_for('locations'))

#Product Movement Form Class
class ProductMovementForm(Form):
    from_location = SelectField('From Location', choices=[])
    to_location = SelectField('To Location', choices=[])
    product_id = SelectField('Product ID', choices=[])
    qty = IntegerField('Quantity')

class CustomError(Exception):
    pass

#Add Product Movement
@app.route('/add_product_movements', methods=['GET', 'POST'])
@is_logged_in
```

```
def add_product_movements():
    form = ProductMovementForm(request.form)

    sql2="SELECT product_id FROM products"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)


    products=[]
    row = ibm_db.fetch_assoc(stmt2)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products=tuple(products)

    locations=[]
    row2 = ibm_db.fetch_assoc(stmt3)
    while(row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations=tuple(locations)

    prods = []
    for p in products:
        prods.append(list(p.values())[0])
    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

    form.from_location.choices = [(l,l) for l in locs]
    form.from_location.choices.append(("Main Inventory","Main Inventory"))
    form.to_location.choices = [(l,l) for l in locs]
    form.to_location.choices.append(("Main Inventory","Main Inventory"))
    form.product_id.choices = [(p,p) for p in prods]

    if request.method == 'POST' and form.validate():
        from_location = form.from_location.data
        to_location = form.to_location.data
        product_id = form.product_id.data
        qty = form.qty.data



        if from_location==to_location:
            raise CustomError("Please Give different From and To Locations!!")



        elif from_location=="Main Inventory":
            sql2="SELECT * from product_balance where location_id=? and product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,to_location)
            ibm_db.bind_param(stmt2,2,product_id)
            result=ibm_db.execute(stmt2)
            result=ibm_db.fetch_assoc(stmt2)
            print("-----------------")
            print(result)
            print("-----------------")
            app.logger.info(result)
            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity + qty

                    sql2="UPDATE product_balance set qty=? where location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)

                    sql2="INSERT into productmovements(from_location, to_location, product_id, qty)
```

```
VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,from_location)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.bind_param(stmt2,4,qty)
                ibm_db.execute(stmt2)
        else:

            sql2="INSERT into product_balance(product_id, location_id, qty) values(?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,product_id)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,qty)
            ibm_db.execute(stmt2)

            sql2="INSERT into productmovements(from_location, to_location, product_id, qty)
VALUES(?, ?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.bind_param(stmt2,4,qty)
            ibm_db.execute(stmt2)


        sql = "select product_num from products where product_id=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,product_id)
        current_num=ibm_db.execute(stmt)
        current_num = ibm_db.fetch_assoc(stmt)

        sql2="Update products set product_num=? where product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-qty)
        ibm_db.bind_param(stmt2,2,product_id)
        ibm_db.execute(stmt2)
```

```
            alert_num=current_num['PRODUCT_NUM']-qty

            if(alert_num<=0):
                alert("Please update the quantity of the product {}, Atleast {} number of pieces must
be added to finish the pending Product Movements!".format(product_id,-alert_num))
        elif to_location=="Main Inventory":
            sql2="SELECT * from product_balance where location_id=? and product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,product_id)
            result=ibm_db.execute(stmt2)
            result=ibm_db.fetch_assoc(stmt2)

            app.logger.info(result)
            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity - qty

                    sql2="UPDATE product_balance set qty=? where location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)

                    sql2="INSERT into productmovements(from_location, to_location, product_id, qty)
VALUES(?, ?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,from_location)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.bind_param(stmt2,4,qty)
                    ibm_db.execute(stmt2)

                    flash("Product Movement Added", "success")
                    sql = "select product_num from products where product_id=?"
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt,1,product_id)
                    current_num=ibm_db.execute(stmt)
                    current_num = ibm_db.fetch_assoc(stmt)

                    sql2="Update products set product_num=? where product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']+qty)
                    ibm_db.bind_param(stmt2,2,product_id)
                    ibm_db.execute(stmt2)

                    alert_num=q
                    if(alert_num<=0):
                        alert("Please Add {} number of {} to {} warehouse!".format(-
q,product_id,from_location))
            else:
                raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))


        else: #will be executed if both from_location and to_location are specified
            f=0
            sql = "SELECT * from product_balance where location_id=? and product_id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt,1,from_location)
            ibm_db.bind_param(stmt,2,product_id)
            result=ibm_db.execute(stmt)
            result = ibm_db.fetch_assoc(stmt)

            if result!=False:
                if(len(result))>0:
                    Quantity = result["QTY"]
                    q = Quantity - qty
                    sql2="UPDATE product_balance set qty=? where location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,q)
                    ibm_db.bind_param(stmt2,2,from_location)
                    ibm_db.bind_param(stmt2,3,product_id)
                    ibm_db.execute(stmt2)
                    f=1

                    alert_num=q
                    if(alert_num<=0):
                        alert("Please Add {} number of {} to {} warehouse!".format(-
q,product_id,from_location))

            else:
                raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))

            if(f==1):
                sql = "SELECT * from product_balance where location_id=? and product_id=?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt,1,to_location)
                ibm_db.bind_param(stmt,2,product_id)
                result=ibm_db.execute(stmt)
                result = ibm_db.fetch_assoc(stmt)

                if result!=False:
                    if(len(result))>0:
                        Quantity = result["QTY"]
                        q = Quantity + qty

                        sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                        stmt2 = ibm_db.prepare(conn, sql2)
                        ibm_db.bind_param(stmt2,1,q)
                        ibm_db.bind_param(stmt2,2,to_location)
                        ibm_db.bind_param(stmt2,3,product_id)
                        ibm_db.execute(stmt2)
                else:
                    sql2="INSERT into product_balance(product_id, location_id, qty) values(?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2,1,product_id)
                    ibm_db.bind_param(stmt2,2,to_location)
                    ibm_db.bind_param(stmt2,3,qty)
                    ibm_db.execute(stmt2)
                sql2="INSERT into productmovements(from_location, to_location, product_id, qty)
VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,from_location)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.bind_param(stmt2,4,qty)
                ibm_db.execute(stmt2)

                flash("Product Movement Added", "success")

        render_template('products.html',form=form)



        return redirect(url_for('product_movements'))

    return render_template('add_product_movements.html', form=form)

#Delete Product Movements
@app.route('/delete_product_movements/<string:id>', methods=['POST'])
@is_logged_in
def delete_product_movements(id):

    sql2="DELETE FROM productmovements WHERE movement_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Product Movement Deleted", "success")
    return redirect(url_for('product_movements'))

if _name_ == '_main_':
    app.secret_key = "secret123"
    #when the debug mode is on, we do not need to restart the server again and again
    app.run(debug=True)
```

GitHub Link

https://github.com/IBM-EPBL/IBM-Project-24352-165994178

Project Demo Link

https://drive.google.com/file/d/1r_3w-1FzJyes9fodE-tzz7HqX_RpNTsd/view?usp=share_link