Project Report

HX8001& Professional Readiness for Innovation, Employability & Entrepreneurship

Team ID	**TEAM ID - PNT2022TMID25323**
Project Name	**INVENTORY MANAGEMENT SYSTEM FOR RETAILERS**

Submitted by

DAVID NIXON RAJ D	**210819104017**
ANISHAARON S	**210819104009**
HEMANTH KUMAR M	**210819104028**
JENISH F	**210819104039**
JOHN WESLEY B	**210819104042**

of

BACHELOR OF ENGINEERING
IN COMPUTER SCIENCE AND ENGINEERING

KINGS ENGINEERING COLLEGE
Sriperumbudur
Irungattukottai, Tamil Nadu 602117

Contents

1. INTRODUCTION

1.1 Project Overview:

Retail Inventory Management is the process of ensuring 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. In today's more turbulent environment there is no longer any possibility of manufacturing and marketing acting independently of each other. It is now generally accepted that the need to understand and meet customer requirements is a prerequisite for survival. At the same time, in the search for improved cost competitiveness, manufacturing management has been the subject of massive renaissance. The last decade has seen the rapid introduction of flexible manufacturing systems, of new approaches to inventory based on materials requirement planning (MRP) and just in time (JIT) methods, and a sustained emphasis on quality. Equally there has been a growing recognition of the critical role that procurement plays in creating and sustaining competitive advantage as part of an integrated logistics process. In this scheme of things, logistics is therefore essentially an integrative concept that seeks to develop a system wide view of the firm. It is fundamentally a planning concept that seeks to create a framework through which the needs of the manufacturing strategy and plan, which in turn link into a strategy and plan for procurement.

1.2 Purpose:

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. In the industries there will be a competitor who will be a low-cost producer and will have greater sales volume in that sector. This is partly due to economies of scale, which enable fixed costs to spread over a greater volume but more particularly to the impact of the experience curve. It is possible to identify and predict improvements in the rate of output of workers as they become more skilled in the processes and tasks on which they work. Bruce Henderson extended this concept by demonstrating that all costs, not just production costs, would decline at a given rate as volume increased. This cost decline applies only to value added, i.e., costs other than bought in supplies. Traditionally it has been suggested that the main route to cost reduction was by gaining greater sales volume and there can be no doubt about the close linkage between relative market share and relative costs. However, it must also be recognized that logistics management can provide a multitude of ways to increase efficiency and productivity and hence contribute significantly to reduced unit costs.

2. LITERATURE SURVEY

2.1 Existing problem

Paper 1: The relationship between capacity utilization and inventory investment

Publication Year: 1957

Author: Abramovitz , Odiglian

They highlighted the relationship between capacity utilization and inventory investment. Existing stock of inventories was expected to adjust to the desired levels. Thus, the variable, existing stock of inventories, was essential to be negatively related with the desired stock. The result was that there is a positive relation among the ratio of inventory to sales and inventory investment. High ratio of stocks to sales in the past suggests the requirement of high levels of inventories in the past and promising high investment in inventories in the current period also.

Paper 2: Study on manufacture inventories

Publication Year: (1970)

Author: Krishnamurty and Sastry

It is the most comprehensive study on manufacturers' inventories. They used the CMI data and the consolidated balance sheet data of public limited companies published by the RBI, in order to analyse each of the major components, like the raw materials, goods-in-process and finished goods, for 21 industries over the period ranging from 1946-62. The study was a time series one although there were some inter-industry cross-section analyses that were carried out in the analysis. The Accelerator represented by change in sales, bank finance and short-term interest rate was found to be an important determinant. The utilisation of productive capacity and price anticipations was also found to be relevant in the study.

Paper 3:

Publication Year: (1972)

Author: George

It was the study on cross section analysis of balance sheet data of 52 public limited companies for the period of 1967- 70. Accelerator, internal and external finance variables were considered in the formulation of equations for raw materials including goods-in-process inventories. However, equations for finished goods inventories conceive only output variables. Deliberation was given on accelerator and external finance variables

2.2 PROBLEM STATEMENT

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.

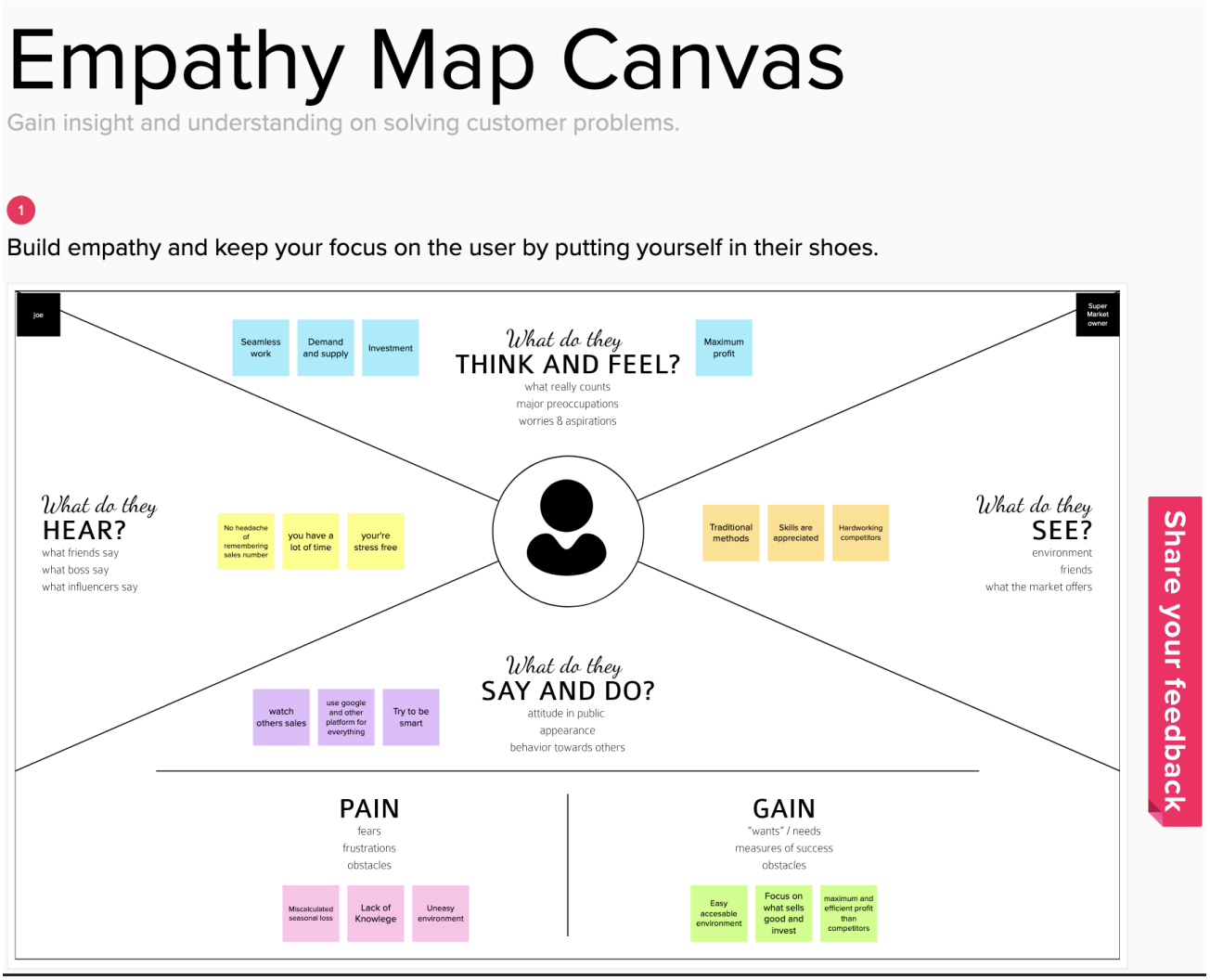
In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks 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.

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.1 IDEATION AND BRAINSTORMING

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!

David Nixon Raj D -Team Leader

- Allowing retailers to keep track of stock in demand
- User friendly so that retailers can access it stress free
- Alerting before a product falls out of stock
- Keeping track of expire dates to make desired action like discount
- easy to find seasonal products to manage supply as per demand
- to keep track of products profit and loss
- making it convenient to file tax and billings
- Making all the data safe and accessable with security

AnishAaron S

- An Application that includes all the present date available inventory along with the quantity for both the customer and the retailer.
- To have a track of seasonal selling products and to keep those products in stock during the demand.
- Predicting the Future sales analysis of the products using machine learning algorithms and past data available dataset
- Triggering the alert message when the stock falls down the threshold amount.
- Providing an easy and user friendly Ecommerce site for the customers.
- Centralized transportation system among the shop branches along with the product tracking functionality

Jenish F

- Customer Feedback and rating system including both the product and the retail shop service.
- Sending E-mail notifcation to the customer regarding the new arrivals and available stocks.
- Keeping a Track of the expiry dates of all the stock and announcing the discounts and offer for those products which is going to expire soon.
- Easy and fast billing system with also provides option for the customers either through cash or through net banking.
- Make sure that the store contains all the day to day vital used from day to dawn.
- Provide special discount for the frst purchase and can add key points with further purchase so future special discounts.

Hemanth Kumar

- Can make use of excel sheet for processing the data.
- Advertise the presence of the store in all the nearest geographic locations.
- Plan appropriate strategic business plans with regard to the competitors and bring the plan noticeable among the customers.
- Bring RFID based product tracking system into the existence.
- Keep a record of regular customers and send them regular notice about the arrivals and exclusive offers and discounts for them.
- Keep a proft and loss records of all the stocks.

John Wesley B

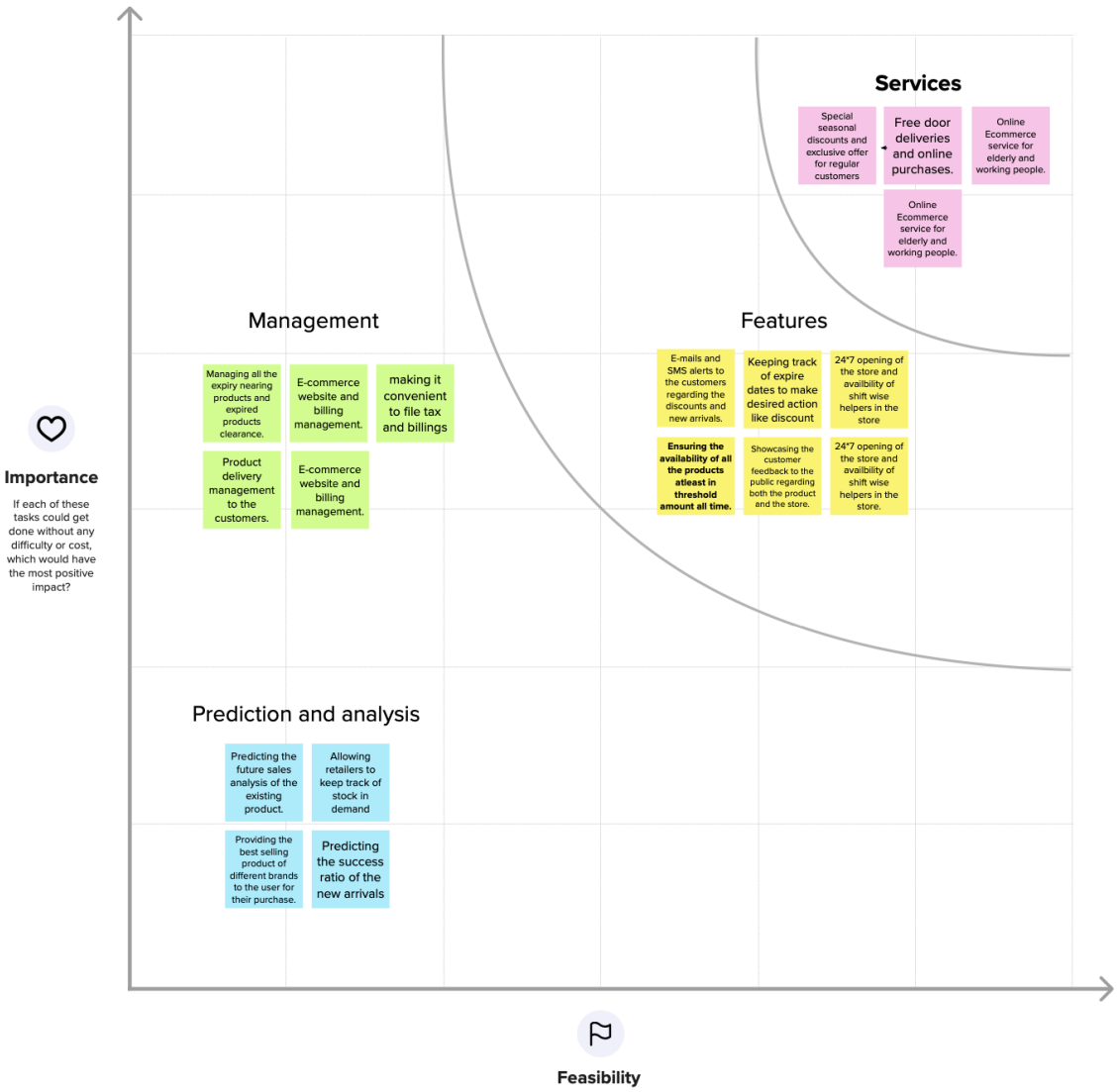
- Deciding whether to invest in a product or not using some predictive analysis of the newly arrived product.
- Scheduling all the product deliveries properly for maximum utilization of transportation.
- Alerting the user regarding the end sale discounts and real time statistics.
- Tax and GST clearance regularly.
- Enhancing customer loyalty and providing transparency in the billing.
- Make sure to have free door deliveries to the nearest areas and to avoid late deliveries.

Step-2: Idea Prioritization

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



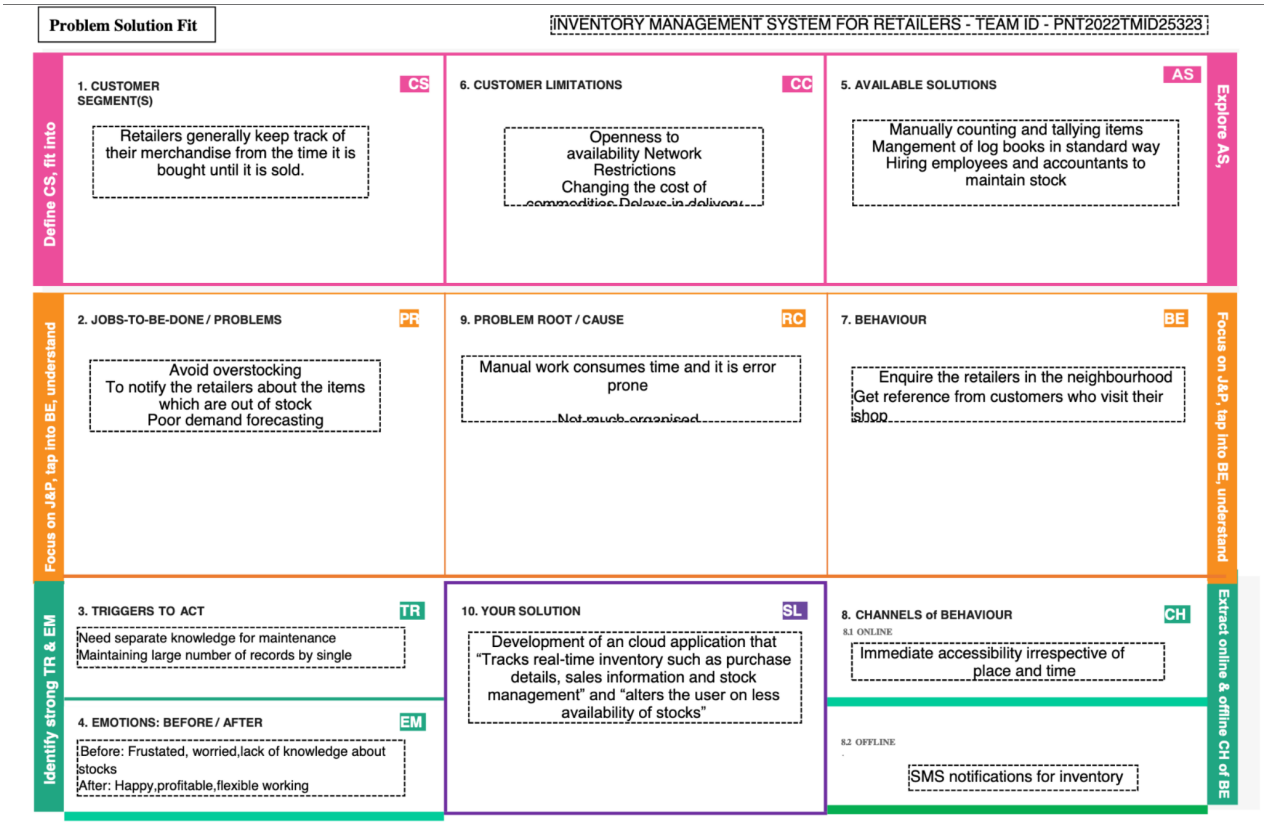
3.1 PROPOSED SOLUTION

S.No.	Parameter	Description
❖	Problem Statement (Problem to be solved)	• The retailers generally facing issues in recording the stocks and its threshold limit available. • The retailers doesn't know which product is getting expired and when it is being expired. • The retailers couldn't track the availability of all the stocks up-to date. • The customers are not satisfied with the retailers store since it doesn't have enough supplements and the deliveries were not made on time.
❖	Idea / Solution description	• This proposed system will have a daily update system whenever a product is sold or it is renewed more. • The system will have an alert triggered to indicate both the expired product and soon going to expire products. • The product availability is tracked daily and an alert system in again kept on to indicate those products which falls below the threshold limit. • All the customers can register their accounts after which they will be given a login credentials which they can use whenever they feel like buying the stocks. • The application allows the customers to know all the present time available stocks and also when the new stock will be available on the store for them to buy. • Tracking the order have become easy with this application for both the retailers and the customers.

❖	**Novelty / Uniqueness**	• **Certain machine learning algorithms are used to predict the seasonal high selling products which can be made available during that time.** • **Prediction of the best selling brand of all certain products based on their popularity, price and customer trust and satisfaction will be implemented.** • **Notifications will be sent to the retailers if any product that the customers have been looking for is not available so that the product can be stocked up soon.** • **Notification will be sent to the customers who buys any certain products regularly when the new arrivals are stocked up.** • **Exclusive discounts and offers are given for regular customers to keep them engaged with the store regularly.**
❖	**Social Impact / Customer Satisfaction**	• **The customers will be highly satisfied since the wasting of time while searching for an unavailable product is reduced.** • **The work load of the retailers will be minimized if the system is automated every day and during every purchase.** • **The customer satisfaction will be improved for getting appropriate response from the retailers and that too immediately.**

❖	**Business Model (Revenue Model)**	• **Hereby we can provide a robust and most reliable inventory management system by using:** **1. ML algorithms for all the prediction purposes using all the past dataset since datasets are undoubtedly available in huge amounts.** **2.Can deploy the most appropriate business advertising models.** **3. To establish a loss preventing strategy.** **4. And to ensure the all time, any where availability of products system.** **5. Usage of freebies business strategy for dragging the customer's attention.**
❖	**Scalability of the Solution**	**• This system can even work more efficiently with large volumes of data.** **• Implementation of anyone and anywhere using the system can be helpful for even a commoner to buy the products.** **• Daily and Each time purchase updates of the stock for preventing inventory shrinkage.** **• Direct chat system with the retailers and the customers for providing best customer service.**

4. PROBLEM FIT



CHAPTER-4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through own application Form Registration through Gmail Registration through LinkedIN Registration through Google Docs.
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login through User name and password. Login through mail ID and password. Login through OTP through mail ID and password. Login through Phone number.
FR-4	Records of the products	Product name Product category Product I'd Stock Count Vendor details
FR-5	Login details	Login Details along with time through Email.

		Login Details along with time through phone number.
FR-6	Updation of inventory Details.	Update through Email Update through User account.
FR-7	Unavailability Alert	Alert Message through mail or phone number.
FR-8	Monitoring of stock	Audit monitoring through incoming and outgoing stock.
FR-9	Database	Usage of a standard database for storing the data.

4.2 Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

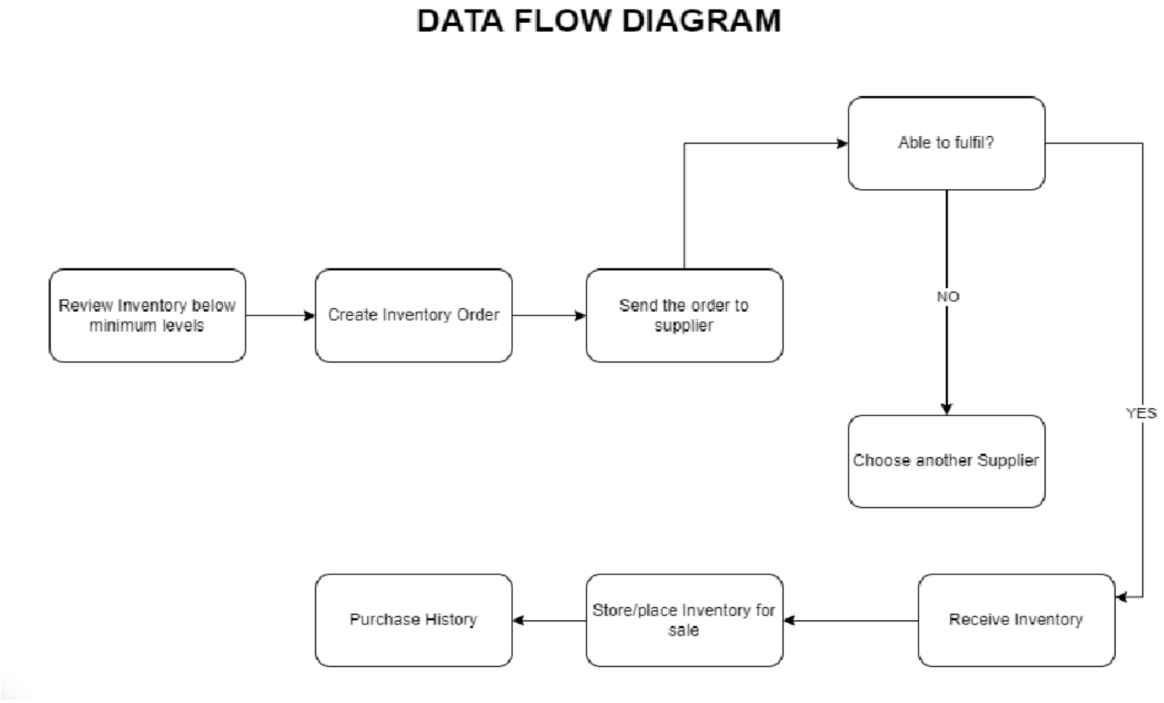
FR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	• Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current 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. • It can use by wide variety of client as it is very simple to learn and not complex to proceed • Easy to use, User-friendly and Responsive.

NFR-2	**Security**	• 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. With Registered Mail id only retailers can log into the application. So it provides authentication. • We are using login for the user and the information will be hashed so that it will be very secure to use.
NFR-3	**Reliability**	• It will be reliable that it can update with every time period so that the accuracy will be good.
NFR-4	**Performance**	• Users 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 saves time. Emails will be sent automatically While stocks are not available.Makes the business process more efficient.Improves organizations performance. • It will be perform fast and secure even at the lower bandwidth
NFR-5	**Availability**	• The availability of products is just one way in which an inventory management system creates customer satisfaction. Inventory management systems are designed to monitor product availability, determine
		purchasing schedules for better customer interaction. • Prediction will be available for every user but only for premium user

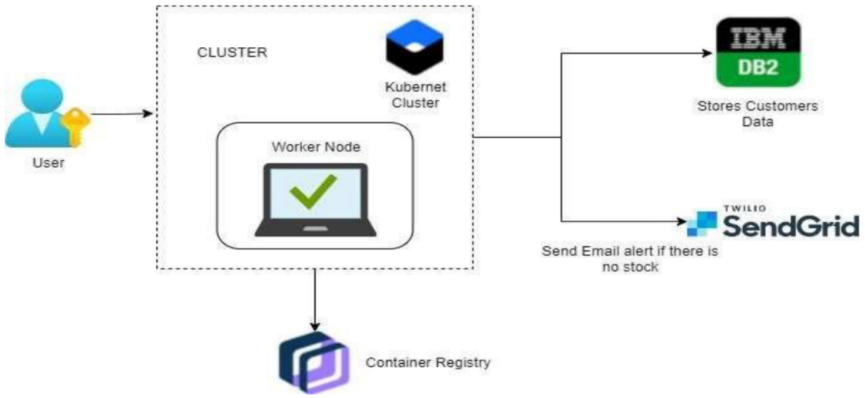
			news,database and price alert will be alert
NFR-6	**Scalability**	•	Scalability is an aspect or rather a functional quality of a system, software or solution.This proposed system for inventory management system can accommodate expansion without restricting the existing workflow and ensure an increase in the output or efficiency of the process
		•	It is scalable that we are going to use data in kilobytes so that the quite amount of storage is satisfied

CHAPTER-5
PROJECT DESIGN

5.1 DATA FLOW DIAGRAM



5.2 SOLUTION AND TECHNOLOGY ARCHITECTURE



5.3 USER STORIES

Use the below template to list all the user stories for the product.

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 email once I have registered for the application	I can receive 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-3
		USN-4	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-2
	Login	USN-5	As a user, I can log into the application by entering email & password	I can log in by entering Gmail & password	High	Sprint-1
	Dashboard	USN-6	As a user, I can track data of sales of products and inventory levels	I can track data of sales of products and inventory levels.	High	Sprint-1

Customer (Web user)	Registration	USN-7	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-8	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-9	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-3
		USN-10	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-2
	Login	USN-11	As a user, I can log into the application by entering email & password	I can log in by entering Gmail & password	High	Sprint-1
	Dashboard	USN-12	As a user, I can track data of sales of products and inventory levels	I can track data of sales of products and inventory levels.	High	Sprint-1
Customer Care Executive	Support	USN-13	As an Executive, I Provide answers for the queries asked by users.	I provide the answers for the queries asked by the users.	High	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release

Administrator	Manage the Stocks	USN-14	As a administrator, I manage the stocks by adding, shipping and storing the stocks in the storage units	I manage the stocks by adding, shipping and storing the stocks in the storage units.	High	Sprint-1
	Control all the users	USN-15	As an administrator, I can control all the users by performing basic CRUD operations.	I can control all the users by performing basic CRUD operations	High	Sprint-1
	Access the database	USN-16	As a administrator, I can control and access the database	I can control and access the database.	High	Sprint-1

CHAPTER-6

6.1 SPRINT PLANNING AND ESTIMATION Project Tracker:

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

Velocity: Sprint – I to 4

Sprint duration = 6 days Velocity of the team = 20 points

$$\text{average velocity (AV)} = \frac{\text{Velocity}}{\text{Sprint duration}}$$

AV = 20/6 = 3.34

Average Velocity = 3.34

6.2 SPRINT DELIVERY SCHEDULE

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	Jenish F
Sprint-1		USN-2	As a user, I can register for the application through E-mail	1	Medium	David Nixon Raj
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	2	Medium	Hemanth Kumar

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High	David Nixon Raj
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	4	High	Jenish F
Sprint-2	Add items to cart	USN-6	As a user, I can add the products I wish to buy to the carts.	5	Medium	AnishAaron
Sprint-3	Stock Update	USN-7	As a user, I can add products which are not available in the dashboard to the stock list.	5	Medium	AnishAaron, Jenish
Sprint-4	Request to Customer Care	USN-8	As a user, I can contact the Customer Care Executive and request any services I want from the customer care.	5	Low	John Wesley
Sprint-4	Contact Administrator	USN-9	I can be able to report any difficulties I experience as a report	5	Medium	Hemanth Kumar

CHAPTER – 7 ADVANTAGES AND DISADVANTAGES

Advantages:

Disadvantages:

CHAPTER-8 CONCLUSION

CHAPTER-9 FUTURE SCOPE

Upgrading the UI that is more user-friendly which will help many users to access the website and also ensures that many retailers can be added into the community.

Using elastic load balancer, it helps to handle multiple requests at the same time which will maintain the uptime of the website with negligible downtime

Also using advanced CSS to make the work flow seamless and most convenient in all devices especially in mobile networks

Also including Machine Learning concepts to analyze and recommend improvement in the overall workchain.

10. 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=b0aebb68-94fa-46ec-a1fc-1
c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=3124
9;SECURITY=SSL;SSLSererCertificate=DigiCertGlobalRootCA.crt;UID=grl1
0864;PWD=sv6usZFTyxRkRLlj","","")

#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-42099-1660649540

Project Demo Link:
https://drive.google.com/file/d/1Y5toi5AZ1Dp5cBZ5h7zyxPCvdtjpKMYl/view?usp=share_link