NAALAIYA THIRAN PROJECT - 2022
19ECI01-PROFESSIONAL READINESS FOR INNOVATION, EMPLOYABILITY AND ENTREPRENEURSHIP

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

A PROJECT REPORT

Submitted by

ANANTHA RAMACHANDRAN	**1905003**
CHANDRAMOULI	**1905008**
LAKSHAN H	**1905025**
SHINY SELVARAJU	**1905048**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE – 641 014

(Government Aided Autonomous Institution affiliated to Anna University)

ANNA UNIVERSITY: CHENNAI 600025

NOVEMBER 2022

Project Report

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. It's essential if retailers like to avoid situations where they run out of popular items or end up with excess items that nobody is buying. Whether it's an online or brick-and-mortar retailer, it can allow to manage the inventory in a way that reduces costs and has a positive impact on your bottom line.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. The other benefits of retail inventory management include increased profits, reduced dead stock, time savings, increased finances and better forecasting. 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.

1.1 PURPOSE

The goal of this project is to create an effective inventory system for retailers that will provide several benefits such as lowering storage costs, not having to tie up money due to dead stock, avoiding many of the tedious, time-consuming tasks that come with managing your inventory, fewer items that take up space rather than meeting your customers' needs and bringing in a profit, lower inventory costs, less dead stock, and enough supply to fill all customer orders, and improving inventory accuracy.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

Inventory is the lifeblood of any product-based business, and it is critical that you manage it effectively in order to achieve business success and growth. Inventory management presents a number of challenges. They include being unable to locate or identify stocks in your inventory, as well as having difficulty shipping products on time, which can harm your company's reputation.

Incomplete, difficult-to-find, or incorrect inventory will inevitably hurt your bottom line. There will be some products or materials that remain unsold or unused, and they may become obsolete, or past their expiry date. These materials or products tend to accumulate over time as they are mostly ignored by inventory managers. Managing inventory waste and defect is another major challenge. To be able to fulfil orders in time, it is essential that you maintain optimal inventory.

Without standard procedures and untrained operators, you could end up with inventory that gets damaged or wasted – and this can not only prove to be very expensive but also lead to dissatisfied customers.

There is a lag in data flow between various assets of the organization. Retailers when ordering their stockpile they order the wrong or goods that are not required for this quarter of the year.

2.1 REFERENCES

i) Inventory Management and it’s effect on customers satisfaction AUTHOR NAME: Scott Grant Eckert

ii) A Study of Inventory Management system of Linamar India Pvt Ltd. AUTHOR NAME: Anajali Mishra & Harshal Anil Salunkhe

iii) Research paper on Inventory Management System AUTHOR NAME: Punam Khobragade, Roshni Selokar, Rina Maraskolhe, Prof.Manjusha Talmale

iv) The Effects of Inventory Management Practices on Operational Performance AUTHOR NAME: Jacklyne Bosibori Otundo and Dr. Walter Okibo Bichanga

v) A Study of Inventory Management System.

AUTHOR NAME: Tariq Sheikh

vi) Combined Inventory Management and routing

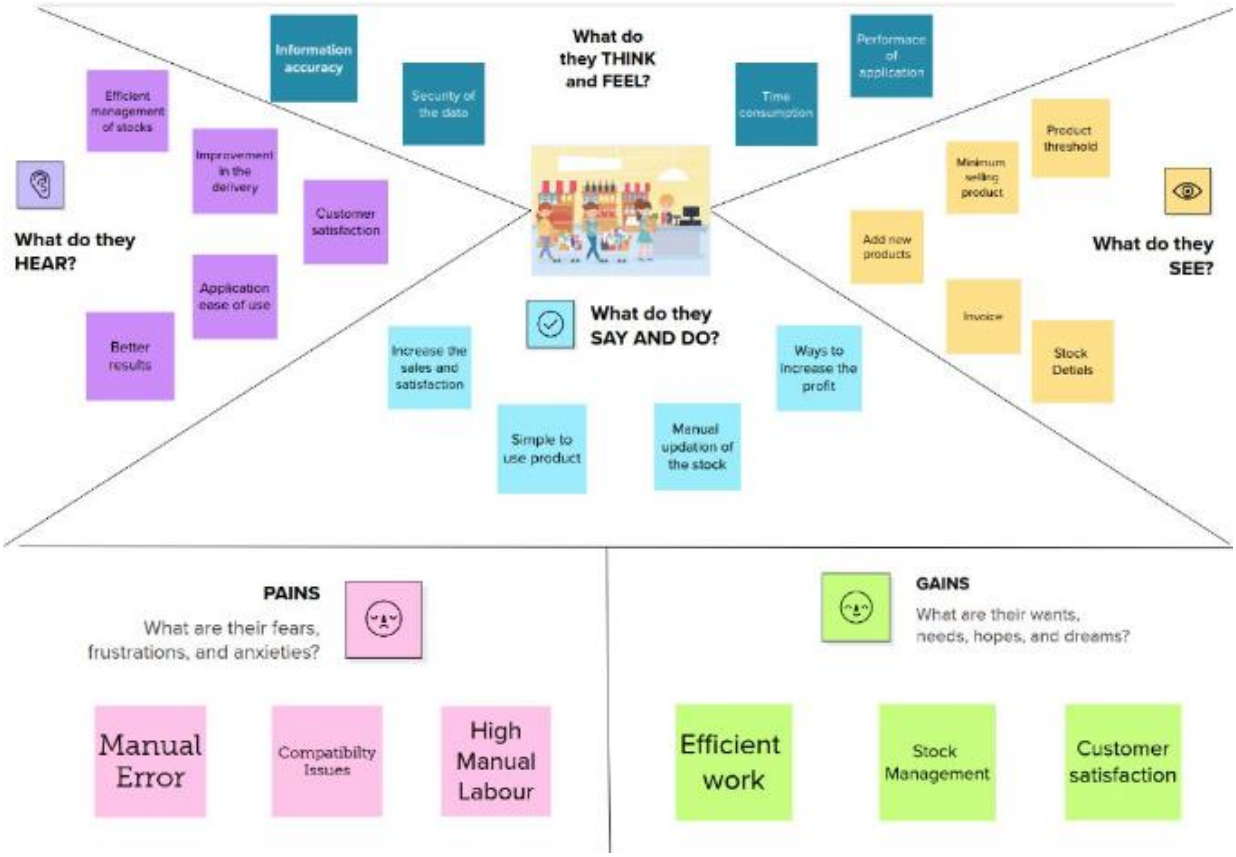
AUTHOR NAME: Henrik Anderson,Arild Hoff,Marielle Christanian, Geir Hasle , Arne Lokkentangen.

2.2 PROBLEM STATEMENT DEFINITION

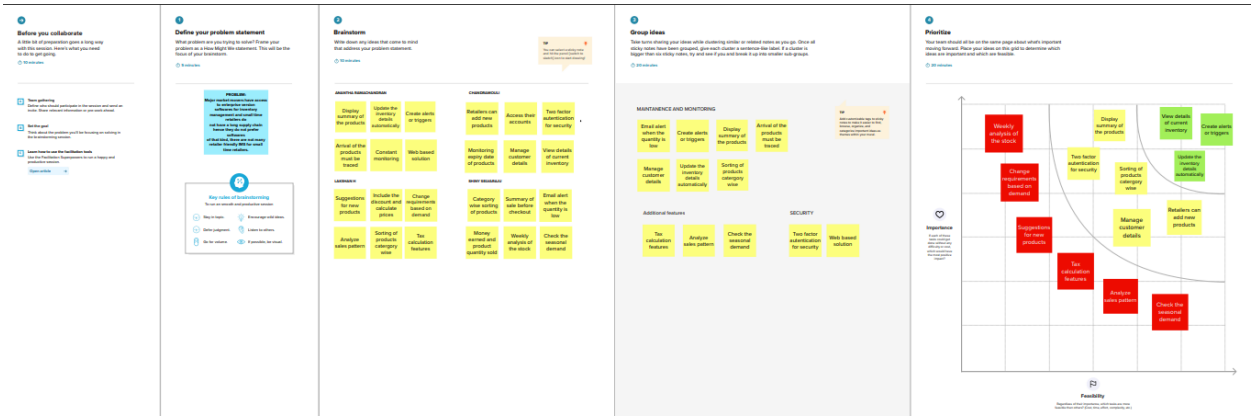
Major market movers have access to enterprise version softwares for inventory management and small time retailers do not have a long supply chain hence they do not prefer softwares of that kind, there are not many retailer friendly IMS for small time retailers.

3. IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



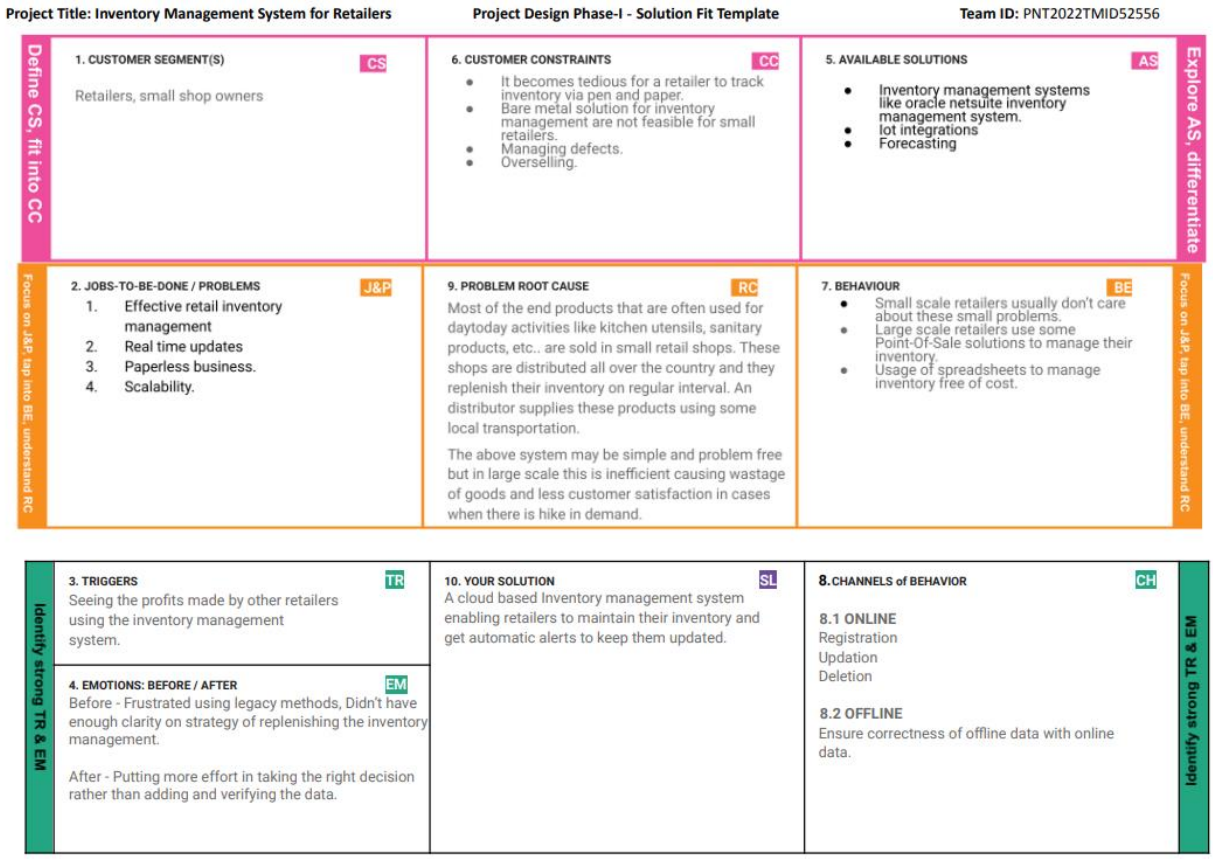
3.1 IDEATION AND BRAINSTORMING



3.2 PROPOSED SOLUTION

A web application is designed to address the above-mentioned issues with the following functionalities: The web application will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. Retailers can add products since it consumes less amount of time than manual entry. 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. An alert is sent automatically by the inventory management system if the stock left count reaches a threshold value and as soon as the alert is received, the stocks required are ordered and as a result pausing of sale is avoided. A simple E-commerce web page is developed to check the stock management. We can order the new products from a particular retailer.

3.3 PROPOSED SOLUTION FIT



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**	The user should be able to register with username, email and password.
FR-2	**Dashboard**	The dashboard should display the overall statistics describing the inventory status.
FR-3	**Add/Remove Item**	User should be able to add/remove products from the inventory.
FR-4	**View inventory**	The user should be abled to view the entire inventory with simple user interface.
FR-5	**Email alert**	An email alert should be triggered in case of shortage of stocks.
FR-6	**Track item**	The user should be abled to fetch the current status of the product (Shipped/Packing/delivered)

4.1 NON-FUNCTIONAL REQUIREMENTS

Non-functional Requirements:

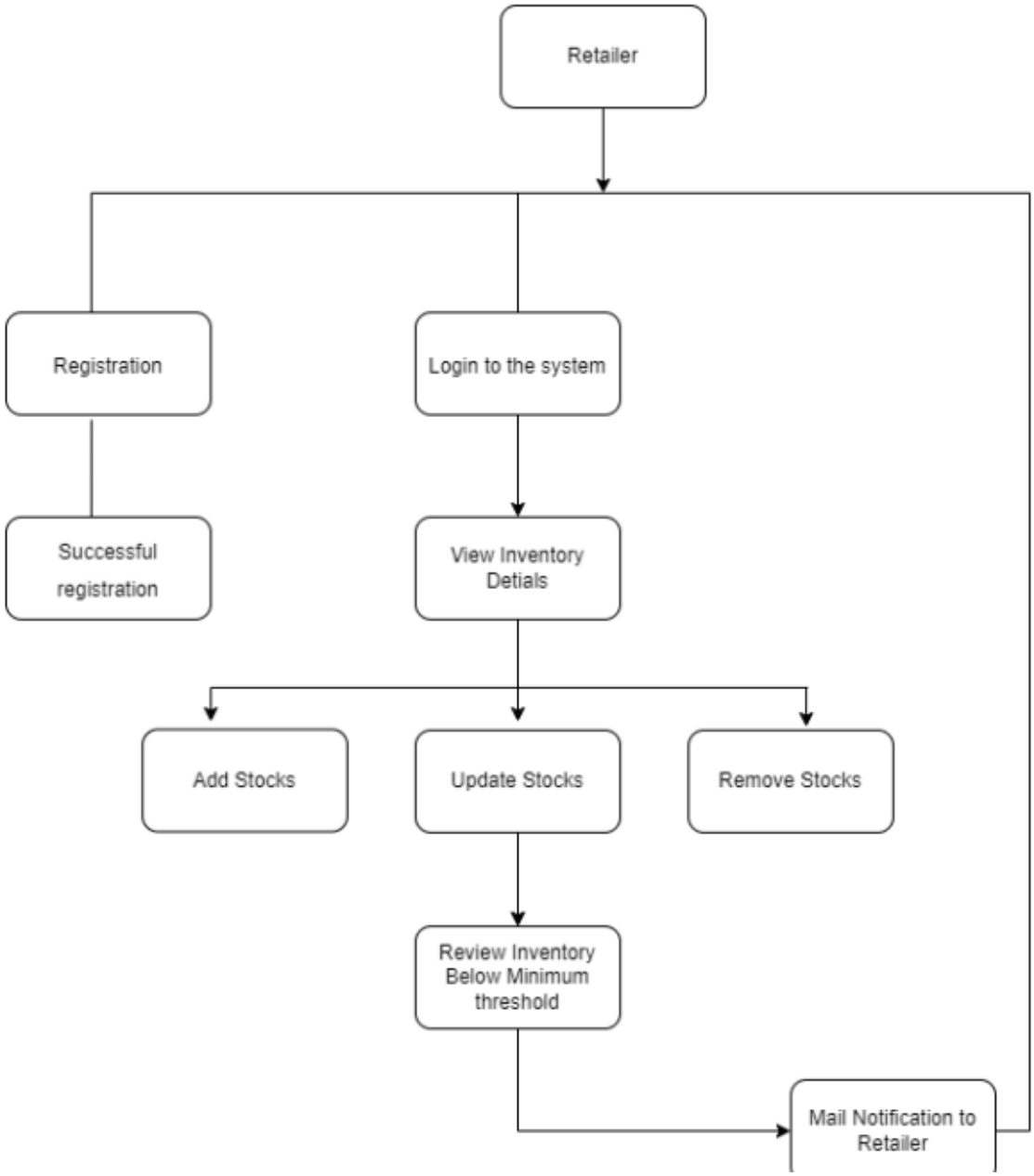
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The system should be easy to use by the retailers such that they do not need to read extensive amount of manuals.
NFR-2	**Security**	The privacy and authenticity of the user should be preserved. It should withold against top vulnerabilities.
NFR-3	**Reliability**	The system must give accurate inventory status to the user continuously. It should successfully add any item and display the same without any error.
NFR-4	**Performance**	The system must not lag and should have no downtime.
NFR-5	**Availability**	The service must be available to user every time the system is turned on.
NFR-6	**Scalability**	The system should support huge amount of users parallely.

5. PROJECT DESIGN

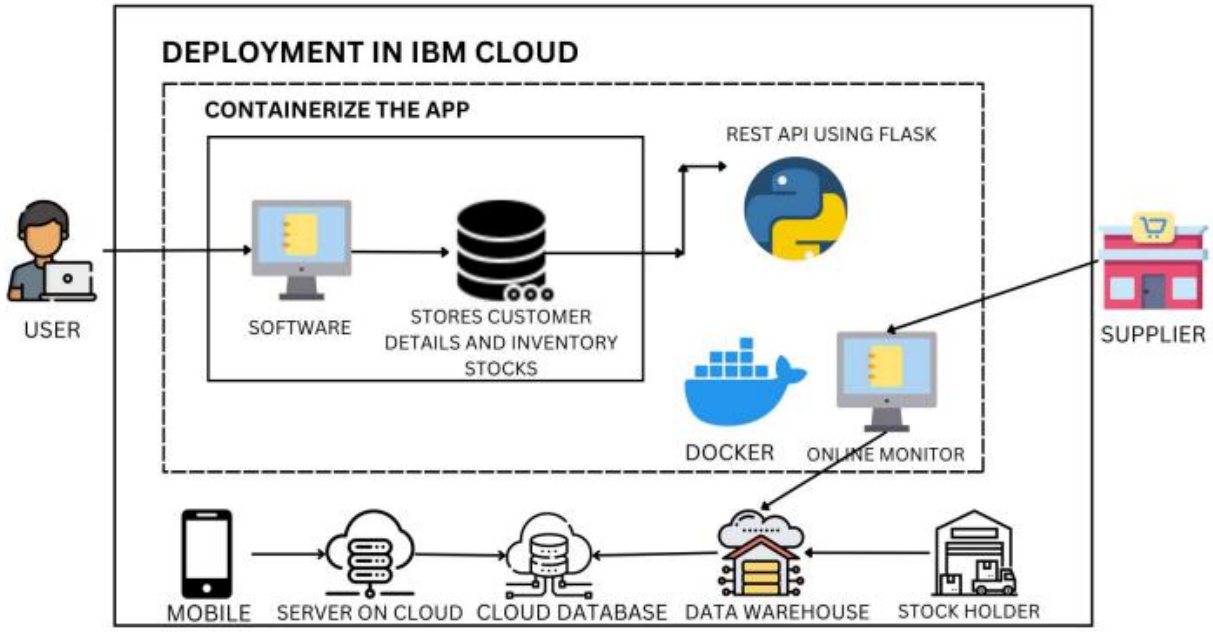
5.1 DATA FLOW DIAGRAMS

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the rightamount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored

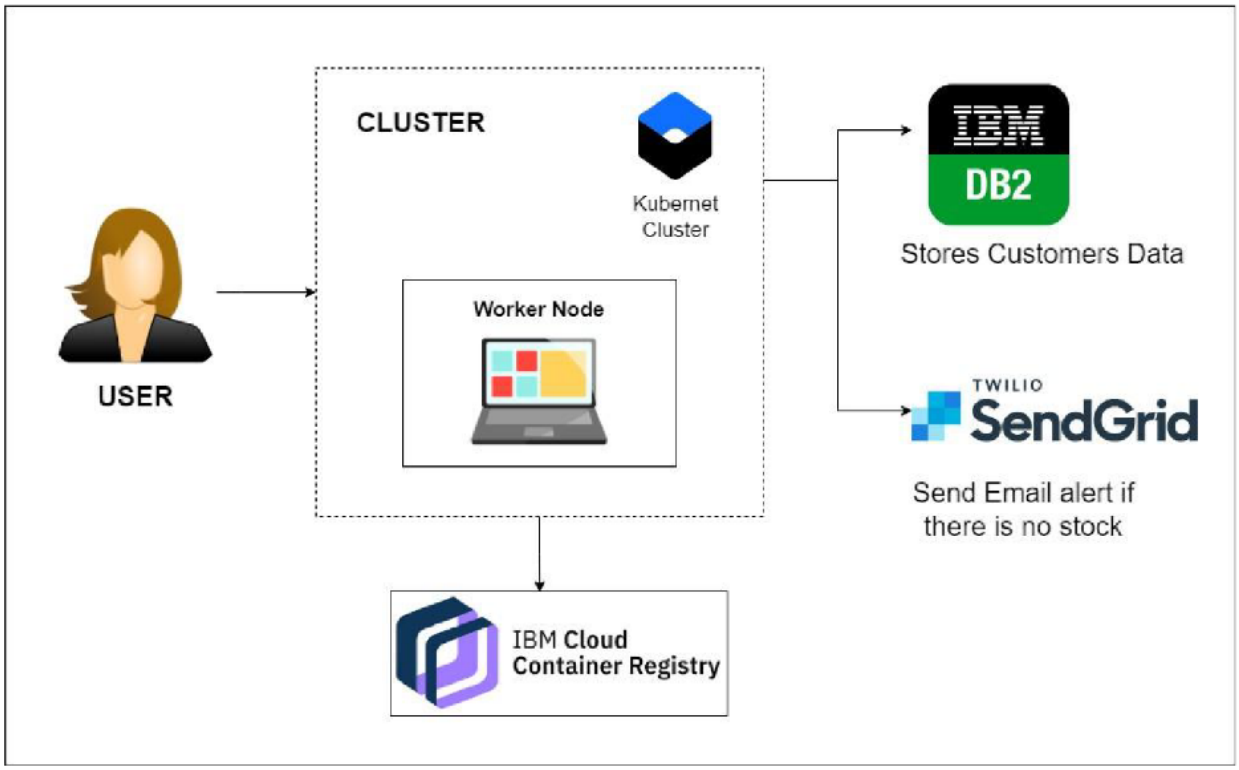


5.1 SOLUTION AND TECHNICAL ARCHITECTURE

5.2.1 SOLUTION ARCHITECTURE



5.2.2 TECHNICAL ARCHITECTURE



5.2 USER STORIES

User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer-Retailers (Web user)	Registration	USN-1	As a retailer, 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 Google Account	I can register & access the dashboard with Google Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register for the application through gmail	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	Validate the userid and 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
	Manage the stocks	USN-7	As a retailer, 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
	Access the Database	USN-8	As a reatailer, I can control and access the database	I can control and access the database.	High	Sprint-1
Customer-Buyer(Web User)	Purchase the products	USN-9	As a buyer aka customer,I can buy any products from particular retailer upto variable quantities.	I can buy any products from particular retailer upto variable quantities.	High	Sprint-1
Customer Care Executive	Support	USN-10	As a Executive,I provide answers for the queries asked by users.	I provide answers for the queries asked by users.	High	Sprint-1

6. PROJECT PLANNING AND SCHEDULING

6.1 Sprint Planning & Estimation

MILESTONE

Setting up the application environment	M-01	Setting up the required resources in the local machine
Integrating send grid service	M-02	To send emails from the application, we need to integrate the SendGrid Service.
Deployment of the app in IBMCloud	M-03	Containerize a Flask application by using Docker and deploy it to the IBM Cloud Kubernetes Service
Ideation Phase	M-04	Collecting information by referring to previous research on a topic and Prepare Literature Survey on the selected Project and Information Gathering, empathy map and ideation
Project Design Phase - I	M-05	Prepare the proposed solution, the problem-solution fit, and the Solution Architecture.
Project Design Phase - II	M-06	Create a customer journey, functional requirements,a data flow diagram, and a technology architecture
Project Planning Phase	M-07	Make a list of milestones, an activity list, and a sprint delivery plan.
Project Development Phase	M-08	Develop and submit Sprint 1,Sprint 2,Sprint 3 and Sprint 4

ACTIVITY LIST

Activity Number	Activity	Sub Activity	Assigned To	Status
1.	Setting up Application Environment	• Create Flask Project • Create IBM Cloud Account • Install IBM Cloud CLI • Docker CLI Installation • Create An Account InSendgrid	All Members	In Progress
2.	Implementing Web Application	• Create UI To Interact With Application	All Members	In Progress
3.	Integrating SendGrid Service	• SendGrid Integration With Python Code	All Members	In Progress
4.	Deployment of App In IBM Cloud	• Containerize The App • Upload Image To IBM Container Registry • Deploy in Kubernetes	All Members	In Progress
5.	Ideation Phase	• Literature Survey On The Selected Project &Information Gathering • Prepare Empathy Map • Ideation	All Members	Completed
6.	Project Design Phase – I	• Proposed Solution • Problem Solution Fit • Solution Architecture	All Members	Completed
7.	Project Planning Phase	• Prepare Milestone & Activity List • Sprint Delivery Plan	All Members	Completed
8.	Project Development Phase	• Delivery Of Sprint-I • Delivery Of Sprint-2 • Delivery Of Sprint-3 • Delivery Of Sprint-4	All Members	Completed

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint 1	20	6 Days	24 Oct 2022	29 Oct 2022	20	31 Oct 2022
Sprint 2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint 3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint 4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

Velocity

Sprint Duration - 6 Days

Velocity of the Team - 20 (points per sprint)

Team's Average Velocity AV = story points / velocity sprint duration = 206 = 3.3

6.3 Reports from JIRA

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	sStory 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.	5	High	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju
Sprint 1	Login	USN-2	As a user, I can log into the application by entering email & password	4	High	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju
Sprint 2	Dashboard	USN-3	As a user, I can see the stock in hand and how much stock will be received and check other details.	4	High	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju
Sprint 2	Product Details	USN-4	As a user, I can view the different products available and the details	3	Low	Anantha Ramachandran Chandramouli G Lakshan H

Sprint 2	Purchase order managem ent	USN-5	As a user, I can enter the newly purchased stock and add or remove the stocks. And upload the purchased details as well.	5	Medium	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju
Sprint 3	Notification	USN-6	As a user, it is good if I get a notification for low stock.	2	Medium	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju
Sprint 3	Profile	USN-7	As a user, I can see my profile and give my details after registering as well.	1	Low	Anantha Ramachandran Chandramouli G Lakshan H Shiny Selvaraju

7. CODING & SOLUTION

i)Sending Email using Sendgrid

CODE:

```
def send_mail(email):
message = Mail(
from_email='1905008cse@cit.edu.in',
to_emails=email,
subject='Account created sucessfully',
html_content='<strong>Your account has been created. Login to manage your
inventory.</strong>')
try:
sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
# sg = SendGridAPIClient(api_key)
response = sg.send(message)
print(response.status_code)
print(response.body)
print(response.headers)
except Exception as e:
print(e.message)
```

When the quantity of the given stocks goes below the specified threshold the function send_mail is invoked and a mail is sent to the retailer that the product is below the specified stock and has to be updated.

If the function is invoked, from and to email address if specified, along with the subject and the content of the mail. Try and catch block for handling exception. Try block has the response code and the header which is printed and catch block to print the error message.

ii) Register Page

```
@app.route("/register", methods=['GET','POST'])
def register():
if request.method == "POST":
email = request.form["email"]
username = request.form["username"]
password = request.form["password"]
firstname = request.form["firstname"]
lastname = request.form["lastname"]

sql = "SELECT * FROM USERS WHERE USER_NAME=?"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)
if account:
return render_template("register.html", msg = "Username already taken! try another")
else:
sql = "INSERT INTO USERS (user_name,first_name,last_name,email,password)
VALUES(?,?,?,?,?)"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,username)
ibm_db.bind_param(stmt,2,firstname)
ibm_db.bind_param(stmt,3,lastname)
ibm_db.bind_param(stmt,4,email)
ibm_db.bind_param(stmt,5,password)
ibm_db.execute(stmt)
send_mail(email);
return redirect(url_for("login"))
elif request.method == "GET":
return render_template("register.html")
```

For the user to register it is in the route /register, where the POST method would get the username, password, firstname and lastname from the user. If the entered username is already

present it will display the message “Username is already taken” or it will insert these values into the database.

iii) Login Page

```
@app.route("/login", methods=['GET','POST'])
def login():
global userName;
global userID;
if request.method == "POST":
username = request.form["username"]
password = request.form["password"]
sql = "SELECT * FROM USERS WHERE USER_NAME=? AND PASSWORD=?"
stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,username)
ibm_db.bind_param(stmt,2,password)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)

if account:
session['user']=username;
return redirect(url_for("dashboard", uid=username))
else:
return render_template("login.html", msg="Invalid username or password!")
elif request.method == "GET":
return render_template("login.html")
```

After the process of registration, if the user enters the username and password. It matches the username and password already stored in the database. If it matches it redirects to the dashboard else will display the message “Invalid username and password”.

iv) Dashboard

```
@app.route("/dashboard", methods=['GET','POST'])
def dashboard():
if request.method == "GET":
sql="select * from products where uid in (select uid from users where user_name = ?);"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,session['user'])
ibm_db.execute(stmt)
```

```
products = []
entry = ibm_db.fetch_assoc(stmt)
while entry != False:
products.append(entry)
entry = ibm_db.fetch_assoc(stmt)
return render_template("view_inventory.html",username=session['user'],products=products);
```

It fetched all the details that are stored in database and displays the available items in the form of a table.

v) Sell Item

```
@app.route("/sellItem",methods=['POST'])
def sell():
pname = request.form["p_name"]
stock = request.form["stock"]
if(int(stock)==0):
return redirect(url_for("dashboard"))
stock = int(stock) - 1;

sql = "UPDATE PRODUCTS SET STOCK_COUNT=? WHERE PRODUCT_NAME=?"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,stock)
ibm_db.bind_param(stmt,2,pname)
ibm_db.execute(stmt)
return redirect(url_for("dashboard"))
```

In order to sell the item, enter the name of the product. The stock value is reduced by 1 and is updated in the database and returns to the dashboard.

vi)Edit product

```
@app.route("/editproduct",methods=["POST"])
def edit_product():
pname = request.form["p_name"]
stock = request.form["stock"]
price = request.form["price"]
m_stock = request.form["mstock"]
```

```
sql = "UPDATE PRODUCTS SET STOCK_COUNT=?, PRICE_PER_UNIT= ?,
MINIMUM_STOCK=? WHERE PRODUCT_NAME=?"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,stock)
ibm_db.bind_param(stmt,2,price)
ibm_db.bind_param(stmt,3,m_stock)
ibm_db.bind_param(stmt,4,pname)
ibm_db.execute(stmt)
return redirect(url_for("epd"))
```

In order to edit the product, the name of the product, stock and price is entered and they are updated in the database.

vii)Add product

```
@app.route("/addproduct",methods=['POST'])
def add_product():
pname = request.form["p_name"]
stock = request.form["stock"]
price = request.form["price"]
m_stock = request.form["minimum_stock"]

sql = "SELECT UID FROM USERS WHERE USER_NAME = ?"
stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,session['user'])
ibm_db.execute(stmt)
uid = ibm_db.fetch_assoc(stmt)
print(uid)
uid = int(uid['UID'])
print(uid)

sql = "INSERT INTO PRODUCTS
(UID,PRODUCT_NAME,STOCK_COUNT,PRICE_PER_UNIT,MINIMUM_STOCK)
VALUES (?,?,?,?,?)"

stmt = ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,uid)
ibm_db.bind_param(stmt,2,pname)
ibm_db.bind_param(stmt,3,stock)
ibm_db.bind_param(stmt,4,price)
```

```
ibm_db.bind_param(stmt,5,m_stock)
ibm_db.execute(stmt)
return redirect(url_for("dashboard"))
```

Similarly, in order to add the product, the name of the product, stock and price is entered and they are added in the database.

8. TESTING

8.1 TEST CASES

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	6	9	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	0	0	0
Skipped	3	0	1	1	5
Won't Fix	0	5	2	1	8
Totals	20	19	13	26	78

8.2 USER ACCEPTANCE TESTING

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	3	0	4
Client Application	51	15	3	33
Security	2	0	0	2

9. RESULT

9.1 PERFORMANCE METRICS

S.No	Project Name	Scope/feature	Functional Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Voluem Changes	Risk Score	Justification
1	Inventory Management	Old but can be added with more features	Low	No Changes	Moderate		>5 to 10%	ORANGE	No significant changes in the process but can be added with rich features to support the project.

10. ADVANTAGES AND DISADVANTGES

Advantages

An inventory system provides an organization with a comprehensive account of available items and the monetary value of these inventory items. The retail price is the price the consumer pays when purchasing an inventory item. Therefore, the retail inventory method values the inventory according to this retail value. The retail inventory method evaluates the ending inventory or cost of goods sold for the organization. This inventory system also requires the organization to consider different factors such as discounts, sales, markups, markdowns and loss of inventory.

An advantage of the retail inventory method is that it does not require a physical inventory. The retail inventory method only requires an organization to record the retail prices of inventory items. If an organization has multiple locations in different cities and states, performing a physical inventory can become a costly and time-consuming undertaking. By using retail inventory, an organization can prepare an inventory for a centralized location. The retail inventory method also allows the organization to create an inventory value report for budgeting or the preparation of financial statements.

Disadvantage

The retail inventory method is only accurate if all pricing across the board is the same and all pricing changes occur at the same rate. In most cases, this is not realistic in retail because of the many variations that exist in merchandise pricing. For example, depreciation, markdowns, product damage and theft can affect the price of the retail inventory. For this reason, any calculations made using the retail inventory method should serve only as an estimate.

11. CONCLUSION

Inventory management systems have the potential to open up new opportunities for retailers by allowing them to manage their inventory more efficiently. It provides the desired level of customer service while allowing for cost-effective operations and minimising inventory investment.

The management system allows retailers to receive email alerts when the stock of a specific product falls below a certain threshold level, allowing him to easily keep track of the products and reorder when necessary, saving storage costs, reducing waste, and meeting customer needs on time. The technical aspects, strengths, and weaknesses of the inventory have been thoroughly discussed, allowing future researchers to gain a thorough understanding of inventory management systems.

However, because inaccurate management can have a negative impact on a customer, the proposed prototypes should be tested in real-time market applications to understand their feasibility and accuracy in the retail market. Furthermore, future research should focus on using inventory data to analyse sales patterns in order to increase profit and reduce waste.

This study will be extremely useful for retailers who currently use manual inventory management and want to transition to digital inventory management for more efficient and effective management.

12. FUTURE SCOPE

This application can be further developed to analyse sales patterns and check seasonal demand for products. Two factor authentication can be used to improve the security of the application. Other features, such as product suggestion and weekly stock analysis, are available.

13. APPENDIX

Source Code:

```
from flask import Flask,render_template, request,redirect,url_for,session

import ibm_db

import re

from flask_session import Session



import os

from sendgrid import SendGridAPIClient

from sendgrid.helpers.mail import Mail



def send_mail(email):

    message = Mail(

        from_email='1905008cse@cit.edu.in',

        to_emails=email,

        subject='Account created sucessfully',

        html_content='<strong>Your account has been created. Login to manage your inventory.</strong>')

    try:

        sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))

        # sg = SendGridAPIClient(api_key)
```

```
        response = sg.send(message)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e.message)



app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
#initializing session
Session(app)


app.secret_key = 'a'


conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=125f9f61-9715-46f9-9399-
c8177b21803b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30426;Security=SSL
;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=xmg49931;PWD=YKecpnUZRCObK
Le2;","","")


print(conn)


@app.route("/")
def home():
```

```
    return redirect(url_for("login"))



@app.route("/login", methods=['GET','POST'])
def login():
    global userName;
    global userID;
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        sql = "SELECT * FROM USERS WHERE USER_NAME=? AND PASSWORD=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)


        if account:
            session['user']=username;
            return redirect(url_for("dashboard", uid=username))
        else:
            return render_template("login.html", msg="Invalid username or password!")
    elif request.method == "GET":
```

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


@app.route("/register", methods=['GET','POST'])
def register():
    if request.method == "POST":
        email = request.form["email"]
        username = request.form["username"]
        password = request.form["password"]
        firstname = request.form["firstname"]
        lastname = request.form["lastname"]


        sql = "SELECT * FROM USERS WHERE USER_NAME=?"
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            return render_template("register.html", msg = "Username already taken! try another")
        else:
            sql = "INSERT INTO USERS (user_name,first_name,last_name,email,password)
VALUES(?,?,?,?,?)"
            stmt = ibm_db.prepare(conn,sql)
            ibm_db.bind_param(stmt,1,username)
            ibm_db.bind_param(stmt,2,firstname)
```

```
            ibm_db.bind_param(stmt,3,lastname)
            ibm_db.bind_param(stmt,4,email)
            ibm_db.bind_param(stmt,5,password)
            ibm_db.execute(stmt)
            send_mail(email);
            return redirect(url_for("login"))
    elif request.method == "GET":
        return render_template("register.html")


@app.route("/dashboard", methods=['GET','POST'])
def dashboard():
  if request.method == "GET":
    sql="select * from products where uid in (select uid from users where user_name = ?);"
    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['user'])
    ibm_db.execute(stmt)


    products = []
    entry = ibm_db.fetch_assoc(stmt)
    while entry != False:
        products.append(entry)
        entry = ibm_db.fetch_assoc(stmt)
    return render_template("view_inventory.html",username=session['user'],products=products);
```

```
@app.route("/sellItem",methods=['POST'])
def sell():
    pname = request.form["p_name"]
    stock = request.form["stock"]
    if(int(stock)==0):
        return redirect(url_for("dashboard"))
    stock = int(stock) - 1;

    sql = "UPDATE PRODUCTS SET STOCK_COUNT=? WHERE PRODUCT_NAME=?"
    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,stock)
    ibm_db.bind_param(stmt,2,pname)
    ibm_db.execute(stmt)
    return redirect(url_for("dashboard"))

@app.route("/epd")
def epd():
    sql="select * from products where uid in (select uid from users where user_name = ?);"
    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['user'])
    ibm_db.execute(stmt)
    products = []
```

```
    entry = ibm_db.fetch_assoc(stmt)
    while entry != False:
        products.append(entry)
        entry = ibm_db.fetch_assoc(stmt)
    return render_template("edit_product.html",username=session['user'],products=products)


@app.route("/editproduct",methods=["POST"])
def edit_product():
    pname = request.form["p_name"]
    stock = request.form["stock"]
    price = request.form["price"]
    m_stock = request.form["mstock"]
    sql = "UPDATE PRODUCTS SET STOCK_COUNT=?, PRICE_PER_UNIT= ?,
MINIMUM_STOCK=? WHERE PRODUCT_NAME=?"
    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,stock)
    ibm_db.bind_param(stmt,2,price)
    ibm_db.bind_param(stmt,3,m_stock)
    ibm_db.bind_param(stmt,4,pname)
    ibm_db.execute(stmt)
    return redirect(url_for("epd"))


@app.route("/adp")
def adp():
```

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


@app.route("/addproduct",methods=['POST'])
def add_product():
    pname = request.form["p_name"]
    stock = request.form["stock"]
    price = request.form["price"]
    m_stock = request.form["minimum_stock"]

    sql = "SELECT UID FROM USERS WHERE USER_NAME = ?"
    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['user'])
    ibm_db.execute(stmt)
    uid = ibm_db.fetch_assoc(stmt)
    print(uid)
    uid = int(uid['UID'])
    print(uid)

    sql = "INSERT INTO PRODUCTS
(UID,PRODUCT_NAME,STOCK_COUNT,PRICE_PER_UNIT,MINIMUM_STOCK)
VALUES (?,?,?,?,?)"

    stmt = ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,uid)
```

```
    ibm_db.bind_param(stmt,2,pname)
    ibm_db.bind_param(stmt,3,stock)
    ibm_db.bind_param(stmt,4,price)
    ibm_db.bind_param(stmt,5,m_stock)
    ibm_db.execute(stmt)
    return redirect(url_for("dashboard"))

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True);
```

LOGIN PAGE:

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

{% block head %}
<title>Register page</title>
{% endblock %}

{% block body%}
<div class="center">
  <p>{{msg}}</p>
  <h1>Register</h1>
  <form class="form" action="/register" method="POST">
```

```
<div class="txt_field">
  <input type="text" required name = "firstname">
  <span></span>
  <label>First_Name</label>
</div>

<div class="txt_field">
  <input type="text" required name = "lastname">
  <span></span>
  <label>Last_Name</label>
</div>

<div class="txt_field">
  <input type="text" required name = "email">
  <span></span>
  <label>E-mail</label>
</div>

<div class="txt_field">
  <input type="text" required name = "username">
  <span></span>
  <label>Username</label>
```

```
    </div>


    <div class="txt_field">
      <input type="password" required name = "password">
      <span></span>
      <label>Password</label>
    </div>


    <input type="submit" value="Register">
    <div class="signup_link">
    </div>
  </form>
</div>
{% endblock %}
```

GIT HUB LINK: https://github.com/IBM-EPBL/IBM-Project-39984-1660574346.git

PROJECT DEMO LINK:
https://drive.google.com/file/d/1zXsHE2MlOeu_ftXloPss7KtJ96BFrEtP/view?usp=drivesdk