PROJECT REPORT

in the Title of

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

TEAM ID : PNT2022TMID49364

TEAM LEAD : Gopinath R

TEAM MEMBER 1: Manikandan C

TEAM MEMBER 2: Nagasaravanan E

TEAM MEMBER 3:Pradeeshwaran R

TABLE OF CONTENT:

1. INTRODUCTION

PROJECTOVERVIEW:

The project Inventory Management System is a complete desktop based application designed on .Net technology using Visual Studio Software This desktop application is based on the management of stock of an organization. The application contains general organization profile, sales details, Purchase details and the remaining stock that are presented in the organization. There is a provision of updating the inventory also. This application also provides the remaining balance of the stock as well as the details of the balance of transaction. Each new stock is created and entitled with the named and the entry date of that stock and it can also be update any time when required as per the transaction or the sales is returned in case. Here the login page is created in order to protect the management of the stock of organization in order to prevent it from the threads and misuse of the inventory

PURPOSE:

The primary purpose of inventory management is to ensure there is enough goods or materials to meet demand without creating overstock, or excess inventory

2.LITERATURE SURVEY

EXISTING PROBLEM

When there is no proper system to track products, materials, or equipment in the store, it can be cumbersome and time-consuming to find them when you have sales orders. This will help your employees identify the products that are needed.

PROBLEM SOLVING DEFINITION

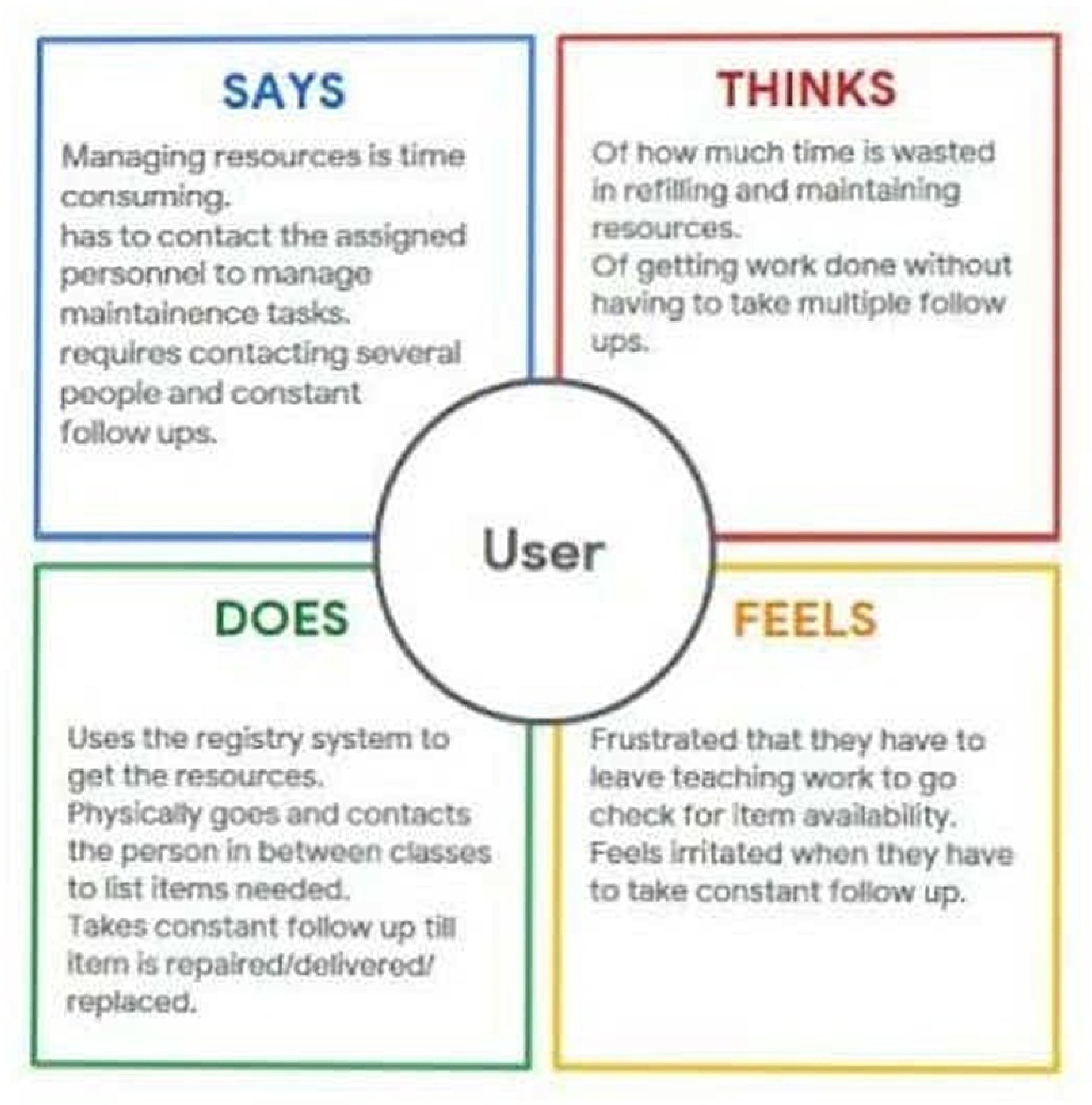
Inventories are necessary for sales, which generate profits and poor management of inventories results in excess inventory, resulting in a lower return on capital invested, affecting the cash conversion cycle The approximate cost to hold inventory is very high, so maintaining excessive levels of inventories can ruin the company, as they have to reduce prices and absorb losses, and if missing could reduce sales, now maintain inventory levels according to sales forecasts.

REFERENCES:

1. www.ideaprojects.com
2. https://www.riotinsight.com/article-inventory-management-system

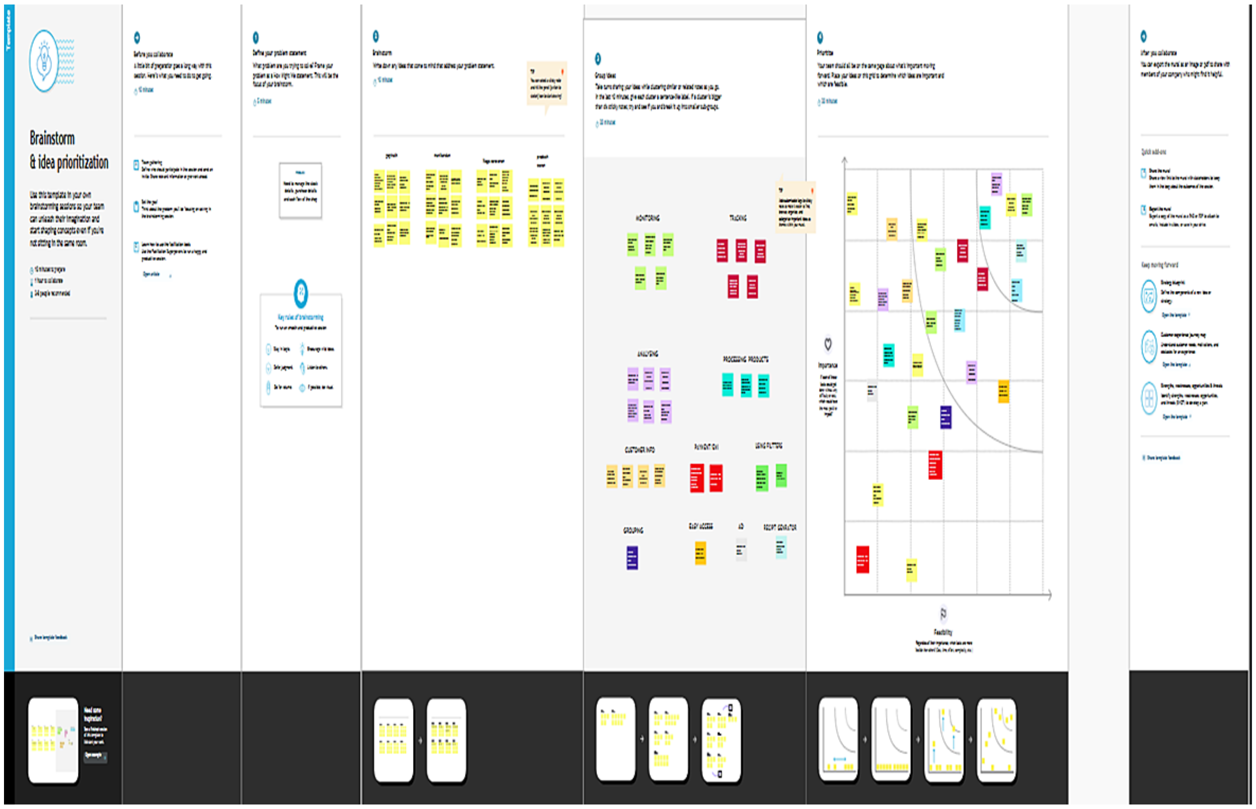
3 . IDEATION AND PROPOSEDSOLUTION

EMPATHY MAP :


SAYS
Managing resources is time consuming.
has to contact the assigned personnel to manage maintainence tasks.
requires contacting several people and constant follow ups.
THINKS
Of how much time is wasted in refilling and maintaining resources.
Of getting work done without having to take multiple follow ups.
User
DOES
Uses the registry system to get the resources.
Physically goes and contacts the person in between classes to list items needed.
Takes constant follow up till item is repaired/delivered/replaced.
FEELS
Frustrated that they have to leave teaching work to go check for item availability.
Feels irritated when they have to take constant follow up.

IDEATION AND BRAINSTROM :

Inventory management is a simplified process of sourcing, storing and overseeing a company's inventory. Why use spreadsheets or ledgers to manually enter data when you can use an advanced automated inventory tracking system.



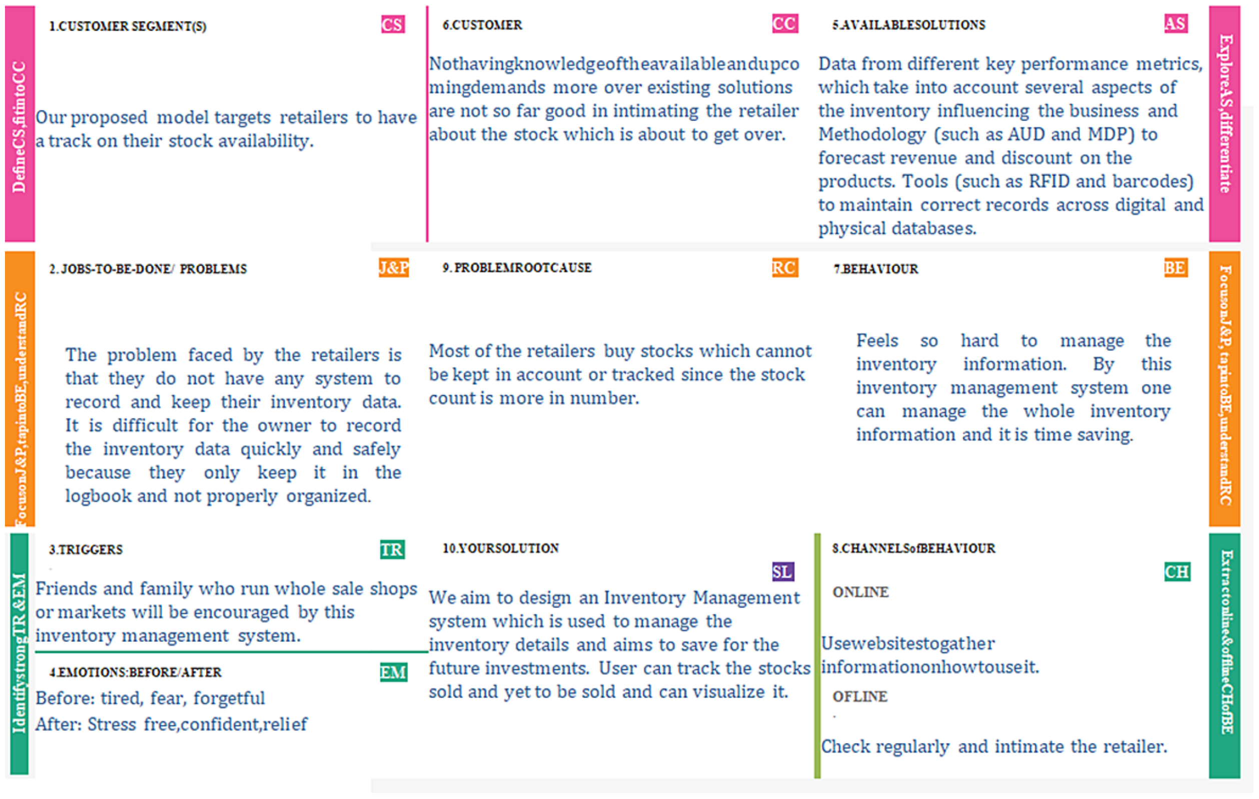
PROPOSED SOLUTION :

S.No.	Parameter	Description
1.	**ProblemStatement(Problemtobesolved)**	The problem faced by the retailers is that they donot have any system to record and keep theirinventory data. It is difficult for the owner to recordthe inventory data quickly and safely because theyonly keep it in the logbook and not properlyorganized.

2.	**Idea/Solutiondescription**	We aim to design an Inventory Management systemwhich is used to manage the inventory details andaims to save for the future investments. User cantrack the stocks sold and yet to be sold and canvisualizeit. The Applicationwill notify the userwhenastockisabouttocomplete.Ourwebapplication will monitor user’s stock by tracking thereceivedSMS’sfromtheuser’smobile.
3.	**Novelty/Uniqueness**	Retailersgetnotifiedwhenthestockisabouttogetover and intimates theuserto buymorestock. ProvidingKeyPerformanceIndicatorforanalysingstock.Demand basedadvanced stockpre-order.
4.	**SocialImpact/CustomerSatisfaction**	Encouragesuser to t r a c k s t o c kavailablityandincreaseprofit.Ithelpstomakeabetterbudgetthathewillhave afinancialcontrol.
5.	**BusinessModel(RevenueModel)**	Thelowcostrequirementfordesigningthis proposedmodelmakesitmorereliableanduserfriendly.
6.	**ScalabilityoftheSolution**	With efficient usage of IBM cloud, this proposedmodel will be able to handle a large number of userdata.Thismakesahugenumberofuserstoeasily accessandefficientlyuseit.

PROBLEM SOLUTION FIT :

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem



DefineCS,fitintoCC

1.CUSTOMER SEGMENT(S) CS

Our proposed model targets retailers to have a track on their stock availability.

6.CUSTOMER CC

Nothavingknowledgeoftheavailableandupcomingdemands more over existing solutions are not so far good in intimating the retailer about the stock which is about to get over.

5.AVAILABLESOLUTIONS AS

Data from different key performance metrics, which take into account several aspects of the inventory influencing the business and Methodology (such as AUD and MDP) to forecast revenue and discount on the products. Tools (such as RFID and barcodes) to maintain correct records across digital and physical databases.

ExploreAS,differentiate

FocusonJ&P,tapintoBE,understandRC

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

The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.

9.PROBLEMROOTCAUSE RC

Most of the retailers buy stocks which cannot be kept in account or tracked since the stock count is more in number.

7.BEHAVIOUR BE

Feels so hard to manage the inventory information. By this inventory management system one can manage the whole inventory information and it is time saving.

FocusonJ&P, tapintoBE,understandRC

IdentifystrongTR &EM

3.TRIGGERS TR

Friends and family who run whole sale shops or markets will be encouraged by this inventory management system.

4.EMOTIONS:BEFORE/AFTER EM

Before: tired, fear, forgetful

After: Stress free,confident,relief

10.YOURSOLUTION SL

We aim to design an Inventory Management system which is used to manage the inventory details and aims to save for the future investments. User can track the stocks sold and yet to be sold and can visualize it.

8.CHANNELSofBEHAVIOUR CH

ONLINE

Usewebsitestogather informationonhowtouseit.

OFLINE

Check regularly and intimate the retailer.

Extractonline&offlineCHofBE

4 . REQUIREMENT ANALYSIS

FUNCTIONAL REQUIREMENTS :

FR. No.	Functional Requirement (Epic)	Sub Requirement (Story/Sub-Task)
FR-1	User Registration	Registration through registration form. Registration through One-Tap Google Sign-in.
FR-2	User Authentication andConfirmation	Authentication via GoogleAuthentication. Confirmation via Email.Confirmation via

		OTP.
FR-3	Product management	Quicklyproduce reports for singleor multiple products. Track information of dead and fast-movingproducts. Track information of suppliers andmanufacturers of the product.
FR-4	Audit Monitoring	The technique of tracking crucial data isknownas audit tracking. Monitor the financial expenses carried outthroughout the whole time(from receivingorder of the product to delivery of the product).
FR-5	Historical Data	Data of everything shouldbe stored foranalytics and forecasting.

FR – 6	CRM (Customer RelationshipManagement)	Track thecustomer experience via ratings givenby them. Get customer reviews regularly or atleast atthe time of product delivery to work on customer satisfaction. User-friendly GUIto increase the customer basefrom only techies to normal people.
FR - 7	Security Policy	User datacollected must be as secureas possible. User data must not be misused. They can only be used for user preferred advertisingpurposes.

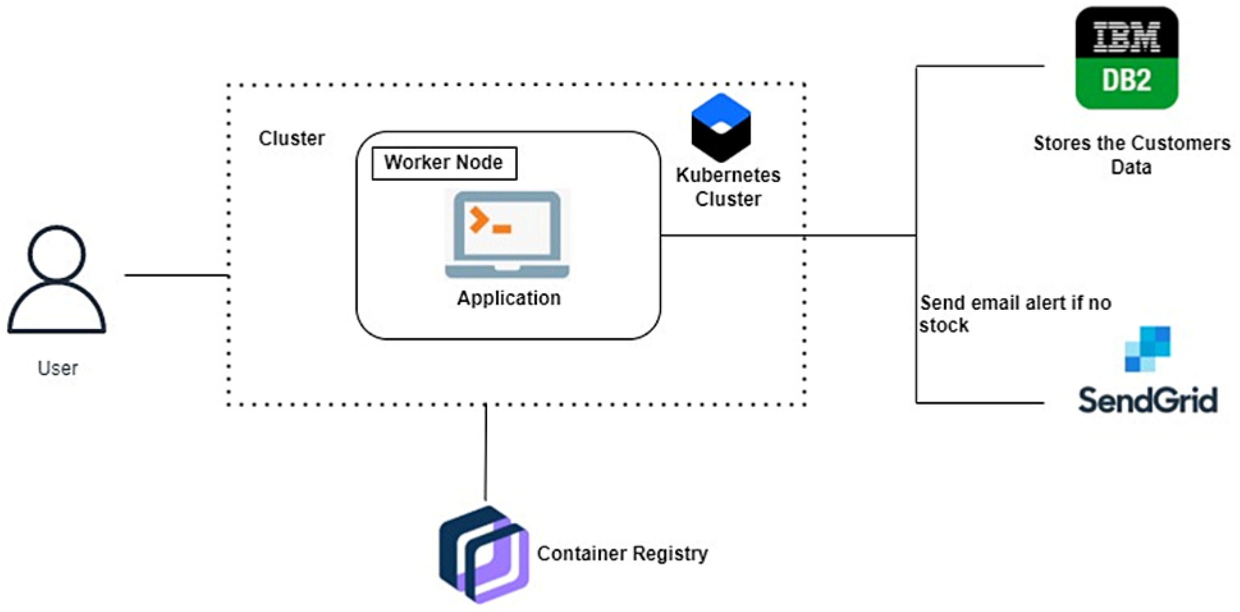
NON – FUNCTIONAL REQUIREMNTS :

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The UI should be accessible to everybody despite of there diversity in languages. People with some impairments should also be able to use theapplication with ease.(Example, integrate google assistant so that blind people can use it). .
NFR-2	Security	The security requirements deal with the primary security. Only authorized userscan access the system withtheir credentials. Administrator or the concerned security team shouldbe alerted on any unauthorized access or data breaches so as to rectifyit immediately.
NFR-3	Reliability	The software should be able to connect to the database in the event of theserver being down due to a hardwareor software failure. The users must me intimated by the periodic maintenance breakof the server so thatthey will be aware of it.
NFR-4	Performance	Performance of the app should be reliable withhigh-end servers on which the software is running.
NFR-5	Availability	The software should be available to the users 24/7with all functionalities working. New moduledeployment should not impact theavailability of existing modules and their functionalities.
NFR-6	Scalability	The wholesoftware deployed must be easilyscalable as the customer base increases.

5. PROJECT DESIGN

DATA FLOW DIAGRAM

The data flow diagram for the proposedproject work



SOLUTION AND TECHNICAL ARCHITECTURE :

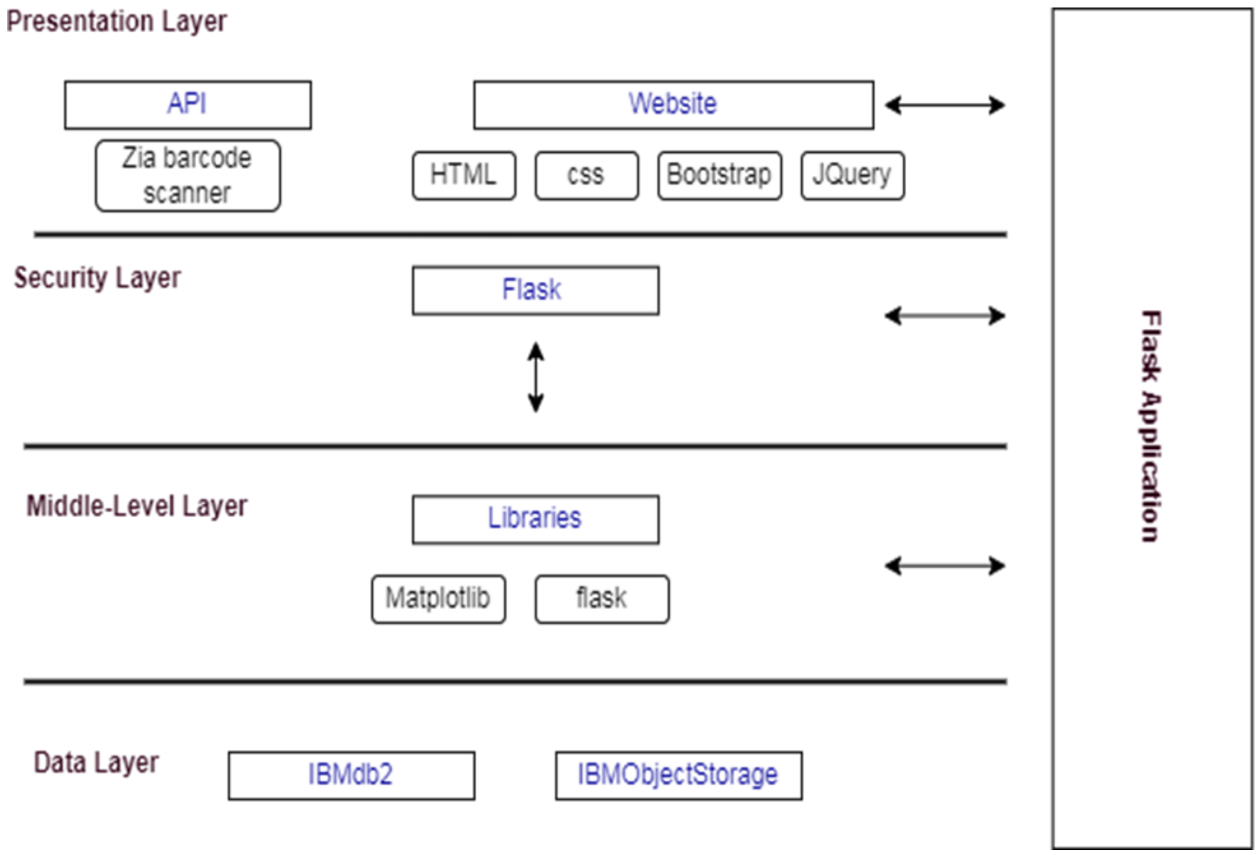


Table-1 : Components & Technologies:

S. No	Component	Description	Technology
1	User Interface	Web UI with Chatbot	HTML, CSS, Bootstrap, Jquery
2	Calculating Products Count	By entering barcodedetails into theapplication	Zia Barcode Scanner
3	Showing high demandproduct	By the products datain IBMdb2	Data Visualization using Python Bar plotby Matplot Library

4.	Alert and Notification	Alerting theretailers regardingthelow stock countof the product	SendGrid
5	Chat	Chat withwatson assistant	IBM Watson Assistant
6	Cloud Database	Database Service on Cloud	IBM DB2
7	File Storage	File storage requirements	IBM Object Storage
8	External API-1 Barcode	To Scanthe product barcode	Zia Barcode Scanner
9	Infrastructure (Server /Cloud)	Cloud ServerConfiguration	Cloud Foundry, Kubernetes

Table-2: Application Characteristics:

S.No	**Characteristics**	**Description**	**Technology**
1.	Open-Source Frameworks	Styling our page,Python flaskmicroframework	Python Flask, Bootstrap
2.	Security Implementations	For securing our cloud data	SSL Certificates
3.	ScalableArchitecture	Three – tierarchitecture (MVC)	Web server - HTML,CSS, Javascript Application server - Python Flask,Docker, Container Registry Database server- IBM DB2
4.	Availability	availabilityof application	IBM Load Balancer
5.	Performance	5 requests per seconds, Use of LocalMachine CacheMemory	IBM Cloud, CDN

6. PROJECT PLANNING AND SCHEDULING

SPRINT PLANNING AND ESTIMATION

In the project planning and scheduling the project backlogs , projectssprintschedule and estimations are tabled .

Sprint	FunctionalRequirement (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	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN
Sprint-1		USN-2	As a user, Ican login through my E-mail.	3	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN
Sprint-1	Confirmation	USN-3	As a user, I can receivemy confirmation emailonce I have registered for the application.	2	High	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN
Sprint-1	Login	USN-4	As a user, I can log in to the authorized accountby entering the registered email and password.	3	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN

Sprint 2:

Sprint	FunctionalRequirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Dashboard	USN-5	As a user, I can viewthe products thatare available currently.	4	High	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN
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	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN

sprint 3 :

Sprint	FunctionalRequirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-3	Sales prediction	USN-7	As a user, I can get access to sales prediction toolwhich can helpme to predict better restock management of product.	6	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN

sprint 4:

Sprint	FunctionalRequirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-4	Request for customercare	USN-8	As a user, I am ableto request customer careto get in touch with the administrators and enquire the doubts andproblems.	4	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN

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	Medium	G.GOPINATH C.MANIKANDAN E.NAGA SARAVANAN R.PRADEESHWARAN

7. CODING AND SOLUTIONING

FEATURES 1 :

When the quantity is gone below 5 ,it sends an alert message to the Manager through mail.

CODE :

```
from flask import Flask, render_template, request, redirect,
url_for , session
import ibm_db
import re
from flask_mail import *
from random import randint
from datetime import datetime



app = Flask(__name__)
app.Secret_key='a'
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-4cd4-
bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=
32304;SECURITY=SSL;SSLServerCertificate=certificate.crt;UID=ngw72
704;PWD=mzQy2ksb3Ff6i3Ex",'','')
mail = Mail(app)
app.secret_key = "abc"
app.config["MAIL_SERVER"]='smtp.gmail.com'
```

```python
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = 'verifyemail0904@gmail.com'
app.config['MAIL_PASSWORD'] = 'fkchuaznhiwjjyuq'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
otp = randint(000000,999999)
date=datetime.now()
@app.route('/')
def home():
    return redirect(url_for('quantity'))
@app.route('/additem')
def additem():
    return render_template('addproduct.html',count=session['count'],name=session['name'])
@app.route('/alter')
def alter():
    return render_template('productid.html',count=session['count'],name=session['name'])
@app.route('/statement')
def statement():
    billingid = randint(000000,999999)
    return redirect(url_for('detail',name=session['name'],bid=billingid))

@app.route('/login',methods= ['GET','POST'])
def login():
    global userid
    msg=''
    if request.method=='POST':
        username=request.form['username']
        session['name']=request.form['username']
        password=request.form['password']
        sql="SELECT * FROM users WHERE username= ? 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)
        print(account)
        if account:
           msg='logged in successfully!'
           return redirect(url_for('display'))
        else:
           msg='incorrect Username/Password !'
           return render_template("login.html",
msg=msg,account="")

@app.route('/team')
def team():
    return render_template('login1.html')
@app.route('/reg')
def reg():
    return render_template('index.html')
@app.route('/forget')
def forget():
    return render_template('vindex.html')
@app.route('/register' , methods=['GET' , 'POST'])
def register():
     msg=''
     if request.method=='POST':
        username=request.form['username']
        email=request.form['email']
        password=request.form['password']
        sql="SELECT * FROM users WHERE username=?"
        stmt=ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account=ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
             msg='account already exits'
             return render_template('index.html',msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+',email):
            msg='invalid email address'
        else:
            insert_sql="INSERT INTO users VALUES(?,?,?)"
            prep_stmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, username)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg='you have successfully registered'
            return render_template("login1.html",msg=msg,name=session['name'])
     elif request.method =='POST':
          msg='please fill out the form'
          return render_template('index.html', msg=msg)

 @app.route('/verify',methods = ["POST"])
 def verify():
     email = request.form["email"]
     session['email']=request.form["email"]
     msg = Message(subject='OTP',sender = 'verifyemail0904@gmail.com', recipients = [email])
     msg.body = str(otp)
     mail.send(msg)
     return render_template('verify.html',email=email)

 @app.route('/validate',methods=["POST"])
 def validate():
     user_otp = request.form['otp']
     email=session['email']
     if otp == int(user_otp):
         return render_template('register.html',email=email)
     return "<h3>failure</h3>"

 @app.route('/display')
 def display():
     val={}
     i=0
     sql="SELECT * FROM products"
     stmt = ibm_db.exec_immediate(conn, sql)
     dictionary = ibm_db.fetch_assoc(stmt)
     if dictionary!=False:
         val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price
 ':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
      while dictionary != False:
       i=i+1
       dictionary = ibm_db.fetch_assoc(stmt)
       if dictionary!=False:
        val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
     print(*val.values())
     return render_template("dashboard.html",account=val,name=session['name'],count=session['count'])

 @app.route('/billp',methods=['POST','GET'])
 def billp():
     val={}
     price={}
     i=0
     billingid=request.form['bd']
     product=request.form['product']
     quantity=request.form['quantity']
     sql='SELECT * FROM  PRODUCTS WHERE PID=?'
     stmt=ibm_db.prepare(conn, sql)
     ibm_db.bind_param(stmt, 1, product)
     ibm_db.execute(stmt)
     price = ibm_db.fetch_assoc(stmt)
     name=str(price['PNAME'])
     if price!=False:
      unity=int(price['QUANTITY'])-int(quantity)
      sql2='UPDATE products SET QUANTITY =? WHERE PID = ?'
      prep_stmt=ibm_db.prepare(conn,sql2)
      ibm_db.bind_param(prep_stmt, 1, unity)
      ibm_db.bind_param(prep_stmt, 2, product)
      ibm_db.execute(prep_stmt)
      amount=int(price['PRICE'])*int(quantity)
      msg='success fully added'
      return(redirect(url_for('trial',amount=amount,product=product,quantity=quantity,bid=billingid,pname=name)))
     else:
```

```python
     msg="not good"
     return redirect(url_for('detail'))
@app.route('/trial/<amount>/<product>/<quantity>/<bid>/<pname>',methods=['POST','GET'])
def trial(amount,product,quantity,bid,pname):
    pid=product
    quantity=quantity
    bid=bid
    amount=amount
    pname=pname
    insert_sql="INSERT INTO BILLING VALUES(?,?,?,?,?)"
    prep_stmt=ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, bid)
    ibm_db.bind_param(prep_stmt, 2, pid)
    ibm_db.bind_param(prep_stmt, 3, quantity)
    ibm_db.bind_param(prep_stmt, 4, amount)
    ibm_db.bind_param(prep_stmt, 5, pname)
    ibm_db.execute(prep_stmt)

    insert_sql1="INSERT INTO BILLS VALUES(?,?,?,?,?)"
    prep_stmt1=ibm_db.prepare(conn, insert_sql1)
    ibm_db.bind_param(prep_stmt1, 1, bid)
    ibm_db.bind_param(prep_stmt1, 2, pid)
    ibm_db.bind_param(prep_stmt1, 3, quantity)
    ibm_db.bind_param(prep_stmt1, 4, amount)
    ibm_db.bind_param(prep_stmt1, 5, pname)

    ibm_db.execute(prep_stmt1)
    msg='success fully added'
    return redirect(url_for('detail',bid=bid))
@app.route('/detail/<bid>',methods=['POST','GET'])
def detail(bid):
    val={}
    bid=bid
    i=0
    total=0
    sqll="SELECT * FROM billing"
    stmt = ibm_db.exec_immediate(conn, sqll)
    dictionary = ibm_db.fetch_assoc(stmt)
    if dictionary!=False:
     val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
     total=total+int(dictionary['PRICE'])
     while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
       val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
           total=total+int(dictionary['PRICE'])
     msg="successfully added"
    else:
     msg="bad not added"
    return render_template("bill.html",msg=msg,account=val,total=total,name=session['name'],bid=bid,count=session['count'])

@app.route('/product', methods=['POST','GET'])
def product():
    pid=request.form["pid"]
    pname=request.form["pname"]
    price=request.form["price"]
    quantity=request.form["quantity"]
    insert_sql="INSERT INTO products VALUES(?,?,?,?)"
    prep_stmt=ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, pid)
    ibm_db.bind_param(prep_stmt, 2, pname)
    ibm_db.bind_param(prep_stmt, 3, price)
    ibm_db.bind_param(prep_stmt, 4, quantity)
    ibm_db.execute(prep_stmt)
    msg='success fully added'
    return redirect(url_for('display'))
@app.route('/delete', methods=['POST','GET'])
def delete():
    bid=request.form['billid']
    pid=request.form['productid']
    print(pid)
    quantity=request.form['quantity']
```

```python
    print(quantity)
    price=request.form['price']
    print(price)
    sql='SELECT * FROM  PRODUCTS WHERE PNAME=?'
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, pid)
    ibm_db.execute(stmt)
    pri = ibm_db.fetch_assoc(stmt)
    sql="DELETE FROM billing WHERE PNAME=?"
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, pid)
    ibm_db.execute(stmt)
    sql1="DELETE FROM BILLS WHERE PNAME=?"
    stmt1=ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1, 1, pid)
    ibm_db.execute(stmt1)
    msg="Successfully deleted"
    unity=int(pri['QUANTITY'])+int(quantity)
    sql2='UPDATE products SET QUANTITY =? WHERE PNAME= ?'
    prep_stmt=ibm_db.prepare(conn,sql2)
    ibm_db.bind_param(prep_stmt, 1, unity)
    ibm_db.bind_param(prep_stmt, 2,pid)
    ibm_db.execute(prep_stmt)
    return redirect(url_for('detail',bid=bid))

@app.route('/ADS',methods=['POST','GET'])
def ADS():
    sql="DELETE FROM BILLING"
    stmt = ibm_db.exec_immediate(conn, sql)
    msg="successfully validated"
    return redirect(url_for('statement'))
@app.route('/quantity')
def quantity():
    val={}
    i=0
    count=1
    sql='SELECT * FROM  PRODUCTS WHERE QUANTITY<=?'
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1,"5")
    ibm_db.execute(stmt)
```

```
        dictionary = ibm_db.fetch_assoc(stmt)
        if dictionary != False:
            val[i]={'productid':dictionary['PID'],'productname':dictionary['PNAME'],'quantity':dictionary['QUANTITY']}
         count=count+1
         while dictionary != False:
          i=i+1
          dictionary = ibm_db.fetch_assoc(stmt)
          if dictionary!=False:
              val[i]={'productid':dictionary['PID'],'productname':dictionary['PNAME'],'quantity':dictionary['QUANTITY']}
               count=count+1
         session['count']=count
         msg="successfully added"
         msgs = Message("the below items quantity is so less ,plese order quickly", sender = 'verifyemail0904@gmail.com', recipients=['shanjeyshanjey0@gmail.com'])
         msgs.body =str(val)
         mail.send(msgs)
        return render_template("home.html",count=session['count'])
    @app.route('/password/<email>',methods=['POST','GET'])
    def password(email):
        email=email
        print(email)
        sql="SELECT * FROM users WHERE email=?"
        prep_stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(prep_stmt, 1,email)
        ibm_db.execute(prep_stmt)
        detail=ibm_db.fetch_assoc(prep_stmt)
        username=detail['USERNAME']
        password=detail['PASSWORD']
        msgs = Message(subject='YOUR PASSWORD IS ', sender = 'verifyemail0904@gmail.com', recipients=[email])
        msgs.body =str(password)
        mail.send(msgs)
        return render_template('login1.html',count=session['count'])

    @app.route('/users')
    def users():
        val={}
        i=0
        sql="SELECT * FROM users"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_assoc(stmt)
        val[i]={'name':dictionary['USERNAME'],'email':dictionary['EMAIL'],'pass':dictionary['PASSWORD']}
        while dictionary != False:
          i=i+1
          dictionary = ibm_db.fetch_assoc(stmt)
          if dictionary!=False:
            val[i]={'name':dictionary['USERNAME'],'email':dictionary['EMAIL'],'pass':dictionary['PASSWORD']}
        print(*val.values())
        return render_template("users.html",account=val,name=session['name'],count=session['count'])
    @app.route('/all')
    def all():
        val={}
        i=0
        sql="SELECT * FROM products ORDER BY quantity"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_assoc(stmt)
        val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
        while dictionary != False:
          i=i+1
          dictionary = ibm_db.fetch_assoc(stmt)
          if dictionary!=False:
            val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
        print(*val.values())
```

```python
        return render_template("quantity.html",account=val,name=session['name'],count=session['count'])
    @app.route('/update', methods=['POST','GET'])
    def update():
        pid=request.form['pid']
        print(pid)
        quantity=request.form['quantity']
        print(quantity)
        price=request.form['pprice']
        print(price)
        sql="DELETE FROM products WHERE PID=?"
        stmt=ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, pid)
        ibm_db.execute(stmt)
        msg="Successfully deleted"
        return redirect(url_for('all'))

    @app.route('/fverify',methods = ["POST"])
    def fverify():
        email = request.form["email"]
        session['email']=request.form["email"]
        msg = Message(subject='OTP',sender = 'verifyemail0904@gmail.com', recipients = [email])
        msg.body = str(otp)
        mail.send(msg)
        return render_template('fverification.html',email=email)

    @app.route('/fvalidate',methods=["POST"])
    def fvalidate():
        user_otp = request.form['otp']
        email=session['email']
        if otp == int(user_otp):
            return redirect(url_for('password',email=email))
        return "<h3>failure</h3>"
    @app.route('/alterbill', methods=['POST','GET'])
    def alterbill():
        val={}
        i=0
        bid=request.form['billingid']
        print(bid)
        sql="SELECT*FROM BILLS WHERE BILLID=?"
        stmt=ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, bid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        bid=dictionary['BILLID']
        val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
        total=total+int(dictionary['PRICE'])
        while dictionary != False:
          i=i+1
          dictionary = ibm_db.fetch_assoc(stmt)
          if dictionary!=False:
               val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
               total=total+int(dictionary['PRICE'])
        print(*val.values())
        return render_template("bill.html",account=val,name=session['name'],bid=bid,total=total,count=session['count'])
    @app.route('/bills')
    def bills():
        val={}
        i=0
        sql="SELECT * FROM BILLS"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_assoc(stmt)
        if dictionary!=False:
            val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
         while dictionary != False:
          i=i+1
          dictionary = ibm_db.fetch_assoc(stmt)
          if dictionary!=False:

```

```
val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
        print(*val.values())
        return render_template("dashboard.html",account=val,name=session['name'],count=session['count'])

    if __name__=='__main__':
        app.run(host='0.0.0.0')

```

FEATURE 2 :

When the quantity is low it shows an message in Website.

CODE :

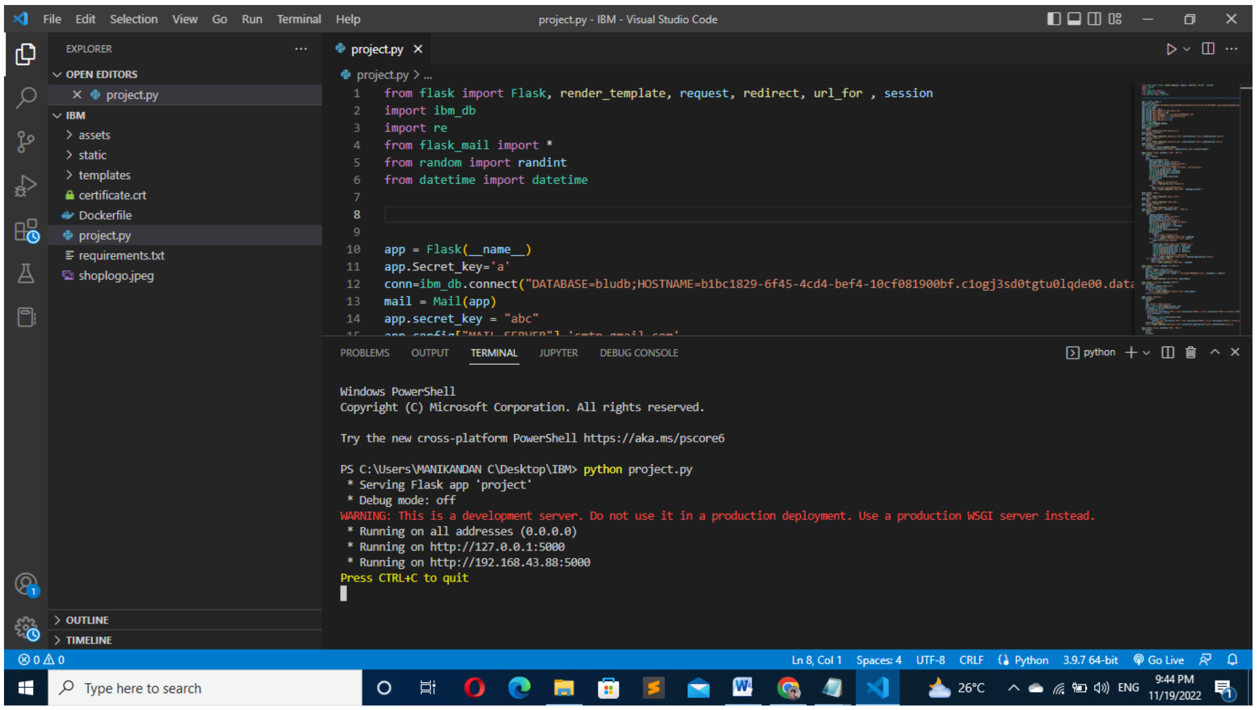
```
{% extends 'nav.html'%}
   {% block content %}

     <table class="table table-striped table-dark align-self-center" style="width:90%;position: relative;top:10px;left:10px">
 <thead>
   <tr>
     <th scope="col">pid</th>
     <th scope="col">pname</th>
     <th scope="col">price</th>
     <th scope="col">quantity</th>
     <th scope="col">delete</th>
   </tr>
 </thead>
 <tbody>
   <form action="/update" method="POST">
  {% for i in account.values()%}
   <tr>
     <td><input type="text" value={{i.pid}} name="pid" readonly></td>
     <td><input type="text" value={{i.name}} name="pname" readonly></td>
     <td><input type="text" value={{i.price}} name="pprice" readonly></td>
```

```
      <td><input type="text" value={{i.quantity}} name="quantity" readonly></td>
      <td><input type="submit" class="btn btn-primary" value="DELETE"id="button"></td>

    </tr>
    {%endfor%}
</form>
     </tbody>
</table>
<!-- JavaScript Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJuaOe923+mo//f6V8Qbsw3" crossorigin="anonymous"></script>
{% endblock%}
```

8. TESTCASES

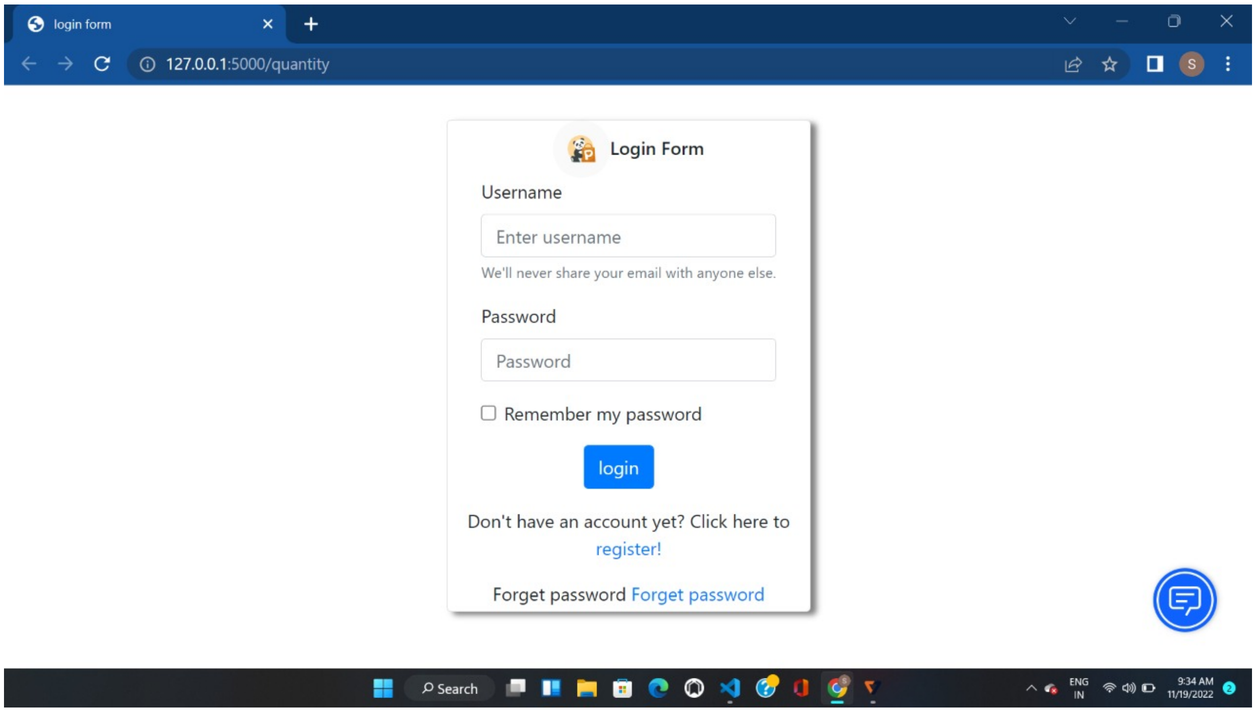
TESTINGS

The training output of the source code seen in the above chapter was executed successfully and got the output .



```python
from flask import Flask, render_template, request, redirect, url_for , session
import ibm_db
import re
from flask_mail import *
from random import randint
from datetime import datetime



app = Flask(__name__)
app.Secret_key='a'
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-4cd4-bef4-10cf081900bf.c1ogj3sd0tgtu0lqde00.data
mail = Mail(app)
app.secret_key = "abc"
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\MANIKANDAN C\Desktop\IBM> python project.py
 * Serving Flask app 'project'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.43.88:5000
Press CTRL+C to quit
```


login form
127.0.0.1:5000/quantity

Login Form

Username

Enter username

We'll never share your email with anyone else.

Password

Password

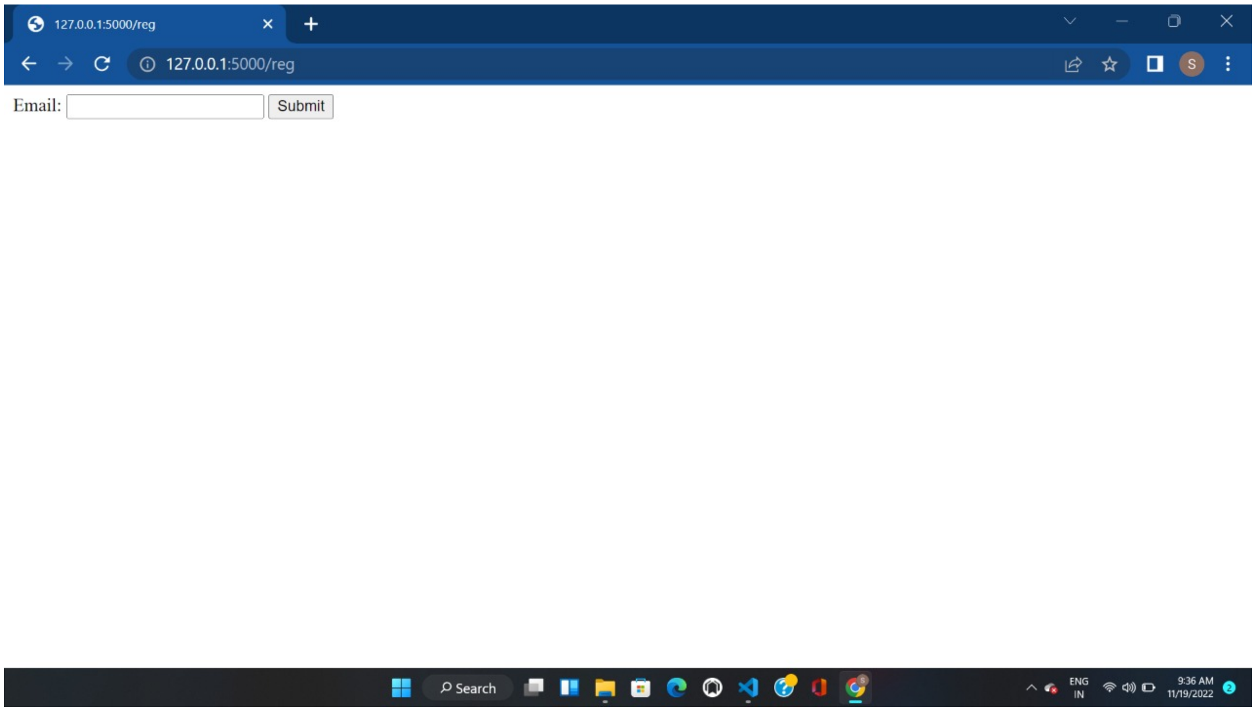
Remember my password

login

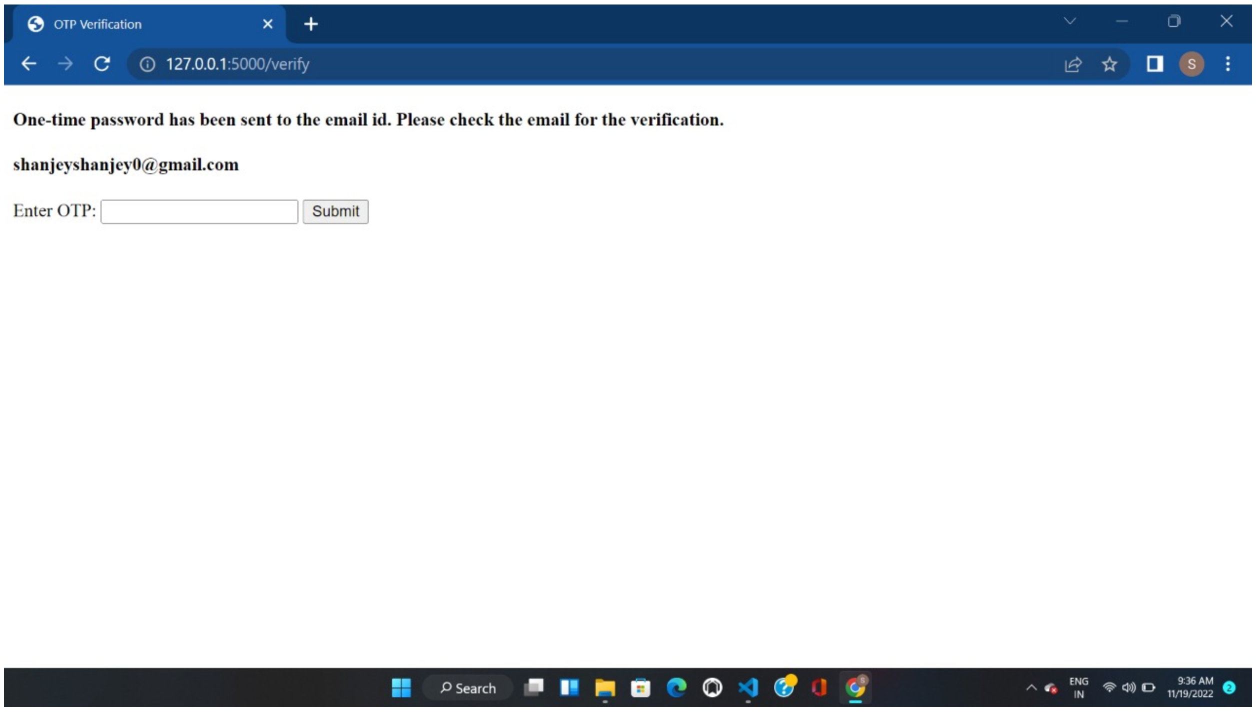
Don't have an account yet? Click here to register!

Forget password Forget password

REGISTER :


127.0.0.1:5000/reg

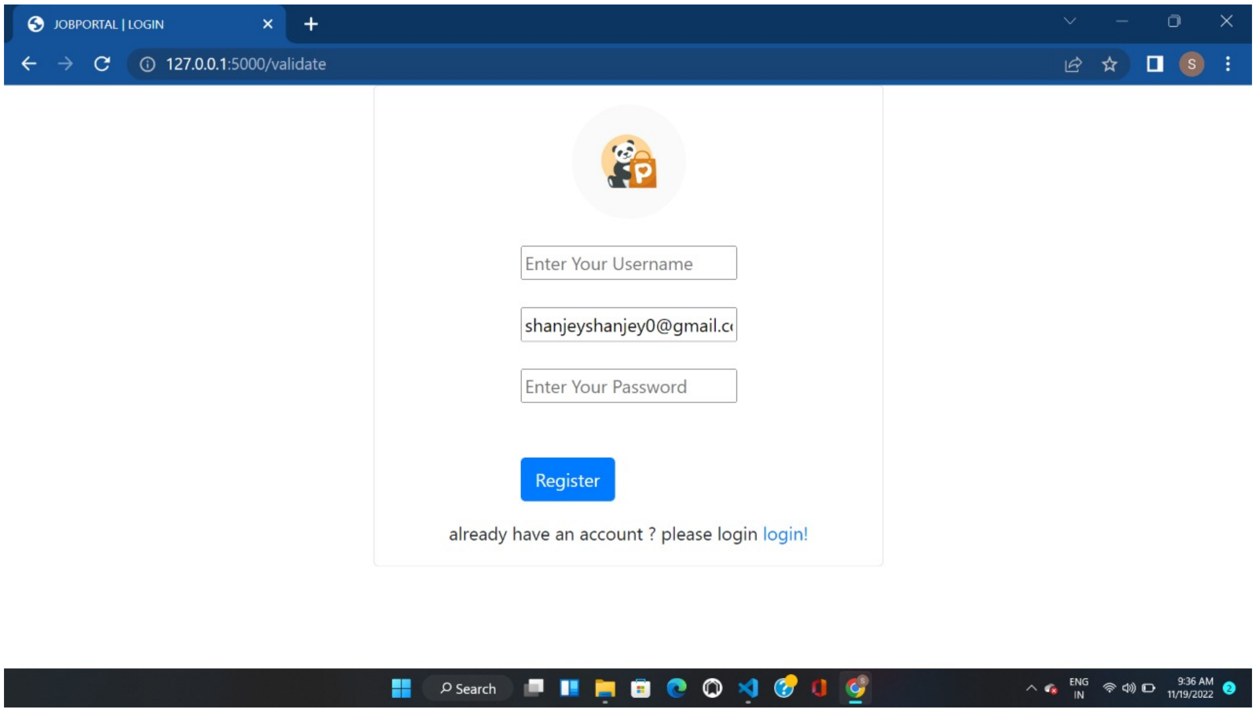
Email: Submit



One-time password has been sent to the email id. Please check the email for the verification.

shanjeyshanjey0@gmail.com

Enter OTP: Submit



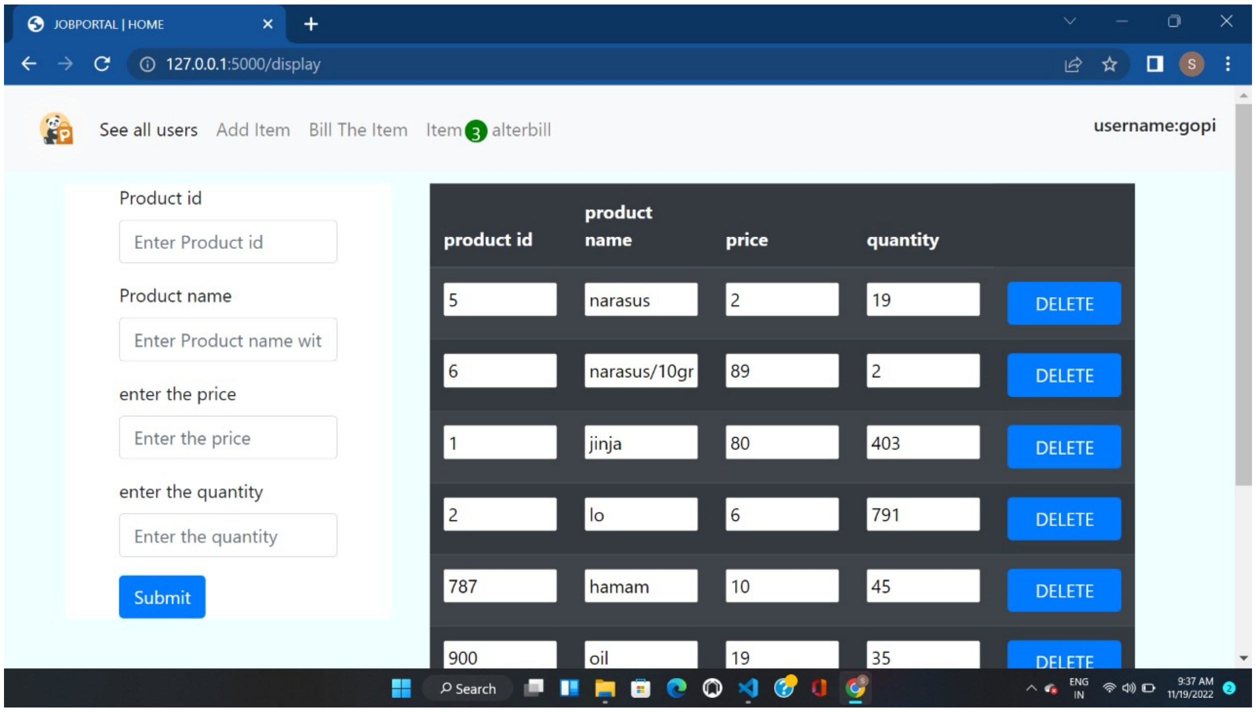
Enter Your Username

shanjeyshanjey0@gmail.c

Enter Your Password

Register

already have an account ? please login login!



Product id

Enter Product id

Product name

Enter Product name wit

enter the price

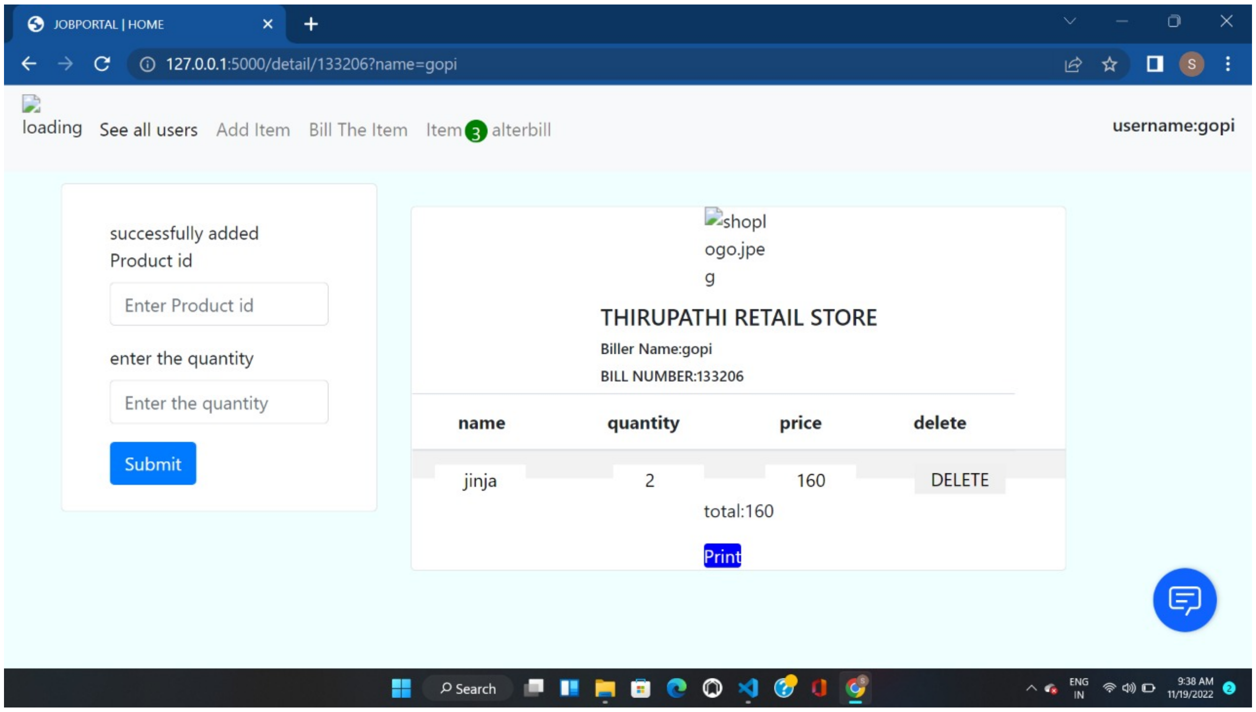
Enter the price

enter the quantity

Enter the quantity

Submit

product id	product name	price	quantity	
5	narasus	2	19	DELETE
6	narasus/10gr	89	2	DELETE
1	jinja	80	403	DELETE
2	lo	6	791	DELETE
787	hamam	10	45	DELETE
900	oil	19	35	DELETE



successfully added

Product id

Enter Product id

enter the quantity

Enter the quantity

Submit

shoplogo.jpeg

THIRUPATHI RETAIL STORE

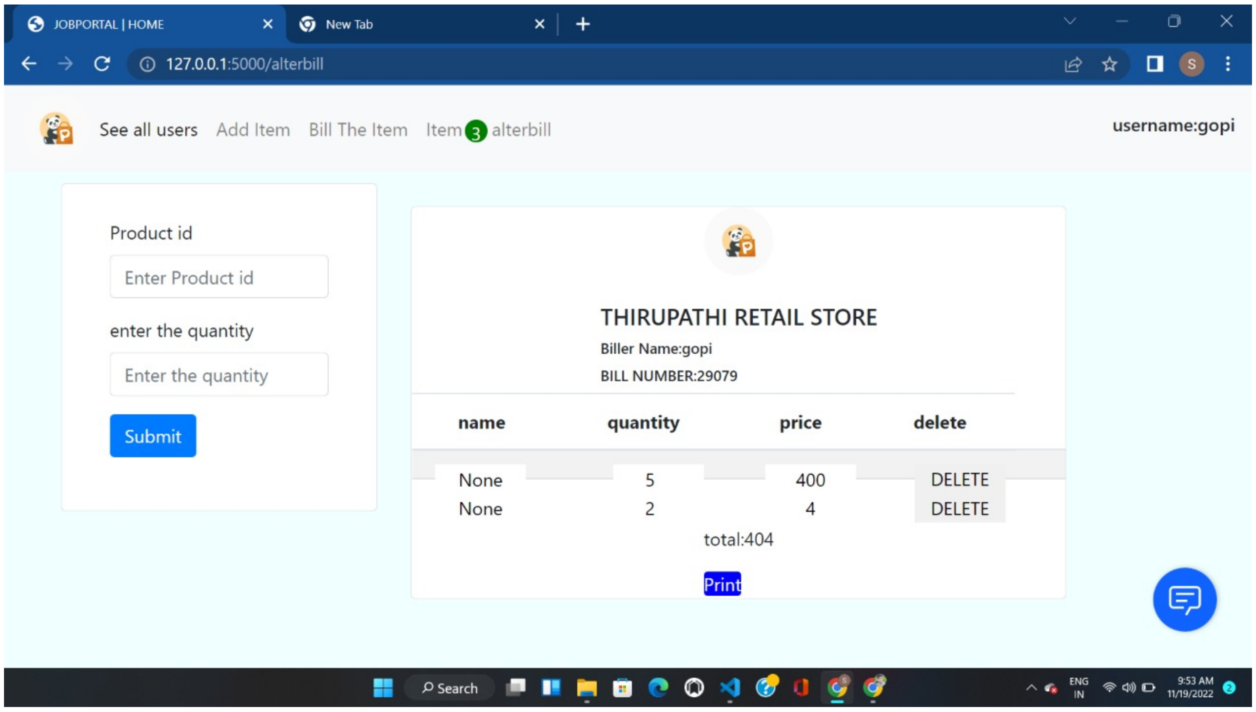
Biller Name:gopi

BILL NUMBER:133206

name	quantity	price	delete
jinja	2	160	DELETE

total:160

Print



Product id

Enter Product id

enter the quantity

Enter the quantity

Submit

THIRUPATHI RETAIL STORE

Biller Name:gopi

BILL NUMBER:29079

name	quantity	price	delete
None	5	400	DELETE
None	2	4	DELETE

total:404

Print

9. RESULTS :

We have successfully completed the project works that the Inventory Management For System Retailers using the web application.

10. ADVANTAGES AND DISADVANTAGES

ADVANTAGE :

1. **It helps to maintain the right amount of stocks:** contrary to the belief that is held by some people, inventory management does not seek to reduce the amount of inventory that you have in stock, however, 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. The goal is to find that zone where you are never losing money in your inventory in either direction. With the aid of an efficient inventory management strategy, it is easy to improve the accuracy of inventory order.
2. **It leads to a more organized warehouse:** with the aid of a good inventory

management system, you can easily organize your warehouse. If your warehouse is not organized, you will find it very difficult to manage your inventory. A lot of businesses choose to optimize their warehouse by putting the items that have the highest sales together in a place that is easy to access in the warehouse. This ultimately helps to speed up order fulfilment and keeps clients happy.

3. Increased information transparency: a good inventory management helps to keep the flow of information transparent. This information includes when items were received, picked, packed, shipped, manufactured et al. You also get to know when you need to order more of any good, when you have too much stock or too little stock

DISADVANTAGE :

1. **Production problem:** even though inventory management can reveal to you the amount of stock you have at hand and the amount that you have sold off, it can also hide production problems that could lead to customer service disasters. Since the management places almost all of its focus on inventory management to the detriment of quality control, broken or incorrect items that would normally be discarded are shipped along with wholesome items.
2. **Increased space is need to hold the inventory:** in order to hold inventory, you will need to have space so unless the goods you deal in are really small in size, then you will need a warehouse to store it. In addition, you will also need to buy shelves and racks to store your goods, forklifts to move around the stock and of course staff. The optimum level of inventory for a business could still be a lot of goods and they will need space to be stored in and in some cases additional operational costs to manage the inventory. This will in turn increase cost and impact negatively on the amount of profit the business makes.
3. **Complexity:** some methods and strategies of inventory management can be relatively complex and difficult to understand on the part of the staff. This may result in the need for employees to undergo training in order to grasp how the system works.
4. Some inventory management systems such as the fixed order period system compels a periodic review of all items. This itself makes the system a bit inefficient.
5. **High implementation costs:** some inventory management systems can come at a high price because the business needs to install specialized systems and software in order to use them. This can be problematic for large businesses which operate in difficult locations.

11 . CONCLUSION

Conclusion Inventory management is a very complex but essential part of the supply chain. An effective inventory management system helps to reduce stock-related costs such as warehousing, carrying, and ordering costs.

12 . FUTURE SCOPE :

According to Easy Post, '**Companies can reap a 25% increase in productivity, a 20% gain in space usage, and a 30% improvement in stock use efficiency if they use integrated order processing for their inventory system**. Advanced mobile applications allow companies to manage their inventory and supply chains effectively.

13 . APPENDIX :

Source code :
python(file) :

```python
from flask import Flask, render_template, request, redirect, url_for , session
import ibm_db
import re
from flask_mail import *
from random import randint
from datetime import datetime



app = Flask(__name__)
app.Secret_key='a'
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-4cd4-bef4-10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32304;SECURITY=SSL;SSLServerCertificate=certificate.crt;UID=ngw72
704;PWD=mzQy2ksb3Ff6i3Ex",'','')
mail = Mail(app)
app.secret_key = "abc"
app.config["MAIL_SERVER"]='smtp.gmail.com'
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = 'verifyemail0904@gmail.com'
app.config['MAIL_PASSWORD'] = 'fkchuaznhiwjjyuq'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
otp = randint(000000,999999)
date=datetime.now()
@app.route('/')
def home():
    return redirect(url_for('quantity'))
@app.route('/additem')
def additem():
    return render_template('addproduct.html',count=session['count'],name=session['name'])
@app.route('/alter')
def alter():
    return render_template('productid.html',count=session['count'],name=session['name'])
@app.route('/statement')
def statement():
    billingid = randint(000000,999999)
    return redirect(url_for('detail',name=session['name'],bid=billingid))

@app.route('/login',methods= ['GET','POST'])
def login():
    global userid
    msg=''
    if request.method=='POST':
        username=request.form['username']
        session['name']=request.form['username']
        password=request.form['password']
        sql="SELECT * FROM users WHERE username= ? and password=?"
```

```python
        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)
        print(account)
        if account:
           msg='logged in successfully!'
           return redirect(url_for('display'))
        else:
           msg='incorrect Username/Password !'
           return render_template("login.html", msg=msg,account="")

@app.route('/team')
def team():
    return render_template('login1.html')
@app.route('/reg')
def reg():
    return render_template('index.html')
@app.route('/forget')
def forget():
    return render_template('vindex.html')
@app.route('/register' , methods=['GET' , 'POST'])
def register():
     msg=''
     if request.method=='POST':
        username=request.form['username']
        email=request.form['email']
        password=request.form['password']
        sql="SELECT * FROM users WHERE username=?"
        stmt=ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account=ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
             msg='account already exits'
             return render_template('index.html',msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+',email):
            msg='invalid email address'
        else:
            insert_sql="INSERT INTO users VALUES(?,?,?)"
            prep_stmt=ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, username)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg='you have successfully registered'
            return render_template("login1.html",msg=msg,name=session['name'])
     elif request.method =='POST':
          msg='please fill out the form'
          return render_template('index.html', msg=msg)

 @app.route('/verify',methods = ["POST"])
 def verify():
     email = request.form["email"]
     session['email']=request.form["email"]
     msg = Message(subject='OTP',sender = 'verifyemail0904@gmail.com', recipients = [email])
     msg.body = str(otp)
     mail.send(msg)
     return render_template('verify.html',email=email)

 @app.route('/validate',methods=["POST"])
 def validate():
     user_otp = request.form['otp']
     email=session['email']
     if otp == int(user_otp):
         return render_template('register.html',email=email)
     return "<h3>failure</h3>"

 @app.route('/display')
 def display():
     val={}
     i=0
     sql="SELECT * FROM products"
     stmt = ibm_db.exec_immediate(conn, sql)
     dictionary = ibm_db.fetch_assoc(stmt)
```

```python
    if dictionary!=False:
     val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
     while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
       val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
    print(*val.values())
    return render_template("dashboard.html",account=val,name=session['name'],count=session['count'])

@app.route('/billp',methods=['POST','GET'])
def billp():
    val={}
    price={}
    i=0
    billingid=request.form['bd']
    product=request.form['product']
    quantity=request.form['quantity']
    sql='SELECT * FROM  PRODUCTS WHERE PID=?'
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, product)
    ibm_db.execute(stmt)
    price = ibm_db.fetch_assoc(stmt)
    name=str(price['PNAME'])
    if price!=False:
     unity=int(price['QUANTITY'])-int(quantity)
     sql2='UPDATE products SET QUANTITY =? WHERE PID = ?'
     prep_stmt=ibm_db.prepare(conn,sql2)
     ibm_db.bind_param(prep_stmt, 1, unity)
     ibm_db.bind_param(prep_stmt, 2, product)
     ibm_db.execute(prep_stmt)
     amount=int(price['PRICE'])*int(quantity)
     msg='success fully added'

```

```
        return(redirect(url_for('trial',amount=amount,product=product,quantity=quantity,bid=billingid,pname=name)))
    else:
     msg="not good"
     return redirect(url_for('detail'))
@app.route('/trial/<amount>/<product>/<quantity>/<bid>/<pname>',methods=['POST','GET'])
def trial(amount,product,quantity,bid,pname):
    pid=product
    quantity=quantity
    bid=bid
    amount=amount
    pname=pname
    insert_sql="INSERT INTO BILLING VALUES(?,?,?,?,?)"
    prep_stmt=ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, bid)
    ibm_db.bind_param(prep_stmt, 2, pid)
    ibm_db.bind_param(prep_stmt, 3, quantity)
    ibm_db.bind_param(prep_stmt, 4, amount)
    ibm_db.bind_param(prep_stmt, 5, pname)
    ibm_db.execute(prep_stmt)

    insert_sql1="INSERT INTO BILLS VALUES(?,?,?,?,?)"
    prep_stmt1=ibm_db.prepare(conn, insert_sql1)
    ibm_db.bind_param(prep_stmt1, 1, bid)
    ibm_db.bind_param(prep_stmt1, 2, pid)
    ibm_db.bind_param(prep_stmt1, 3, quantity)
    ibm_db.bind_param(prep_stmt1, 4, amount)
    ibm_db.bind_param(prep_stmt1, 5, pname)

    ibm_db.execute(prep_stmt1)
    msg='success fully added'
    return redirect(url_for('detail',bid=bid))
@app.route('/detail/<bid>',methods=['POST','GET'])
def detail(bid):
    val={}
    bid=bid
    i=0
    total=0
    sqll="SELECT * FROM billing"
```

```python
    stmt = ibm_db.exec_immediate(conn, sqll)
    dictionary = ibm_db.fetch_assoc(stmt)
    if dictionary!=False:
     val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
     total=total+int(dictionary['PRICE'])
     while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
       val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
           total=total+int(dictionary['PRICE'])
     msg="successfully added"
    else:
     msg="bad not added"
    return render_template("bill.html",msg=msg,account=val,total=total,name=session['name'],bid=bid,count=session['count'])

@app.route('/product', methods=['POST','GET'])
def product():
    pid=request.form["pid"]
    pname=request.form["pname"]
    price=request.form["price"]
    quantity=request.form["quantity"]
    insert_sql="INSERT INTO products VALUES(?,?,?,?)"
    prep_stmt=ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, pid)
    ibm_db.bind_param(prep_stmt, 2, pname)
    ibm_db.bind_param(prep_stmt, 3, price)
    ibm_db.bind_param(prep_stmt, 4, quantity)
    ibm_db.execute(prep_stmt)
    msg='success fully added'
    return redirect(url_for('display'))
@app.route('/delete', methods=['POST','GET'])
def delete():
    bid=request.form['billid']
    pid=request.form['productid']
    print(pid)
    quantity=request.form['quantity']
    print(quantity)
    price=request.form['price']
    print(price)
    sql='SELECT * FROM  PRODUCTS WHERE PNAME=?'
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, pid)
    ibm_db.execute(stmt)
    pri = ibm_db.fetch_assoc(stmt)
    sql="DELETE FROM billing WHERE PNAME=?"
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, pid)
    ibm_db.execute(stmt)
    sql1="DELETE FROM BILLS WHERE PNAME=?"
    stmt1=ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1, 1, pid)
    ibm_db.execute(stmt1)
    msg="Successfully deleted"
    unity=int(pri['QUANTITY'])+int(quantity)
    sql2='UPDATE products SET QUANTITY =? WHERE PNAME= ?'
    prep_stmt=ibm_db.prepare(conn,sql2)
    ibm_db.bind_param(prep_stmt, 1, unity)
    ibm_db.bind_param(prep_stmt, 2,pid)
    ibm_db.execute(prep_stmt)
    return redirect(url_for('detail',bid=bid))

@app.route('/ADS',methods=['POST','GET'])
def ADS():
    sql="DELETE FROM BILLING"
    stmt = ibm_db.exec_immediate(conn, sql)
    msg="successfully validated"
    return redirect(url_for('statement'))
@app.route('/quantity')
def quantity():
    val={}
    i=0
    count=1
    sql='SELECT * FROM  PRODUCTS WHERE QUANTITY<=?'
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1,"5")
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    if dictionary != False:
     val[i]={'productid':dictionary['PID'],'productname':dictionary['PNAME'],'quantity':dictionary['QUANTITY']}
     count=count+1
     while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
       val[i]={'productid':dictionary['PID'],'productname':dictionary['PNAME'],'quantity':dictionary['QUANTITY']}
           count=count+1
     session['count']=count
     msg="successfully added"
     msgs = Message("the below items quantity is so less ,plese order quickly", sender = 'verifyemail0904@gmail.com', recipients=['shanjeyshanjey0@gmail.com'])
     msgs.body =str(val)
     mail.send(msgs)
    return render_template("home.html",count=session['count'])
@app.route('/password/<email>',methods=['POST','GET'])
def password(email):
    email=email
    print(email)
    sql="SELECT * FROM users WHERE email=?"
    prep_stmt=ibm_db.prepare(conn,sql)
    ibm_db.bind_param(prep_stmt, 1,email)
    ibm_db.execute(prep_stmt)
    detail=ibm_db.fetch_assoc(prep_stmt)
    username=detail['USERNAME']
    password=detail['PASSWORD']
    msgs = Message(subject='YOUR PASSWORD IS ', sender = 'verifyemail0904@gmail.com', recipients=[email])
    msgs.body =str(password)
    mail.send(msgs)
```

```python
    return render_template('login1.html',count=session['count'])

@app.route('/users')
def users():
    val={}
    i=0
    sql="SELECT * FROM users"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    val[i]={'name':dictionary['USERNAME'],'email':dictionary['EMAIL'],'pass':dictionary['PASSWORD']}
    while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
        val[i]={'name':dictionary['USERNAME'],'email':dictionary['EMAIL'],'pass':dictionary['PASSWORD']}
    print(*val.values())
    return render_template("users.html",account=val,name=session['name'],count=session['count'])
@app.route('/all')
def all():
    val={}
    i=0
    sql="SELECT * FROM products ORDER BY quantity"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
    while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
        val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
    print(*val.values())
    return
render_template("quantity.html",account=val,name=session['name'],
count=session['count'])
@app.route('/update', methods=['POST','GET'])
def update():
    pid=request.form['pid']
    print(pid)
    quantity=request.form['quantity']
    print(quantity)
    price=request.form['pprice']
    print(price)
    sql="DELETE FROM products WHERE PID=?"
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, pid)
    ibm_db.execute(stmt)
    msg="Successfully deleted"
    return redirect(url_for('all'))

@app.route('/fverify',methods = ["POST"])
def fverify():
    email = request.form["email"]
    session['email']=request.form["email"]
    msg = Message(subject='OTP',sender =
'verifyemail0904@gmail.com', recipients = [email])
    msg.body = str(otp)
    mail.send(msg)
    return render_template('fverification.html',email=email)

@app.route('/fvalidate',methods=["POST"])
def fvalidate():
    user_otp = request.form['otp']
    email=session['email']
    if otp == int(user_otp):
        return redirect(url_for('password',email=email))
    return "<h3>failure</h3>"
@app.route('/alterbill', methods=['POST','GET'])
def alterbill():
    val={}
    i=0
    bid=request.form['billingid']
    print(bid)
    sql="SELECT*FROM BILLS WHERE BILLID=?"
    stmt=ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, bid)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    bid=dictionary['BILLID']
    val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
    total=total+int(dictionary['PRICE'])
    while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:
        val[i]={'product':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
           total=total+int(dictionary['PRICE'])
    print(*val.values())
    return render_template("bill.html",account=val,name=session['name'],bid=bid,total=total,count=session['count'])
@app.route('/bills')
def bills():
    val={}
    i=0
    sql="SELECT * FROM BILLS"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    if dictionary!=False:
        val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
     while dictionary != False:
      i=i+1
      dictionary = ibm_db.fetch_assoc(stmt)
      if dictionary!=False:

```

```
val[i]={'pid':dictionary['PID'],'name':dictionary['PNAME'],'price':dictionary['PRICE'],'quantity':dictionary['QUANTITY']}
    print(*val.values())
    return render_template("dashboard.html",account=val,name=session['name'],count=session['count'])

if __name__=='__main__':
    app.run(host='0.0.0.0')

```

TEMPLATES :

FRONTEND PAGES..

DEMO VIDEO:

DEMO VIDEO..