PROJECT REPORT

Date	**18.11.2022**
Team ID	**PNT2022TMID16711**
Project Name	**Inventory Management System For Retailers**
Team Members	**NavinRaj.G (111919205027)** **Karthik.S (111919205019)** **Vijay.D (111919205051)** **Gokul Krisna.V (111919205010)** **Palani Soundar.D (111919205030)**

INDEX

1. INTRODUCTION

Project Overview

The objective of this system is to manage the items in an inventory such as tracking orders, placing orders to other suppliers and checking the items in the inventory. The system allows the admin to maintain the items in the inventory.

Whenever the item levels go low, the system places an order to the supplier. The supplier gets the notification of these orders as soon as they are placed and can send the items to the inventory. There are two login pages each for the admin and supplier.

The software has been developed using the most powerful and secured backend Python and IBM Cloud for the databases and most widely accepted frontend JavaScript with HTML and CSS coding

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

Retail management refers to the process of helping customers find products in your store. It includes everything from increasing your customer pool to how products are presented, and how you fulfill a customer's needs. A good store manager helps customers leave the store with a smile.

2. LITERATURE SURVEY

Existing problem

- The problem faced by the company is they do not have any systematic system to record and keep their inventory data. It is difficult for the admin to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.
- Good planning and sales forecast before setting optimal inventory levels, appropriate inventory management requires close coordination between the areas of sales, purchasing and finance.

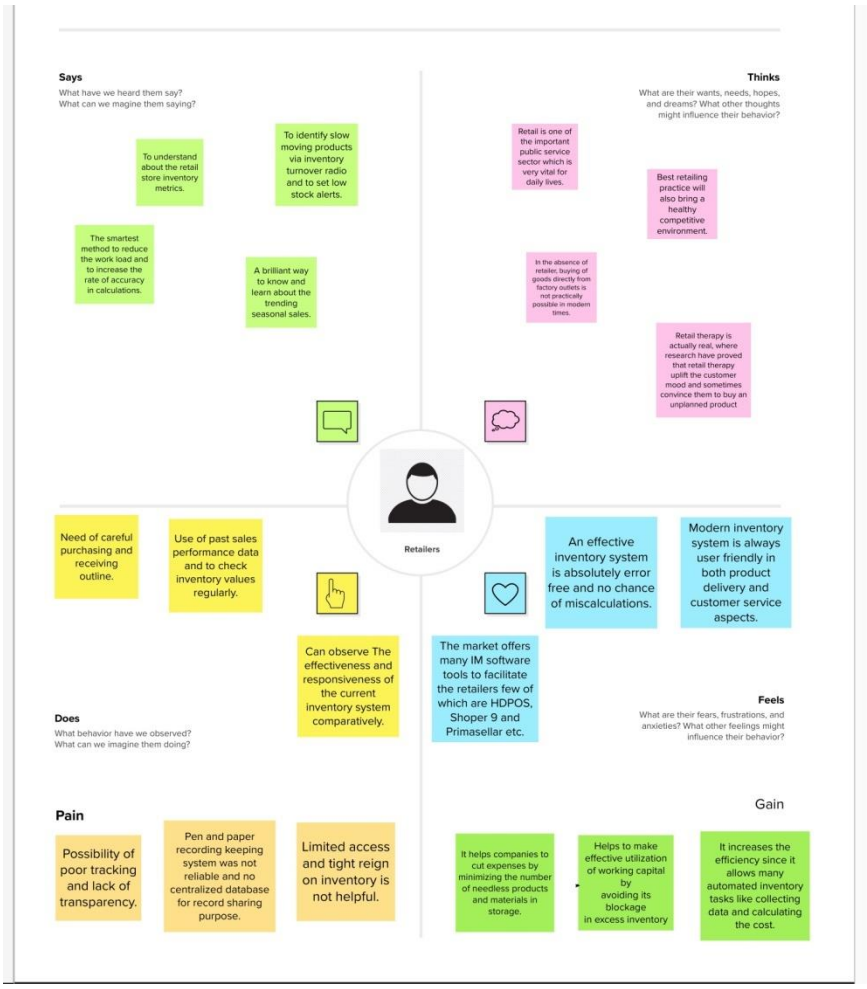
Problem Statement Definition

Retail inventory management works by creating systems to log products, receive them into inventory, track changes when sales occur, manage the flow of goods from purchasing to final sale and check stock counts.

3. IDEATION & PROPOSED SOLUTION

Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

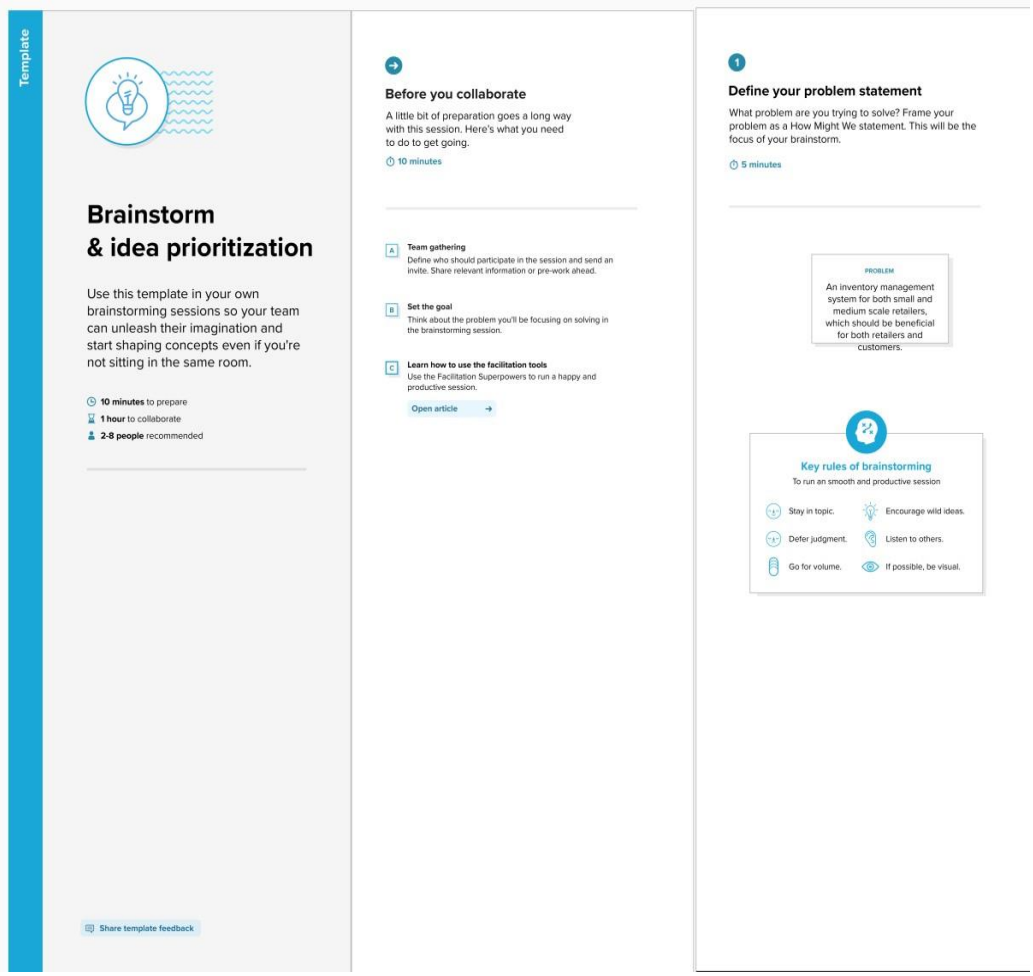


Ideation & Brainstorming

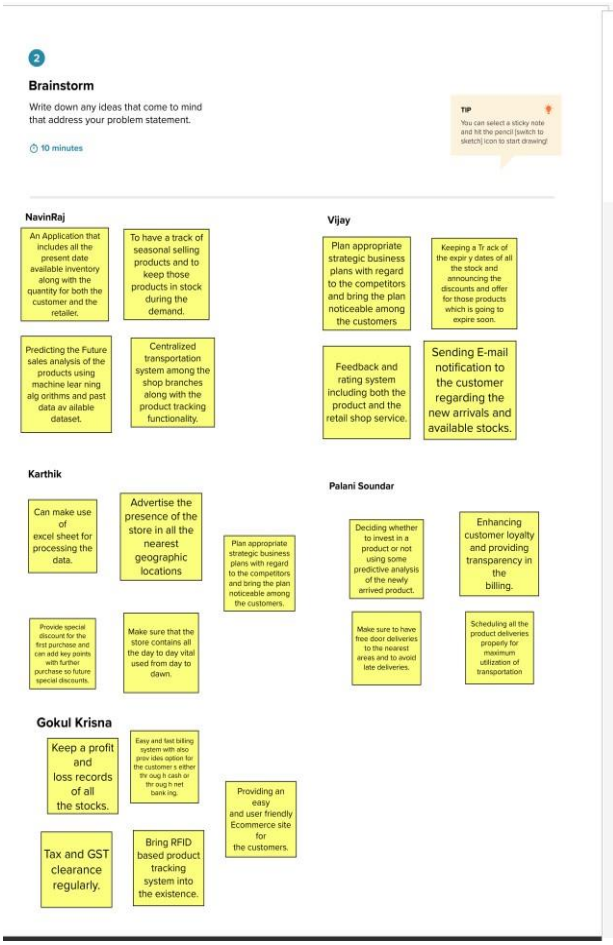
Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

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

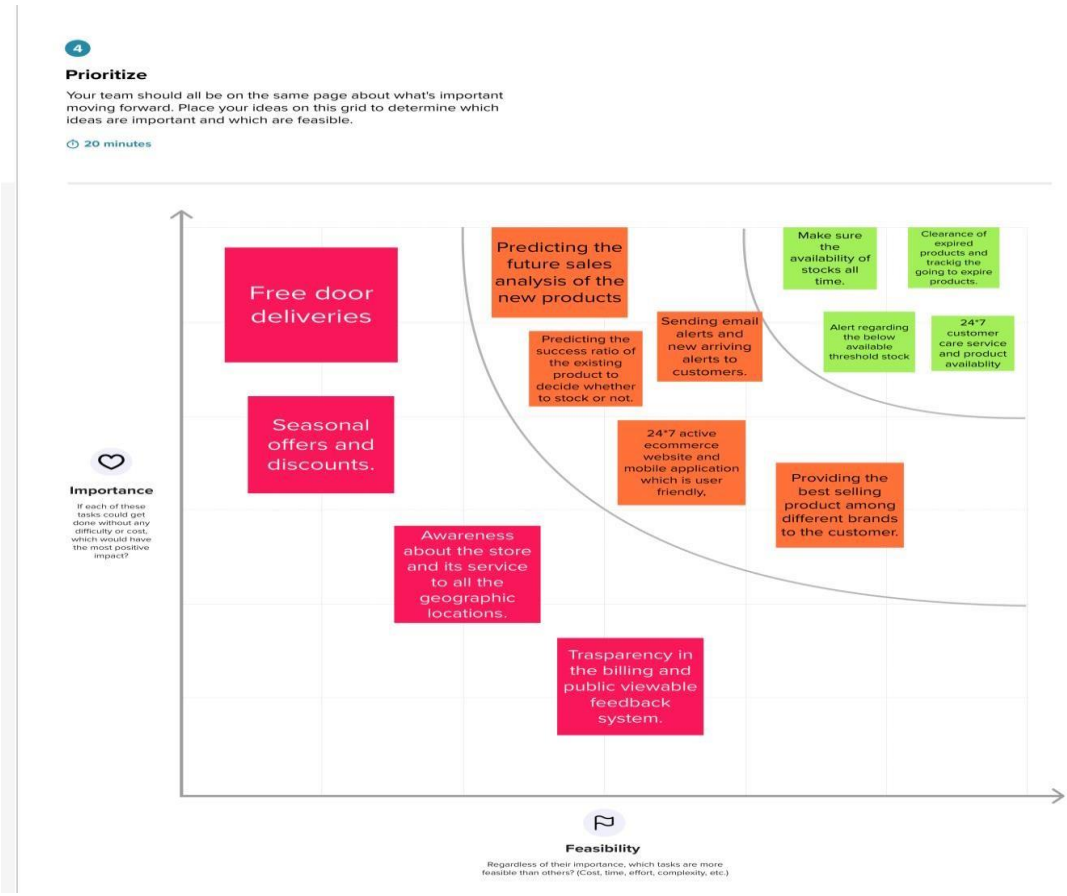
Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping



Step-3: Idea Prioritization



Proposed Solution

The system customizes and only shows recommended jobs based on the user's skill set and preferences (Using graphql api)
Similarly, the same recommendation system helps provide job applicant recommendations to the job recruiters to find the most eligible candidates for their firm.
All important data - job seeker's and hoster's personal information needs to be also stored safely and securely. Using a sql database is the most easiest, safest and convienent way possible.
Data needs to also be private in some cases like when information is shared with the host while applying for a job.

Problem Solution fit

Customer Problem Statement

4. REQUIREMENT ANALYSIS

Functional requirement

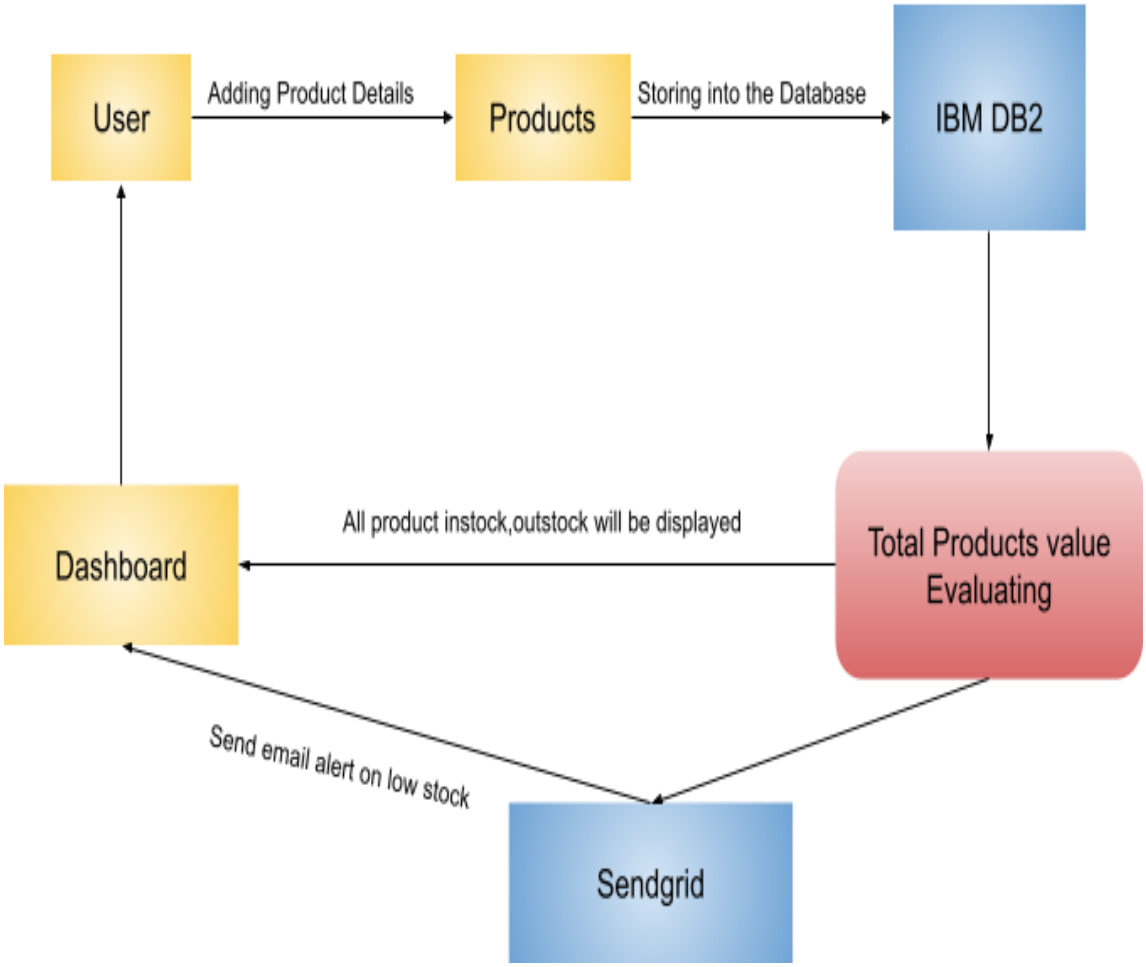
- The System aims at providing an efficient interface to the user for managing of inventory, it shall also provide the user varied options for managing the inventory through various functions at hand. The ingredient levels are continuously monitored based on their usage and are checked for the threshold levels in the inventory and accordingly the user is alerted about low levels of certain ingredients. The design is such that the user does not have to manually update the inventory every time, the System does if for the user.
- The System calculates and predicts the amount of usage for specific set days that are pre-set by the user(admin) , it also alerts the user of an impending action to order ingredients before the specific day set by the user. Therefore the user never has to worry about manually calculating the estimated usage of the ingredients as the System does it for the user.
- The simple interface of the System has functions like adding a recipe, removing or updating the recipe. It also extends to functions such as adding a vendor for an ingredient,, removing the vendor, checking threshold levels, processing orders, altering processed orders etc.

Non-Functional requirements

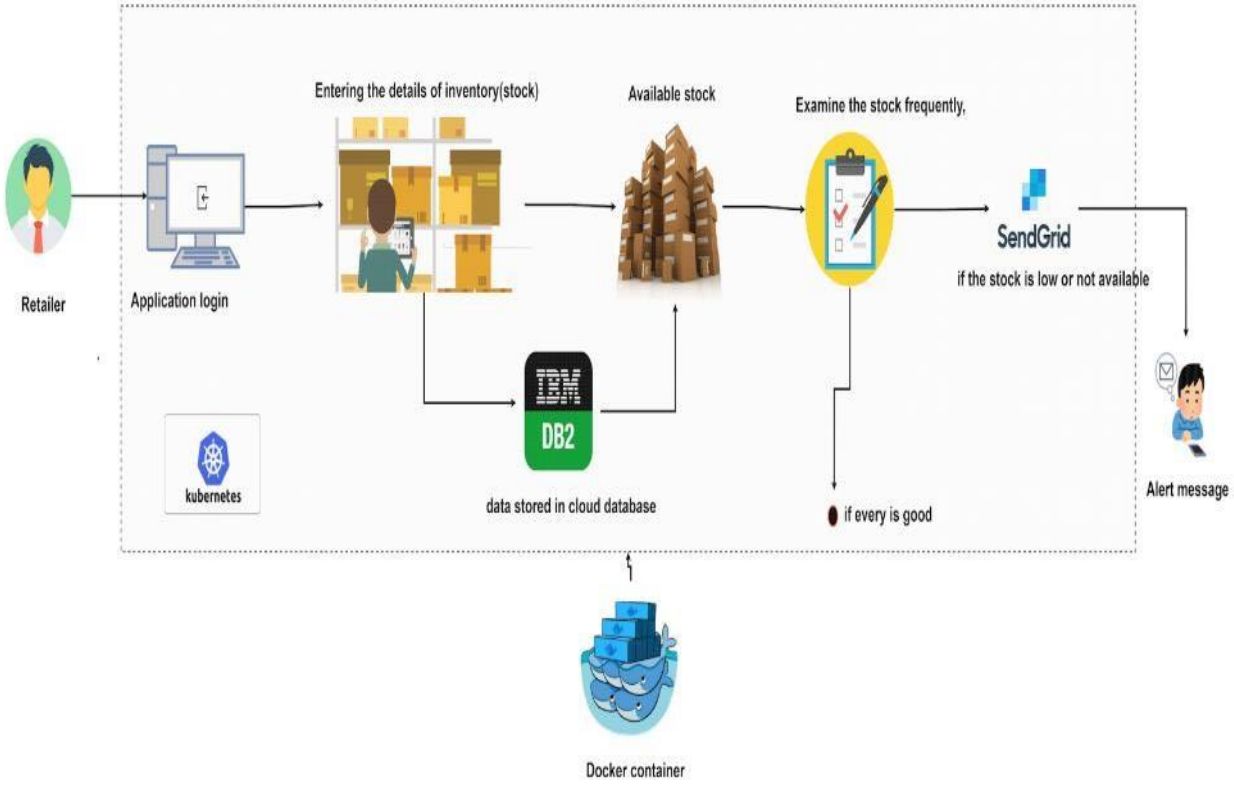
- The system must not lag, because the workers using it don't have down-time to wait for it to complete an action.
- The system must complete updating the databases, adding of recipe, ingredient, vendor and occasions successfully every time the user requests such a process.
- All the functions of the system must be available to the user every time the system is turned on.
- The calculations performed by the system must comply according to the norms set by the user and should not vary unless explicitly changed by the user
- The System must give accurate inventory status to the user continuously. Any inaccuracies are taken care by the regular confirming of the actual levels with the levels displayed in the system.
- The System must successfully add any recipe, ingredients, vendors or special occasions given by the user and provide estimations and inventory status in relevance with the newly updated entities.

5. PROJECT DESIGN

Data Flow Diagrams



Solution & Technical Architecture



User Stories

User Type	Functional Requirement(Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Retailer	Registration	USN-1	As a user, I can register for the application byentering 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	Medium	Sprint-1
	Login	USN-3	As a user, I can log into the application byentering email & password	I can access my account /dashboard	High	Sprint-1
	Dashboard	USN-4	As a user, I can view the stock list andsuppliers list	Once I log in to the system, I can able to view the stocks	Medium	Sprint-1
	Items	USN-5	As a user, I can add the items.	I can create a new type ofitem	High	Sprint-2
		USN-6	As a user, I can see the items	I can be able to see the items that can be added to the inventory	Low	Sprint-2
	Inventory	USN-7	As a user, I can add the items to inventory.	I can add items to the inventory with quantity	High	Sprint-2
		USN-8	As a user, I can see the items in the inventory.	I can see the inventory items with quantity	Low	Sprint-2
	Indication	USN-9	As a user, I can be able to receive indication	I receive a notification when the stock running low	High	Sprint-3
	Location	USN-10	As a user, I can be able to see items from a particular store location	I can be able to make purchase from a particularlocation	Medium	Sprint-3
		USN-11	As a user, I can add a new location of my store	I can be able to add new store locations	Medium	Sprint - 3
Customer	Purchase	USN -12	As a customer, I can be able to purchase goodfrom the particular location of the store	I can able to purchasefrom the store	High	Sprint - 4
Retailer & Customer	Deployment	USN-13	As a user, I can access the software in the web	I can access the software in web	High	Sprint -4

6. PROJECT PLANNING & SCHEDULING

Sprint Planning & Estimation

Use the below template to create product backlog and sprint 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.	5	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	4	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-1		USN-3	As a user, I can register for the application through Gmail	3	Medium	G Navin Raj S Karthik
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	4	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-1	Dashboard	USN-5	As a user, I can see the stock in hand and how much stock will be received and check other details.	4	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-2	Customer details	USN-6	As a user, I can see the customer details like name, company, location, and so on.	3	Low	D Vijay V Gokul Krisna
Sprint-2	Invoice management	USN-7	As a user, I can see, manage, and update or modify the invoice of my shop	1	Low	V Gokul Krisna D Palani Soundar

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Sale and order management	USN-8	As a user, I can see, manage, and update the sale and order	5	Medium	S Karthik V Gokul Krisna
Sprint-2	Return management	USN-9	As a user, I can manage the returned items and check for damaged or defective items.	5	Medium	D Vijay D Palani Soundar
Sprint-2	Purchase order management	USN-10	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	G Navin Raj V Gokul Krisna
Sprint-3	Stocks	USN-11	As a user, I can see the stock level, fast-moving, and death stocks.	4	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-3	Report	USN-12	As a user, I can see the report of the stock	1	Low	S Karthik D Palani Soundar
Sprint-3	Notification	USN-13	As a user, it is good if I get a notification for low stock.	2	Medium	G Navin Raj D Vijay
Sprint-3	Supplier	USN-14	As a user, I can see the supplier details for a better understanding.	3	Low	S Karthik D Palani Soundar
Sprint-2	Profile	USN-15	As a user, I can see my profile and give my details after registering as well.	1	Low	D Vijay V Gokul Krisna
Sprint-3	bill	USN-16	As a user, I like to print the product the sold now and maintain it.	4	Medium	G Navin Raj D Palani Soundar

Download file | iLovePDF
IBM-Project-15827-1659605000/
IBM
github.com/IBM-EPBL/IBM-Project-15827-1659605000/blob/main/Project%20Design%20%26%20Planning/Project%20Planning%20Phase/Sprint%20Delivery%20P...

Sprint-3	bill	USN-16	As a user, I like to print the product the sold now and maintain it.	4	Medium	G Navin Raj D Palani Soundar
Sprint-3	Chatbot	USN-17	As a customer care executive,I can view the complaints on chat box, As a customer, I should be able solve and reply for the customers queries and as a customer, I can close the complaint after assisting	4	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar
Sprint-4	Containerization	USN-18	As a user, I can access the software with high performance	10	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-4	Deployement	USN-19	As a user, I can access the software in the web	10	High	G Navin Raj S Karthik D Vijay V Gokul Krisna D Palani Soundar

Problem Solution....docx
Problem Solution....pdf
Ideation-Brainsto....docx
Empathy Map (1).docx
Ideation-Brainstor....pdf
Show all
Type here to search
29°C
ENG IN
19:21 18-11-2022

Project Tracker, Velocity & Burndown Chart: (4 Marks)

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 :

$$AV = \text{story points} / velocity\ sprint\ duration$$
$$= 206$$
$$= 3.3$$

Milestone And Activities

Milestones and Activities:

Milestones	Activities
Registration	• Retailer Registration
Login	• Retailer Login
Item Actions	• Add item type • Display item types
Inventory Actions	• Add stock to inventory • Display stock from inventory
Notification	• Sent notification when threshold of stock reached
Location	• Add store location • Update operations to handle location
Customer	• Able to see stocks • Able to make purchase
Deployment	• Project deployment in IBM Cloud

7. CODING&SOLUTIONING

(Explain the features added in the project along with code)

Feature 1

Complete insights into key products and service drivers. With the help of tables and symbols, marketers can effectively track and analyse factors that have an effect on important bottom lines like profitability. Store Managers can also effectively optimise product mix across channels, lines and brands with the product scorecards available. Some of the different KPIs that managers can avail of from product performance metrics are product sales by region, change in sales and margin per product, ROI per product, top competitor by product category and much more..

Feature 2

The entire organisation can access the same store data simultaneously and thus everyone has an understanding of what the customer wants. Managers can better monitor progress, respond immediately to customer needs, adjust parameters for continuous improvement, and exercise greater control over the organisation.

One can record and analyze inventory results and merchandise processes daily to know whether business decisions are based on timely, accurate information.

Code

```
from flask import Flask, render_template, url_for, request, redirect, session, make_response
import sqlite3 as sql
from functools import wraps
import re
import ibm_db
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;SSLServerCertif
icate=DigiCertGlobalRootCA.crt;UID=gkx49901;PWD=kvWCsySl7vApfsy2", '', '')

app = Flask(__name__)
app.secret_key = 'jackiechan'


def rewrite(url):
    view_func, view_args = app.create_url_adapter(request).match(url)
    return app.view_functions[view_func](**view_args)


def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if "id" not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function


@app.route('/')
def root():
    return render_template('login.html')


@app.route('/user/<id>')
@login_required
def user_info(id):
    with sql.connect('inventorymanagement.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM users WHERE email="{id}"')
        user = cur.fetchall()
    return render_template("user_info.html", user=user[0])


@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
```

```
    msg = ''

    if request.method == 'POST':
        un = request.form['username']
        pd = request.form['password_1']
        print(un, pd)
        sql = "SELECT * FROM users WHERE email =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, un)
        ibm_db.bind_param(stmt, 2, pd)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['loggedin'] = True
            session['id'] = account['EMAIL']
            userid = account['EMAIL']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

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


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

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

        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            mg = 'Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms = 'name must contain only character and number'
        else:
            insert_sql = 'INSERT INTO users (USERNAME,FIRSTNAME,LASTNAME,EMAIL,PASSWORD) VALUES
(?,?,?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, username)
            ibm_db.bind_param(pstmt, 2, "firstname")
            ibm_db.bind_param(pstmt, 3, "lastname")
```

```
            # ibm_db.bind_param(pstmt,4,"123456789")
            ibm_db.bind_param(pstmt, 4, email)
            ibm_db.bind_param(pstmt, 5, pw)
            print(pstmt)
            ibm_db.execute(pstmt)
            mg = 'You have successfully registered click login!'
            message = Mail(
                from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
                to_emails=email,
                subject='New SignUp',
                html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for
choosing us.</p>')

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

            response = sg.send(message)
            print(response.status_code, response.body)
            return render_template("login.html", meg=mg)

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


@app.route('/dashboard', methods=['POST', 'GET'])
@login_required
def dashBoard():
    sql = "SELECT * FROM stocks"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    stocks = []
    headings = [*dictionary]
    while dictionary != False:
        stocks.append(dictionary)
        # print(f"The ID is : ", dictionary["NAME"])
        # print(f"The name is : ", dictionary["QUANTITY"])
        dictionary = ibm_db.fetch_assoc(stmt)

    return render_template("dashboard.html", headings=headings, data=stocks)


@app.route('/addstocks', methods=['POST'])
@login_required
def addStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            quantity = request.form['quantity']
            price = request.form['price']
            total = int(price) * int(quantity)
            insert_sql = 'INSERT INTO stocks (NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES
(?,?,?,?)'
```

```
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, price)
            ibm_db.bind_param(pstmt, 4, total)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))


@app.route('/updatestocks', methods=['POST'])
@login_required
def UpdateStocks():
    if request.method == "POST":
        try:
            item = request.form['item']
            print("hello")
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE stocks SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
            if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
                insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, item)
                ibm_db.execute(pstmt)
                dictonary = ibm_db.fetch_assoc(pstmt)
                print(dictonary)
                total = dictonary['QUANTITY'] * dictonary['PRICE_PER_QUANTITY']
                insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, total)
                ibm_db.bind_param(pstmt, 2, item)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))


@app.route('/deletestocks', methods=['POST'])
@login_required
```

```python
def deleteStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            insert_sql = 'DELETE FROM stocks WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))


@app.route('/update-user', methods=['POST', 'GET'])
@login_required
def updateUser():
    if request.method == "POST":
        try:
            email = session['id']
            field = request.form['input-field']
            value = request.form['input-value']
            insert_sql = 'UPDATE users SET ' + field + '= ? WHERE EMAIL=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('profile'))


@app.route('/update-password', methods=['POST', 'GET'])
@login_required
def updatePassword():
    if request.method == "POST":
        try:
            email = session['id']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM  users WHERE EMAIL=? AND PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
```

```
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE EMAIL=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
        finally:
            # print(msg)
            return render_template('result.html')


@app.route('/orders', methods=['POST', 'GET'])
@login_required
def orders():
    query = "SELECT * FROM orders"
    stmt = ibm_db.exec_immediate(conn, query)
    dictionary = ibm_db.fetch_assoc(stmt)
    orders = []
    headings = [*dictionary]
    while dictionary != False:
        orders.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template("orders.html", headings=headings, data=orders)


@app.route('/createOrder', methods=['POST'])
@login_required
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']
            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['PRICE_PER_QUANTITY'])
                query = 'INSERT INTO orders (STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES
(?,?,?,?,?)'
                pstmt = ibm_db.prepare(conn, query)
                ibm_db.bind_param(pstmt, 1, stock_id)
                ibm_db.bind_param(pstmt, 2, quantity)
                ibm_db.bind_param(pstmt, 3, date)
                ibm_db.bind_param(pstmt, 4, delivery_date)
```

```
                ibm_db.bind_param(pstmt, 5, price)
                ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))


@app.route('/updateOrder', methods=['POST'])
@login_required
def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + "= ?" + " WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))


@app.route('/cancelOrder', methods=['POST'])
@login_required
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))


@app.route('/suppliers', methods=['POST', 'GET'])
@login_required
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
```

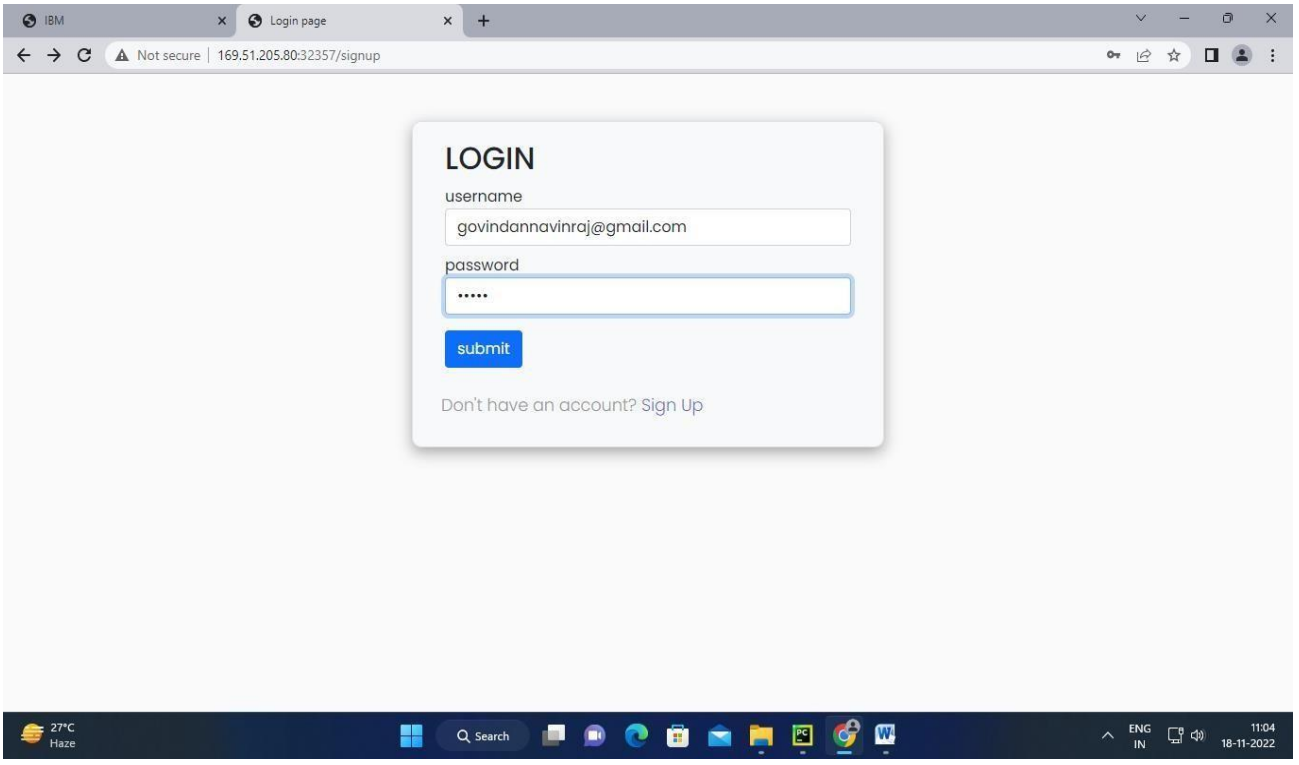
```
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

# get order ids from orders table and identify unassigned order ids
    sql = "SELECT ID FROM orders"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    order_ids = []
    while dictionary != False:
        order_ids.append(dictionary['ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

    unassigned_order_ids = set(order_ids) - set(orders_assigned)
    return render_template("suppliers.html", headings=headings, data=suppliers,
order_ids=unassigned_order_ids)


@app.route('/updatesupplier', methods=['POST'])
@login_required
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/addsupplier', methods=['POST'])
@login_required
def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']
            insert_sql = 'INSERT INTO suppliers (NAME,ORDER_ID,LOCATION) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
```

```
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:
            email = session['id']
            insert_sql = 'SELECT * FROM users WHERE EMAIL=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
        except Exception as e:
            msg = e
        finally:
            # print(msg)
            return render_template("profile.html", data=dictionary)


@app.route('/logout', methods=['GET'])
@login_required
def logout():
    print(request)
    resp = make_response(render_template("login.html"))
    session.clear()
```

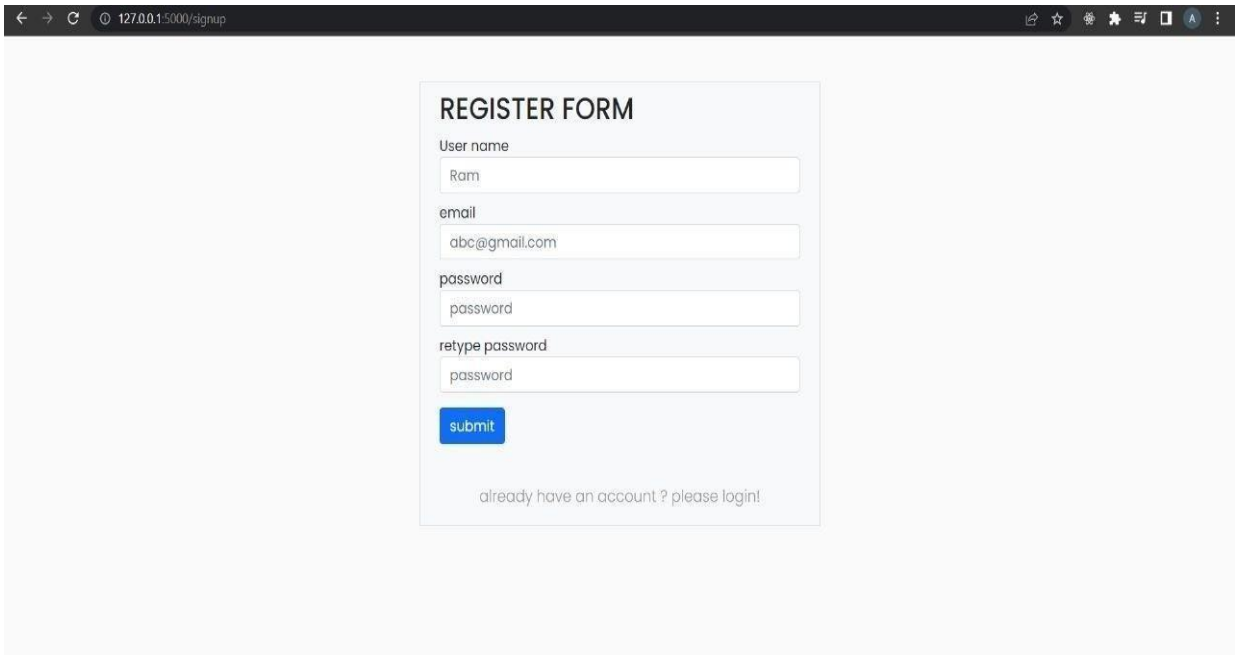
```
    return resp


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

8. RESULTS & OUTPUTS:

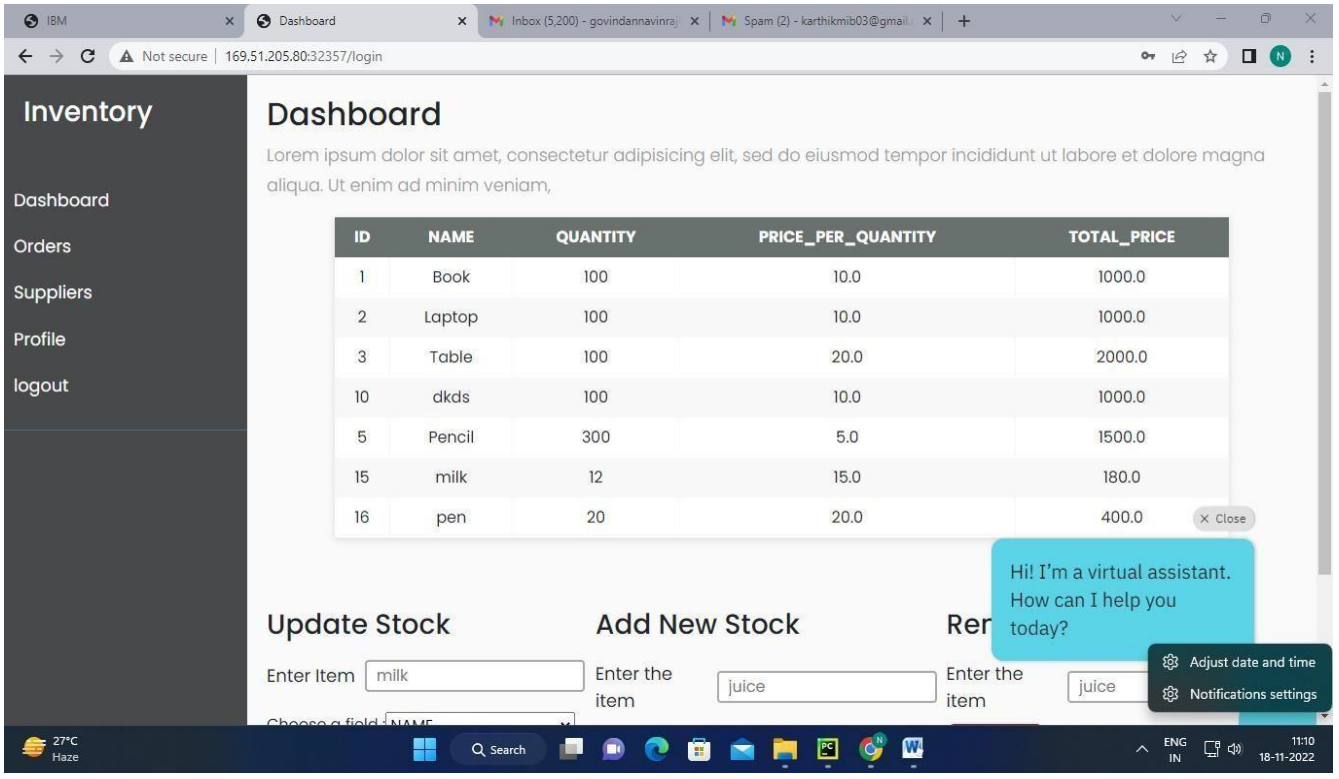
Login Page:



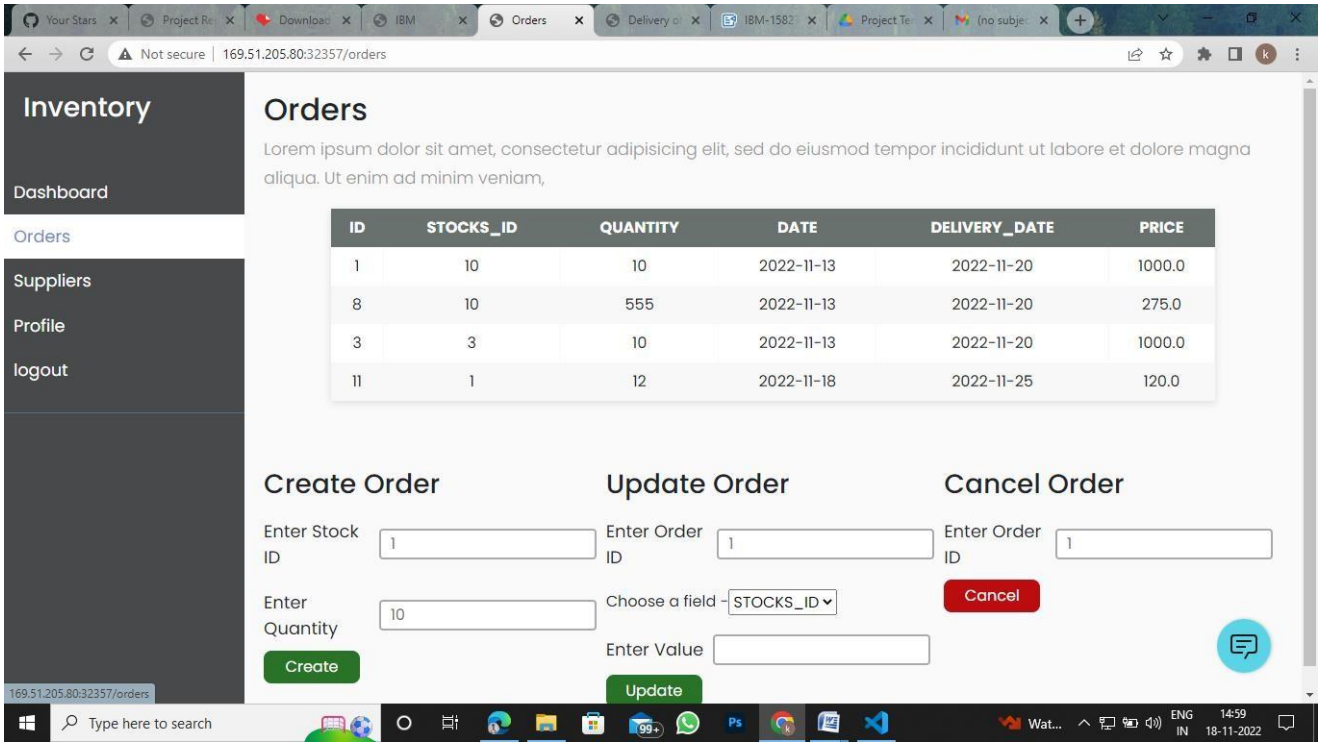
Register Page:



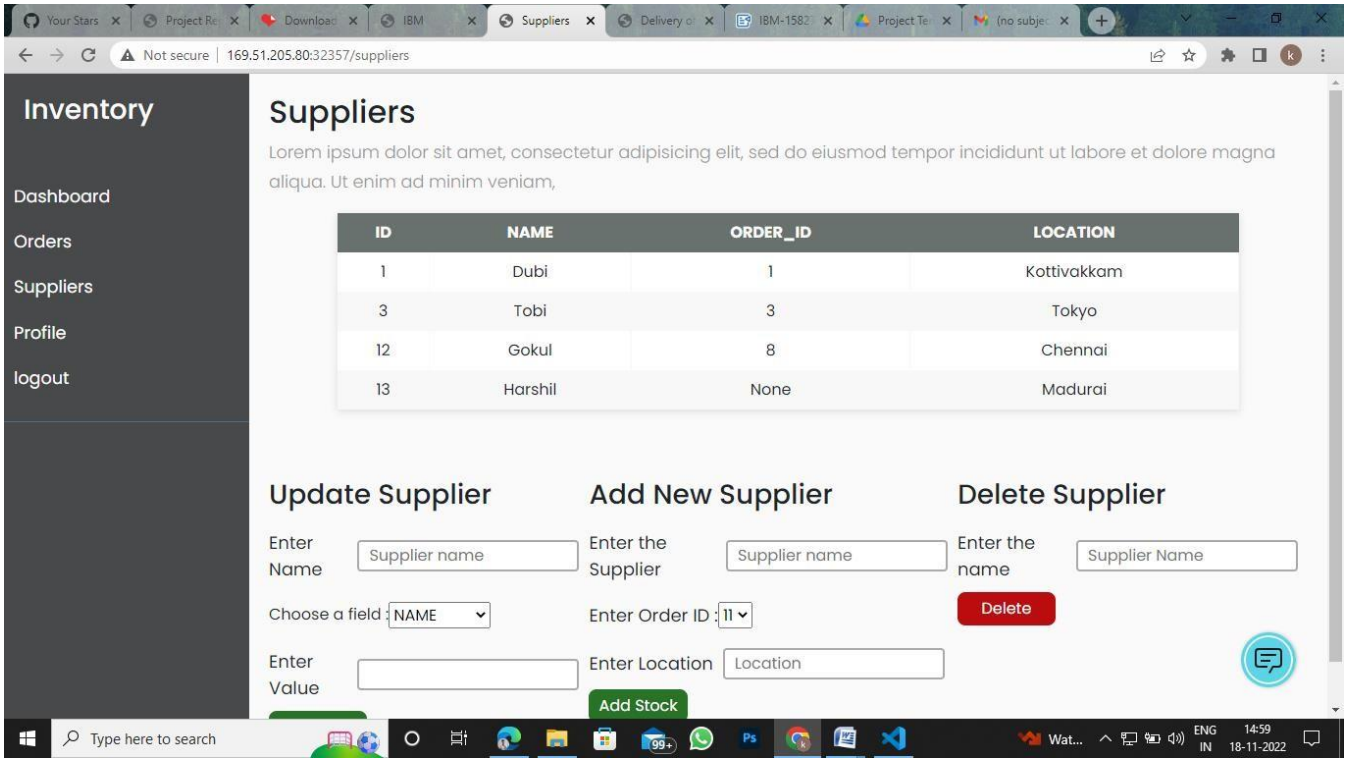
Inventory Page:



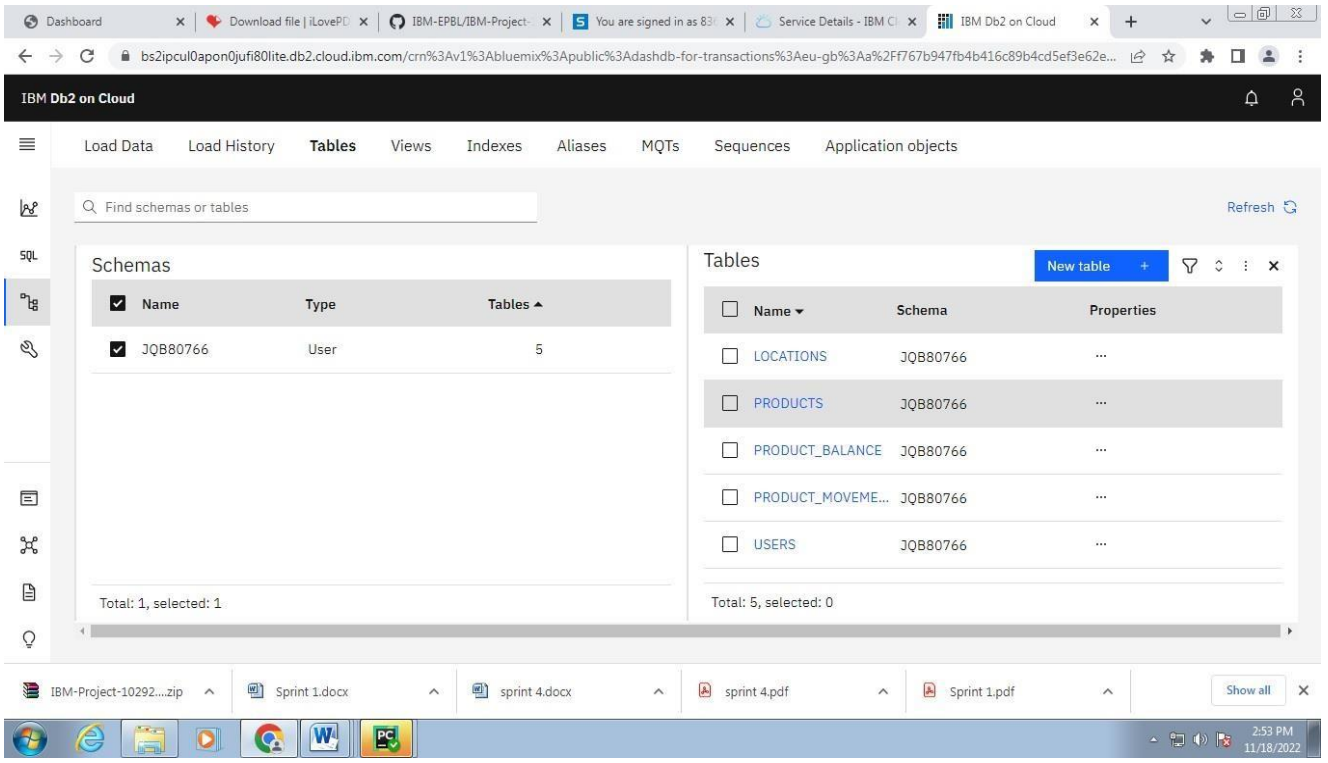
Orders Page:



Suppliers Page:



Cloud Integration:

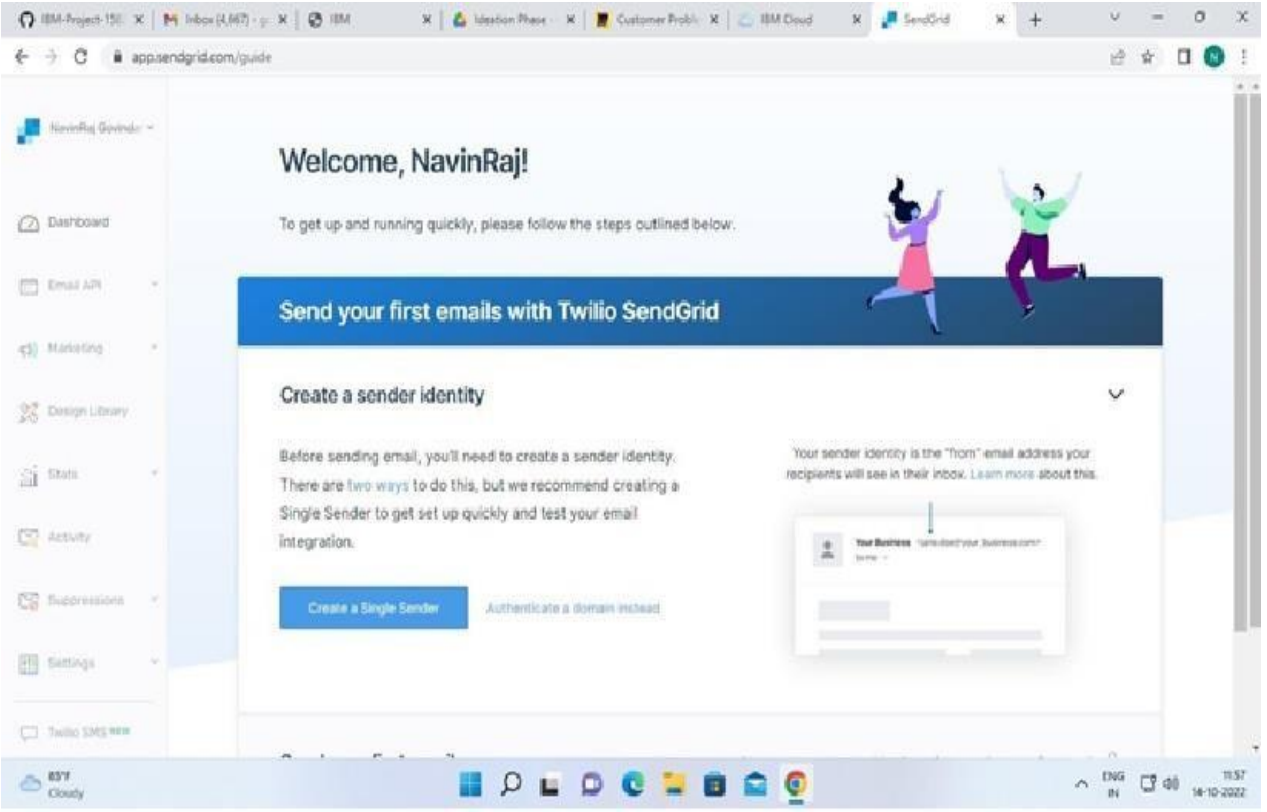


Code for IBM Cloud Connection:

```
from flask import Flask, render_template, url_for, request, redirect, session, make_response
import sqlite3 as sql
from functools import wraps
import re
import ibm_db
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;SSLServerCertif
icate=DigiCertGlobalRootCA.crt;UID=gkx49901;PWD=kvWCsySl7vApfsy2", '', '')
```

Note: DigiCertGlobalRootCA.crt should be downloaded and configured within the project folder.

Sendgrid Integration:

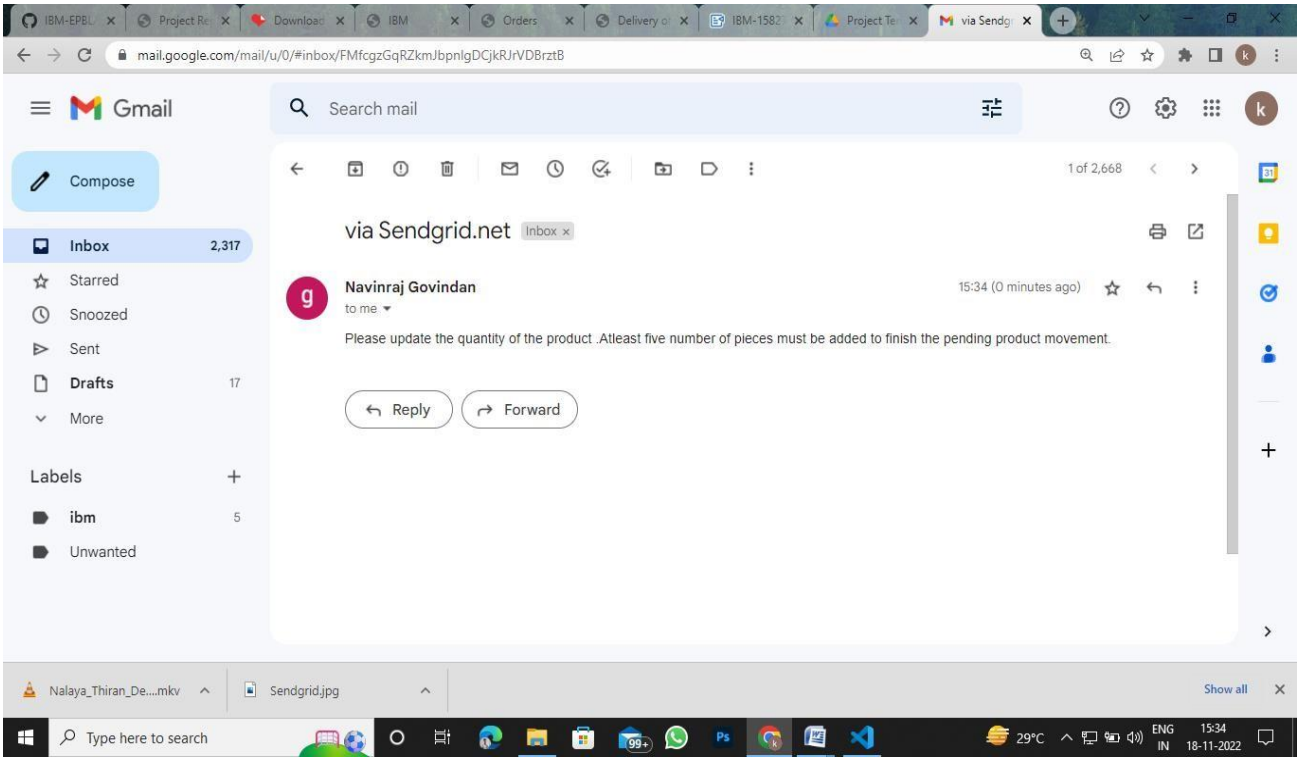


Code for Sendgrid Integration:

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase

def alert(main_msg):
   mail_from = 'dksudha24@pec.ac.in'
   mail_to = 'dksudha25@pec.ac.in'
   msg = MIMEMultipart()
   msg['From'] = mail_from
   msg['To'] = mail_to
   msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
   mail_body = main_msg
   msg.attach(MIMEText(mail_body))

   try:
      server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)
      server.ehlo()
```

```
    server.login('apikey', 'SENDGRID_APIKEY')
    server.sendmail(mail_from, mail_to, msg.as_string())
    server.close()
    print("Mail sent successfully!")
except:
    print("Some Issue, Mail not Sent :(")
```

Email alert on Stock Shortage:



Deploying the Application using Docker and Kubernetes:

```
PowerShell
Loading personal and system profiles took 534ms.
→ flaskapp git:(main) code .
→ flaskapp git:(main) docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/acc
ess-tokens/
→ flaskapp git:(main) docker tag inventory akaash007/inventory:tatest
→ flaskapp git:(main)  docker push akaash007/inventory:tatest
The push refers to repository [docker.io/akaash007/inventory]
40ded361844d: Pushed
4cc577098a3b: Pushed
ffe20d996398: Pushed
371c3d5ecf92: Pushed
bfc1deb8136e: Pushed
1f123186824c: Pushed
3d6eb1152931: Pushed
100796cdf3b1: Pushed
54acb5a6fa0b: Pushed
8d51c618126f: Pushed
9ff6e4d46744: Pushed
a89d1d47b5a1: Pushed
655ed1b7a428: Pushed
tatest: digest: sha256:3a1161b5252df6a4f1f0f41d8623a6ada2bb665900b0c6465102869726c99e31 size: 3053
→ flaskapp git:(main)
```

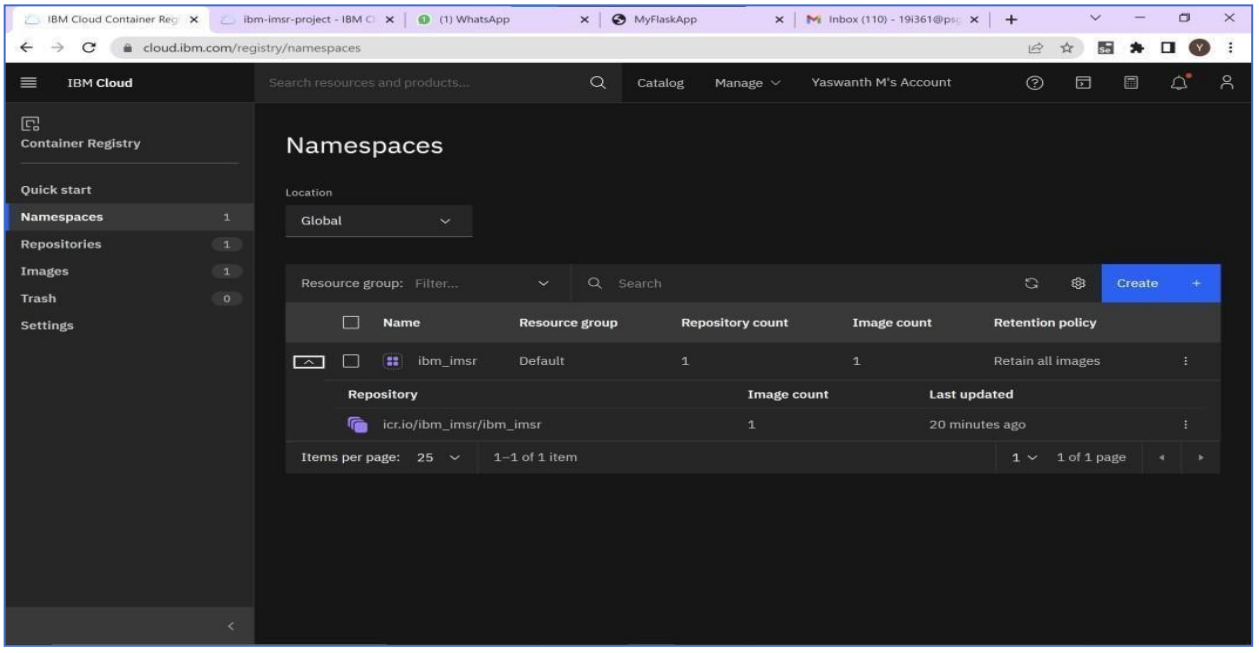
Building an image for our project,

```
  File "/usr/local/lib/python3.11/site-packages/flask/app.py", line 1820, in full_dispatch_request
PS C:\Users\yaswa\Downloads\IBM\IMSR-1> docker build -t yaswanthmanoharan/ibm_imsr .
[+] Building 2.7s (11/11) FINISHED
 => [internal] load build definition from Dockerfile                                                    0.0s
 => => transferring dockerfile: 32B                                                                     0.0s
 => [internal] load .dockerignore                                                                       0.0s
 => => transferring context: 2B                                                                         0.0s
 => [internal] load metadata for docker.io/library/python:latest                                        2.4s
 => [auth] library/python:pull token for registry-1.docker.io                                           0.0s
 => [internal] load build context                                                                       0.0s
 => => transferring context: 24.29kB                                                                    0.0s
 => CACHED [2/5] WORKDIR /inventory                                                                     0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                 0.0s
 => CACHED [4/5] RUN pip install -r requirements.txt                                                    0.0s
 => [5/5] COPY . .                                                                                      0.0s
 => exporting to image                                                                                  0.1s
 => => exporting layers                                                                                 0.0s
 => => writing image sha256:0afb0c793a704eaf85acc886443c57a0cbeca9473b841897ef4a9162f3c4bd06            0.0s
 => => naming to docker.io/yaswanthmanoharan/ibm_imsr                                                   0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\yaswa\Downloads\IBM\IMSR-1> docker run -p 8080:5000 yaswanthmanoharan/ibm_imsr
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serve
r instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [14/Nov/2022 03:57:11] "GET /login HTTP/1.1" 200 -
172.17.0.1 - - [14/Nov/2022 03:57:22] "POST /login HTTP/1.1" 302 -
172.17.0.1 - - [14/Nov/2022 03:57:23] "GET /dashboard HTTP/1.1" 200 -
172.17.0.1 - - [14/Nov/2022 03:57:27] "GET /product_movements HTTP/1.1" 200 -
172.17.0.1 - - [14/Nov/2022 03:57:30] "GET /add_product_movements HTTP/1.1" 200 -
[2022-11-14 03:57:37,822] ERROR in app: Exception on /add_product_movements [POST]
Traceback (most recent call last):
```

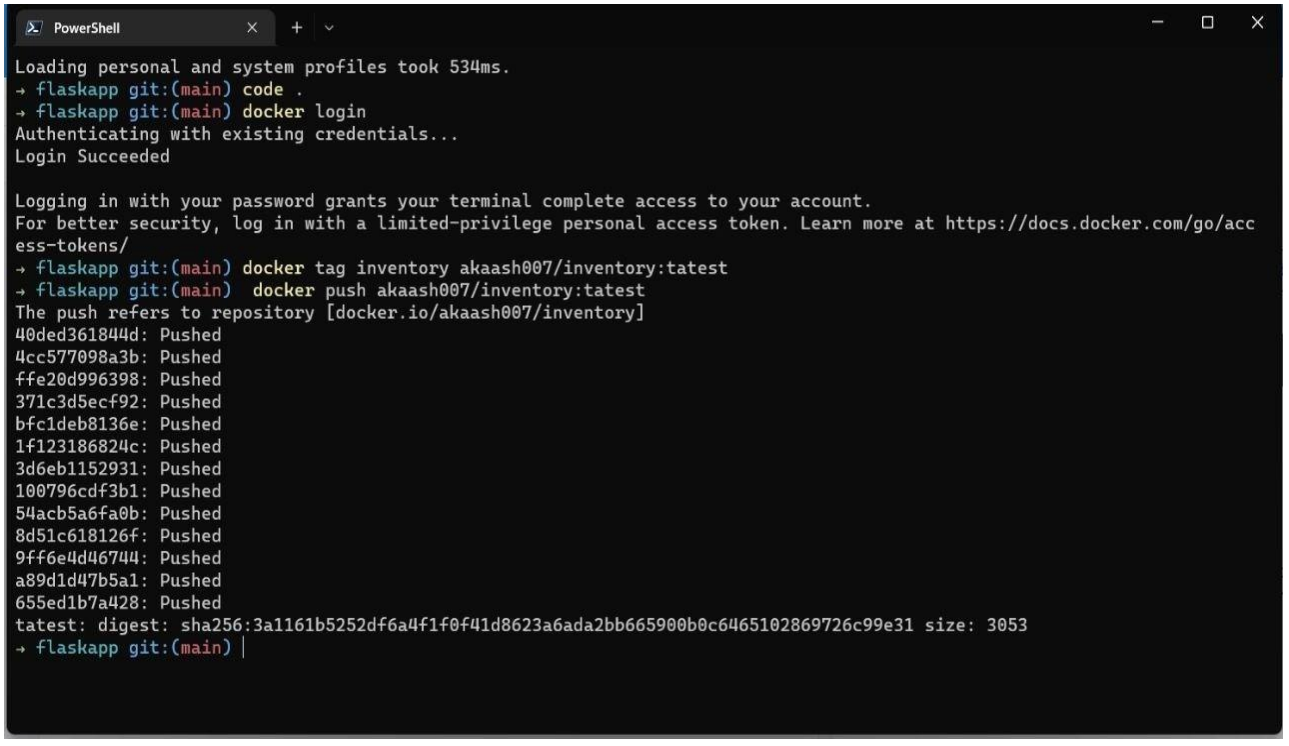
Create a valid deployment.yaml file,

```
PS C:\Users\yaswa\Downloads\IBM\IMSR-1> kubectl apply -f deployment.yaml
deployment.apps/ibmimsr created
PS C:\Users\yaswa\Downloads\IBM\IMSR-1>
```

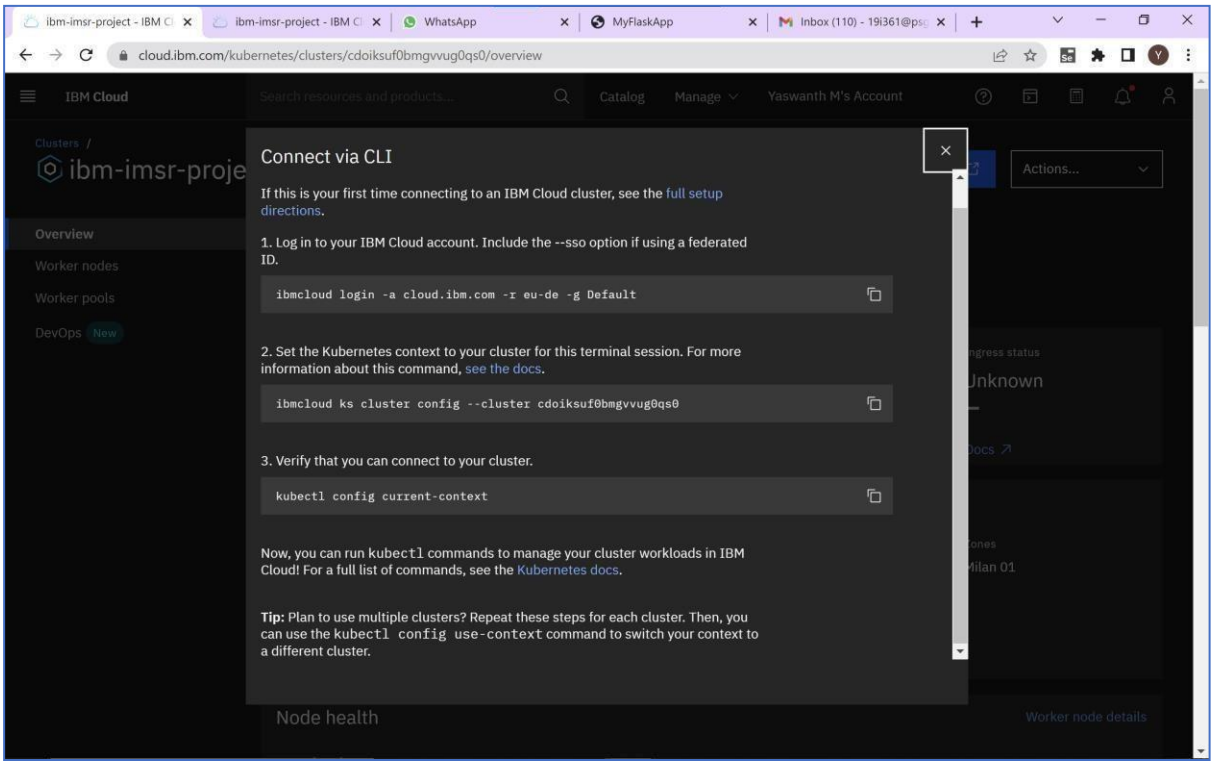
Create a namespace in IBM Container registry,



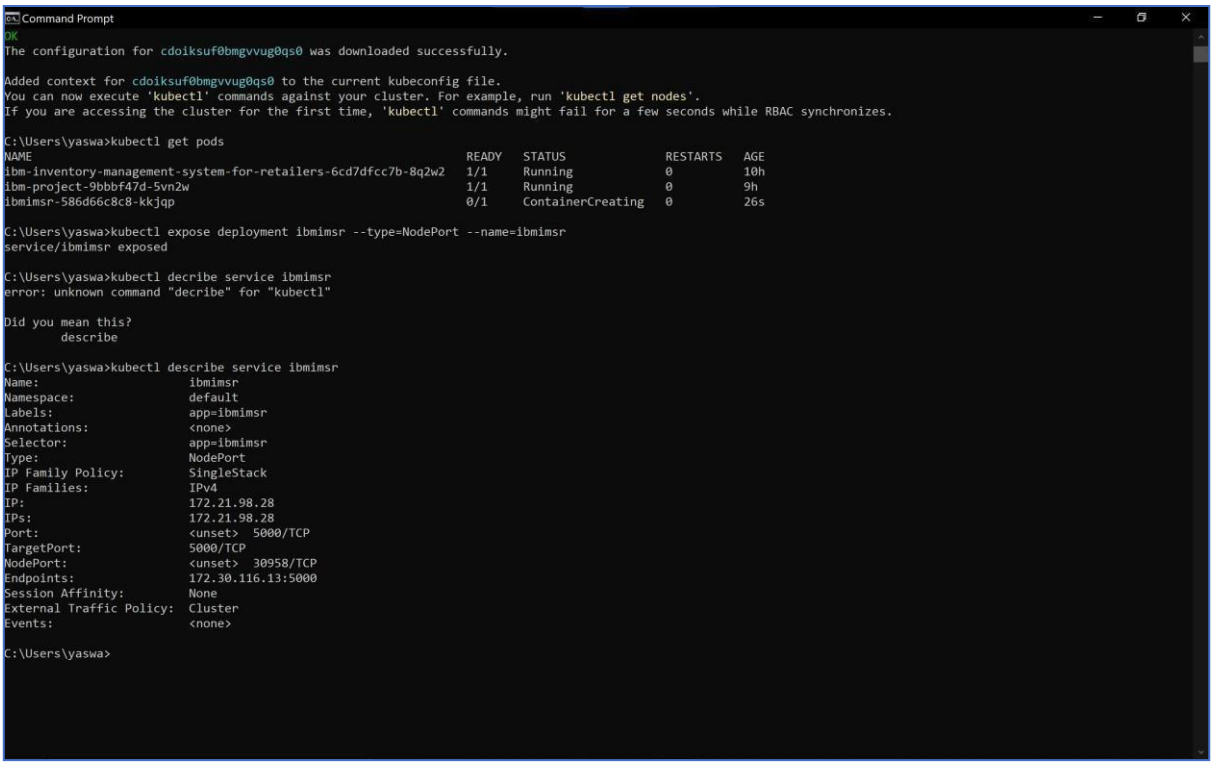
Pushing the project into IBM container Registry,



Note: Create a Kubernetes Cluster in IBM Cloud and wait for the work node to get fully deployed.Then, Login into Kubernetes Cluster using the following commands,



Expose your application using the following command and check for the port number using the nextcommand.



Then, Check for the public IP address in your IBM Kubernetes Cluster under Worker Node,

Thus we have the Public IP address and the Nodeport.Now

just type in this format - <Public_IP>:<NodePort>

For our Inventory management system application it is, **169.51.205.80:32357**

Type this in the browser and click enter to access the deployed application

Performance Metrics

Inventory Performance is a measure of how effectively and efficiently inventory is used and replenished. The goal of inventory performance metrics is to compare actual on-hand dollars versus forecasted cost of goods sold. Many Leanpractitioners claim that inventory performance is the single best indicator of the overall operational performance of a facility.

9. ADVANTAGES & DISADVANTAGES

- Paper-based retail inventory management can take a lot of time and effort. The retail inventory management software can cut short your in-store inventory process cycles through automation. Automation would give you time to focus on other productive business tasks.
- Inventory management is one of the crucial retail processes. Thus, any discrepancy in the inventory control would impact all other operations in your company. The retail inventory software can streamline the inventory processes, which would, in turn, improve the efficiency of your entire business
- Manual inventory control would increase your labor and process costs. The software would not only help you save time, but it would also help you reduce costs. As a result, the profitability of your business would improve. Also, you can invest the excess funds in activities that promote your business growth.
- One of the biggest problems with any computerized system is the potential for a system crash. A corrupt hard drive, power outages and other technical issues can result in the loss of needed data. At the least, businesses are interrupted when they are unable to access data they need. Business owners should back up data regularly to protect against data loss.
- Hackers look for any way to get company or consumer information. An inventory system connected to point-of-sale devices and accounting is a valuable resource to hack into in search of potential financial information or personal details of owners, vendors or clients. Updating firewalls and anti-virus software can mitigate this potential issue.
- When everything is automated, it is easy to forego time-consuming physical inventory audits. They may no longer seem necessary when the computers are doing their work. However, it is important to continue to do regular audits to identify loss such as spoilage or breakage. Audits also help business owners identify potential internal theft and manipulation of the computerized inventory system.

10. 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. As you have read above, there are different techniques that businesses can utilize to simplify and optimize stock management processes and control systems.

11. FUTURE SCOPE

In summary, successful companies will embrace the challenges of inventory management in the 21st century by levering the technology that is being offered through the Fourth Industrial Revolution. More important, companies will look at inventory as a strategic asset, that when properly deployed will deliver increased value and competitive advantage. Effective collaboration between supply chain partners will take on increased importance. The intensifying risks inherent with global sourcing in combination with a better appreciation of TCO will motivate companies to rethink their global inventory strategies.

12. REFERENCES

- Aggarwal, S.: A review of current inventory theory and its applications. International Journalof Production Research 12, 443–472 (1974)
- Anily, S., Federgruen, A.: One warehouse multiple retailer systems with vehicle routing costs. Management Science 36, 92–114 (1990)
- Beckmann, M.: An inventory model for arbitrary interval and quantity distributions of demand. Management Science 8, 35–57 (1961)
- Hamann, T., Proth, J.: Inventory control of repairable tools with incomplete information. International Journal of Production Economics 31, 543–550 (1993)

13. APPENDIX

GitHub link

https://github.com/IBM-EPBL/IBM-Project-15827-1659605000.git

Demo Video Link:

https://drive.google.com/file/d/1rdxrw-gZwrEBBnnlkQapNRDwDeJwQyL2/view?usp=share_link