RMK ENGINEERING COLLEGE

(An Autonomous Institution)

R.S.M. Nagar, Kavaraipettai-601 206, Gummidipoondi Taluk, Thiruvallur District.

PROJECT

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

DONE BY

TEAM ID – PNT2022TMID15822

RAMYA R - 111719104131

RAMYA M - 111719104130

RITHIKA L - 111719104133

S.SRI RISHITHA REDDY - 111719104135

ABSTRACT

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply. In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROJECT OVERVIEW

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

1.2 PURPOSE

- Helps companies identify which and how much stock to order at what time.
- Better level of customer service.
- Maintaining sufficient stocks.
- Optimizing product sales.

2. LITERATURE SURVEY

2.1Existing Problem

The Inventory Management System refers to the system and processes to manage an organization's stock with the involvement of Technology. This system can be used to store the Inventory details, stock maintenance, update the inventory based on the sales details, and generate sales and inventory reports daily or weekly. Inventory management is a challenging problem area in supply chain management. Companies need to have inventories in warehouses to fulfil customer demand, meanwhile, these inventories have holding costs and this is the frozen fund that can be lost. This system is categorized as individual sales and inventory management aspects. There are different problems faced by retailers in maintaining their stocks. An inventory Management System is important to ensure quality control in businesses that handle transactions revolving around the goods. Without, Inventory Management even larger scale businesses cannot find the products which they have in very small quantity or not having that particular product or stock. The inventory System plays a very important role in tracking large-scale business. An automated inventory system helps retailers to minimize errors in recording the goods.

2.2 Reference

Author Name: Gaur, Fisher and Raman
Year of Publishing: 2005
Description:

Their study examined firm-level inventory behaviour among retailing companies. They took a sample of 311 public-listed retail firms for the years 1987–2000 for investigating the relationship between stock turnover gross margin, capital intensity, and sales surprise. All observed that stock aggregate turnover for retailing companies was positively related to the capital intensity with sales surprise while inversely related to gross margins.

Author Name: Pradeep Singh
Year of Publishing: 2008
Description:
In his study attempted to investigate stock with working capital managing Indian Farmers Fertilizer Cooperative Limited (IFFCO) / National Fertilizer Limited (NFL). He concluded that the overall position of the working fund of IFFCO / NFL is satisfactory. But there arises the need for improvement in stocking as the situation of IFFCO. Although the stock was not properly utilized as well as maintained bay IFFCO during the investigation period. Also managing the organization of the NFL surgeries tries to properly utilize stock with try to care for the stock according to requirements. So that liquidity will not interrupt.

Author Name: Capkun, Hameri & Wesis
Year of Publishing: 2009
Description:
Statistically analyzed the association among stock levels with fund situation in manufacturing companies using capital information on the large sample of US-based production units over a 26-year period, from, 1980 to 2005. According to them, a significant relationship existed between inventory performance along with the performance of its components and profitability.

Author Name: Sahari, Tinggi & Kadri
Year of Publishing: 2012
Description:
They focused on the association between the inventory management system and company performance corresponding to fund capability. Therefore according to that reason, they looked at 82 sample construction companies in Malaysia during the period of 2006– 2010. Using the regression and correlation analysis methods, they deduced that inventory management is positively correlated with firm performance. In addition, the results indicate that there is a positive link between inventory management and capital intensity.

Author Name: Panigrahi
Year of Publishing: 2013
Description:
According to his analysis inventory management practices used by Indian cement firms and their effects must be on working fund efficiency. The study also investigated the relationship between profitability and inventory conversion days. The study, using a sample of the top five cement companies of India over a period of 10 years from 2001 to 2010, concluded there must exist an inverse relationship between the conversion period of inventory and profit margin.

Author Name: Edwin Sitienei & Florence Memba
Year of Publishing: 2015
Description:
Conducted a study on the Effect of Inventory Management on the profitability of Cement Manufacturing Companies in Kenya. The study concluded that Gross profit margin is negatively correlated with the inventory conversion period, an ieee in sales, which denotes the

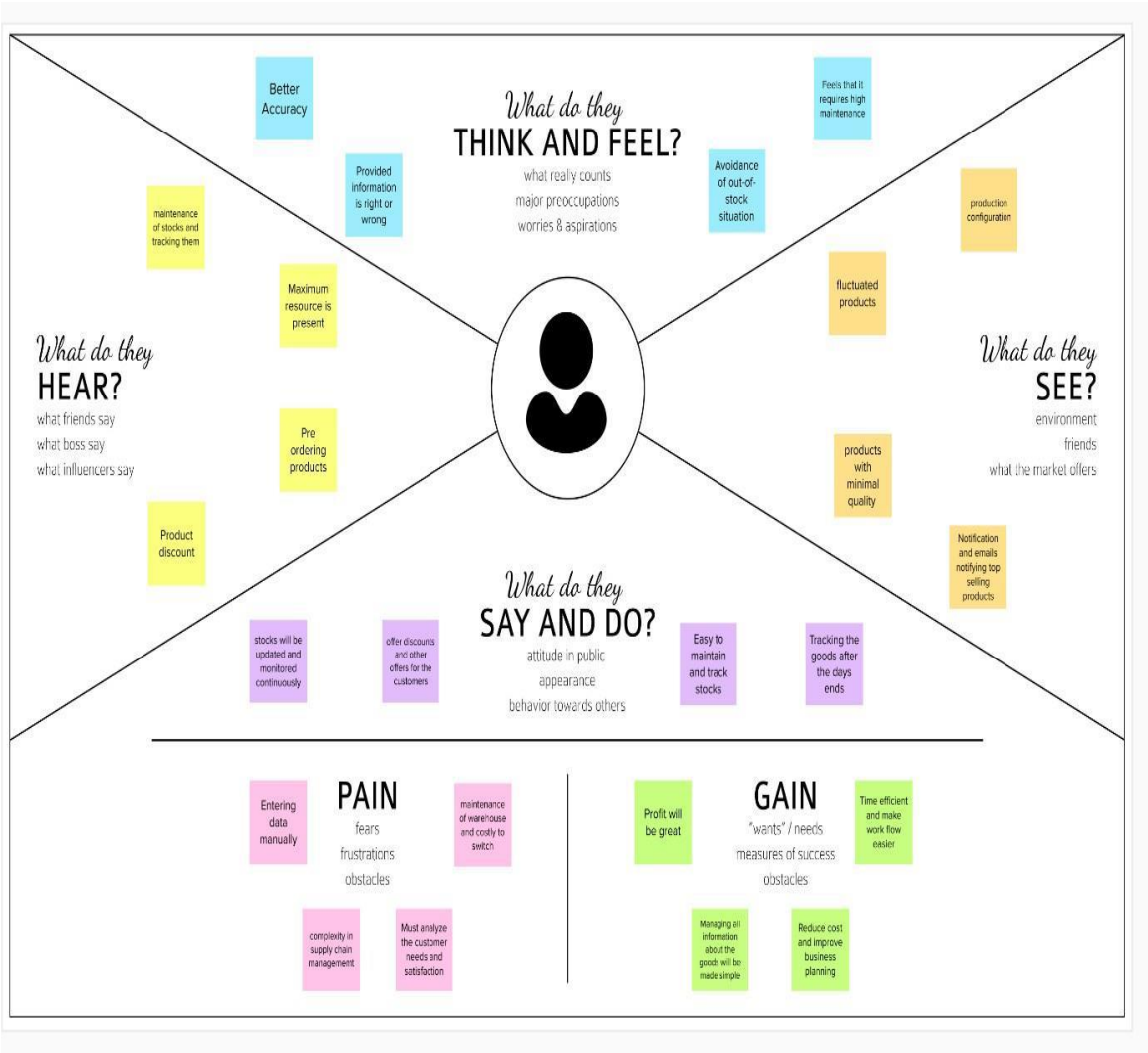
firm size enriches the firm's inventory levels, which pushes profits upwards due to optimal inventory levels. It is also noted that firms' inventory systems must maintain appropriate inventory levels to enhance profitability and reduce the inventory costs associated with holding excessive stock in warehouses.

2.3 Problem Statement Definition

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Retailer	Find if a particular product is present or not	I did not find the information about the product	There are no entries on the product which I'm searching	Annoyed and irritated
PS-2	Retailer	Maintain transportation payslip for the delivered goods	It's hard to track the goods and cost	The process takes a large amount of time	Uninterested
PS-3	Retailer	Collect the customer reviews and feedback	It is difficult to collect	I do not have the contacts of the customer to get the reviews	Disappointed
PS-4	Retailer	Find the high demand	It's difficult to calculate	It takes more time	Challenging
PS-5	Retailer	Maintain a ledger	It is difficult to secure	The information can be lost or stolen	Terrified

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

1

Define your problem statement

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

5 minutes

PROBLEM

Inventory management system for both customers and retailers to keep track of their goods (like running out of stocks are excess supply) which is beneficial for both customer and retailers.

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

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

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Team Lead - Ramya R

- Creates an application setup and database connection
- Maintains the cloud storage
- Automate the inventory system for retailers
- Get on control on the inventory system
- Create a chat bot on the platform and keep track of stock and goods availability
- Triggering the alert messages when the socks fall down

Ramya M

- Customer Feedback and rating system including both the product and the retail shop service
- Create a safe environment for workers and customers
- Keeping a Track of the expiry dates of all the stock and announcing the discounts and offer for those products which is going to expire soon
- update product and details regularly
- Sends mail and notification regarding the stocks present
- Keep a record of regular customers and send them regular notice about the arrivals and exclusive offers and discounts for them.

Rithika L

- Calculate the current inventory stock by subracting the product which were sold today.
- Provide special discount for the first purchase and can add key points with further purchase so future special discounts
- Can make use of excel sheet for processing the data.
- Exchange and return features should be maintained
- Easy and fast billing system with also provides option for the customers either through cash or through net banking.
- Bring RFID based product tracking system into the existence

Rishitha Reddy S

- Scheduling all the product deliveries properly for maximum utilization of transportation.
- Adding the retailers contacts on the respective products
- Customer details should be maintained very securely
- Customer Feedback and rating system including both the product and the retail shop service.
- Enhancing customer loyalty and providing transparency in the billing.
- Make sure to have free door deliveries to the nearest areas and to avoid late deliveries.

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

Features

- E-mails and SMS alerts to the customers regarding the discounts and new arrivals
- Transparency in the billing.
- Ensuring the availability of all the products atleast in threshold amount all time
- Showcasing the customer feedback to the public regarding both the product and the store

Services

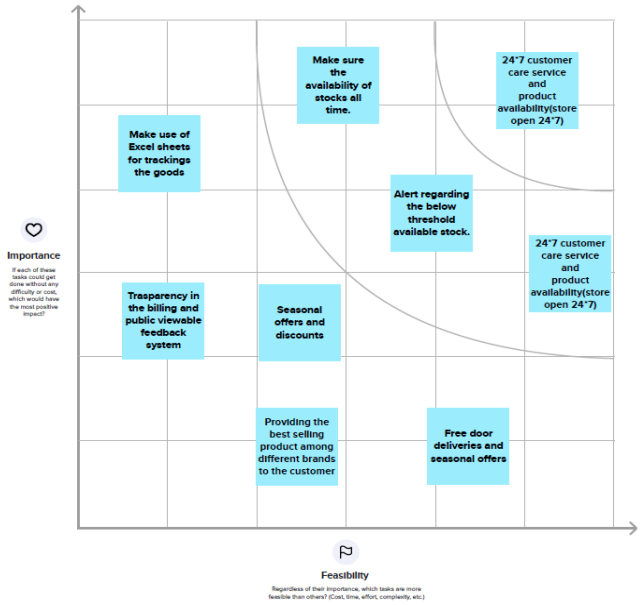
- Free door deliveries and online purchases
- Special sales on more purchased goods for regular customers
- 24*7 customer care service
- Online Ecommerce service for elderly and working people

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



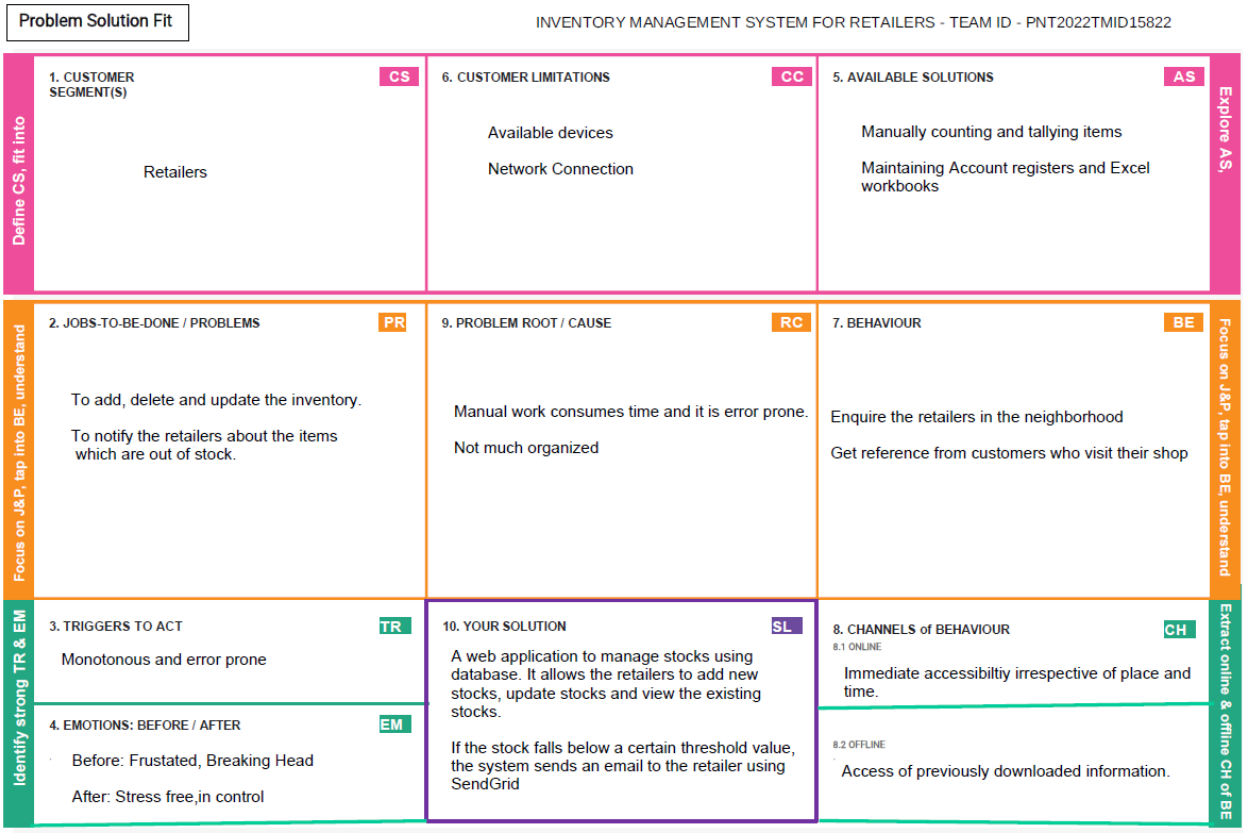
3.3Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• The main focus of this project is the day-to-day problems facing by the retailers and how to reduce the risk and misuse of information. • Retailers do not know all the information about the product they own. Some may be expired or out of quantity. The reviews and contact of customer for further use cannot be maintained. • So, in order to overcome all these problems, we are implementing the inventory system for the retailers with smooth usage and better in terms of confidentiality.
2.	Idea / Solution description	• The proposed system will have information about all the products and whether the product is outdated or can be recycled. • This system has an alert triggered to know about the expired product and soon to be expiring product. • The information about the customer is collected and stored in cloud storage safely. • All the information which is confidential were stored and configured in cloud in very secured way. • Customers can register and get login credential for further use, they can track the goods they ordered. • Regular update of goods and stocks are maintained and updated time to time in the website for customer use.

		• Customer satisfactory is much needed. • The application allows the customers to know all the present time available stocks and also when the new stock will be available on the store for them to buy.
3.	Novelty / Uniqueness	• Notifications will be sent to the retailers if any product that the customers have been looking for is not available so that the product can be stocked up soon. • Notification will be sent to the customers who buys any certain products regularly when the new arrivals are stocked up. • Exclusive discounts and offers are given for regular customers to keep them engaged with the store regularly.
4.	Social Impact / Customer Satisfaction	• The customer will be dissatisfied when the product they trying to find is not available. • Customer reviews are more important in order to improve the process. • The workload of the retailers will be lessened if the process is automated every day during every purchase. • The customer will be satisfied if the appropriate response is received from the retailers.
5.	Business Model (Revenue Model)	The most reliable inventory statement can be implemented by • Deploy the application in the cloud application and make use of databases. • Maintaining the privacy of the users. • Establishing a loss Prevention strategy. • To ensure all the products that is the customer is expecting a available or available in the nearby surrounding. • Usage of freebies business strategy for gaining the customer attention.

6.	Scalability of the Solution	• The system is very efficient in large amount of storage. That is, we can store large amount of information about the products because we are deploying the application in cloud. • Authority for adding the products and maintaining the product in closed terms are done by specific members who are assigned to do the task. • Updating the stocks for each purchase will help in the prevention for inventory shrinkage. • Chat bots will play a major help in communicating between the customer and retailers for providing the best customer service.

3.4 Problem Solution Fit



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

Functional Requirements:

Following are the functional requirements of the proposed solution.

	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	• Registration through Form • Registration through Gmail
FR-2	User Confirmation	• Confirmation via Email • Confirmation via OTP • Authentication via Google Authentication
FR-3	Data Management	• Add the products and enter the details about the products. • Quickly produce reports for single or multiple products. • Track information of dead and fast-moving products. • Track information of suppliers and manufacturers of the product.
FR-4	History of Data	• Keeps track of the products in the warehouse and always updates the value in the cloud database. • If a shortage of any product is found then the retailers will receive an E-mail about the shortage of product.
FR-5	Audit Monitoring	• The technique of tracking crucial data is known as audit tracking. • Monitor the financial expenses carried out throughout the whole time (from

		receiving order of the product to delivery of the product).
FR-6	CRM (Customer Relationship Management)	• Track the customer experience via ratings given by them. • Get customer reviews regularly or at-least at the time of product delivery to work on customer satisfaction. • User-friendly GUI to increase the customer base from only techies to normal people. • Special offers for regular customers have to be provided through credits in the web-app itself.
FR-7	Warehouse Location and Return, Replacement Policies	• Warehouse are prepared according to the preferred locations. • Return and Replacements policies plays a major role in customer satisfaction. • Return the product which is received wrongly or defective is made easy. • Replacement of the defective product is made easier.
FR-8	Security Policies	• User data collected must be as secure as possible. User data must not be misused. • They can only be used for user preferred advertising purposes.

4.2 NON-FUNCTIONAL REQUIREMENT

Non-functional Requirements:

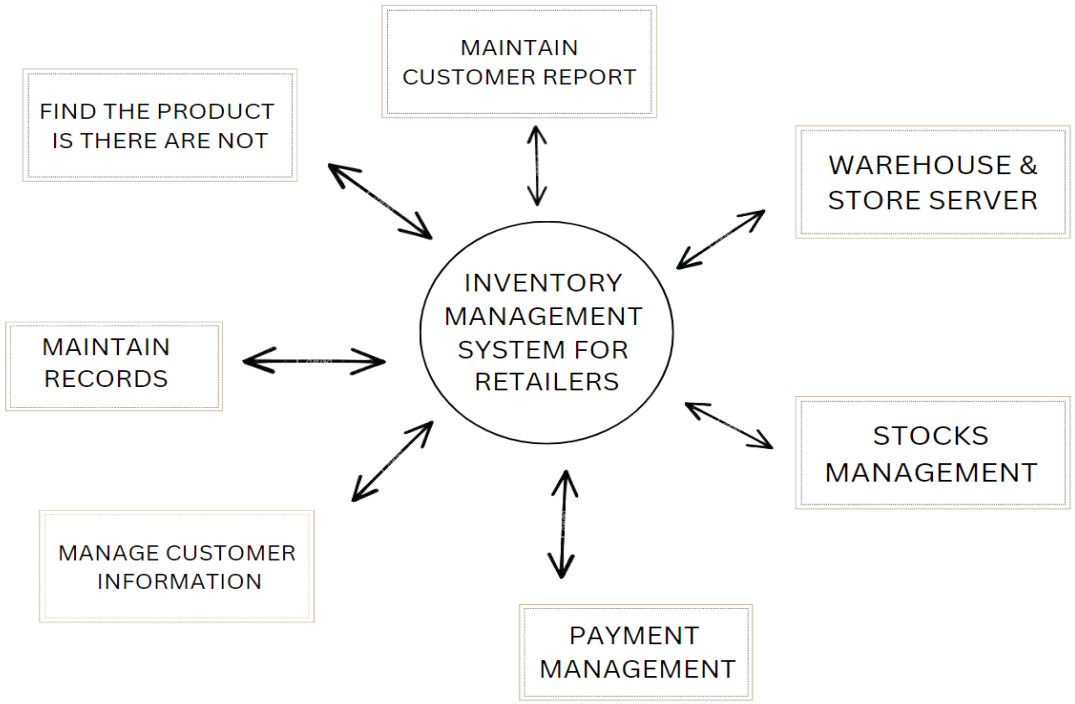
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	• The UI should be accessible to everybody despite of their diversity in languages. • People with some impairments should also be able to use the application with ease. (Example, integrate google assistant so that blind people can use it).

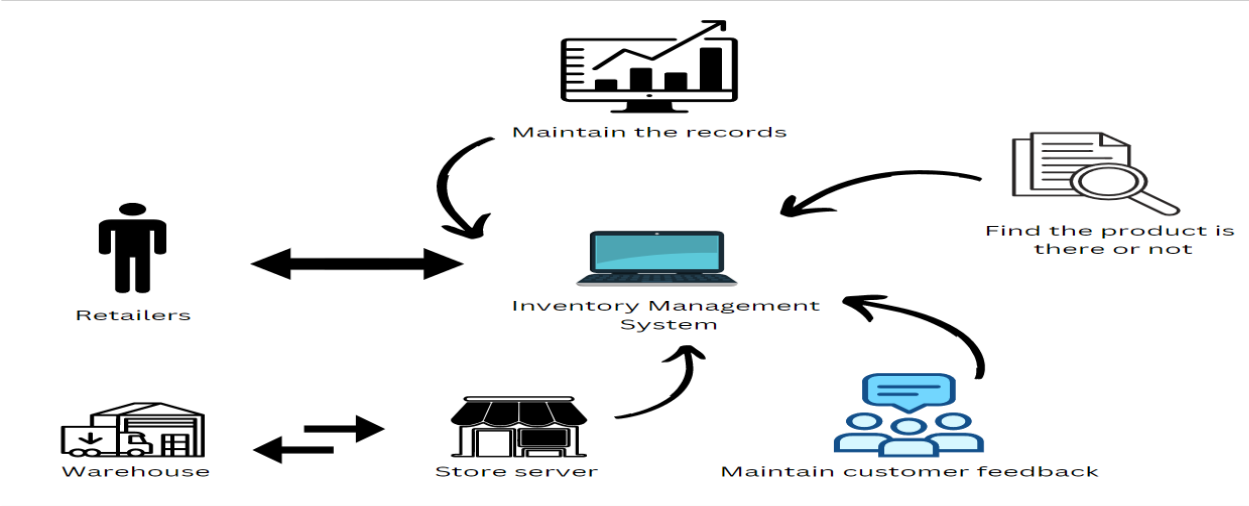
		• The app and UI should be platform and device independent. • It should be compatible with wide range of devices possible.
NFR-2	**Security**	• The security requirements deal with the primary security. • Only authorized users can access the system with their credentials. • Administrator or the concerned security team should be alerted on any unauthorized access or data breaches so as to rectify it immediately.
NFR-3	**Reliability**	• The software should be able to connect to the database in the event of the server being down due to a hardware or software failure. • The recovery of the application should be immediate such that the downtime of the application should be negligible. • The users must me intimated by the periodic maintenance break of the server so that they will be aware of it.
NFR-4	**Performance**	• Performance of the app should be reliable with high-end servers on which the software is running.
NFR-5	**Availability**	• The software should be available to the users 24/7 with all functionalities working. • New module deployment should not impact the availability of existing modules and their functionalities.
NFR-6	**Scalability**	• The whole software deployed must be easily scalable as the customer base increases.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2Solution & Technical Architecture



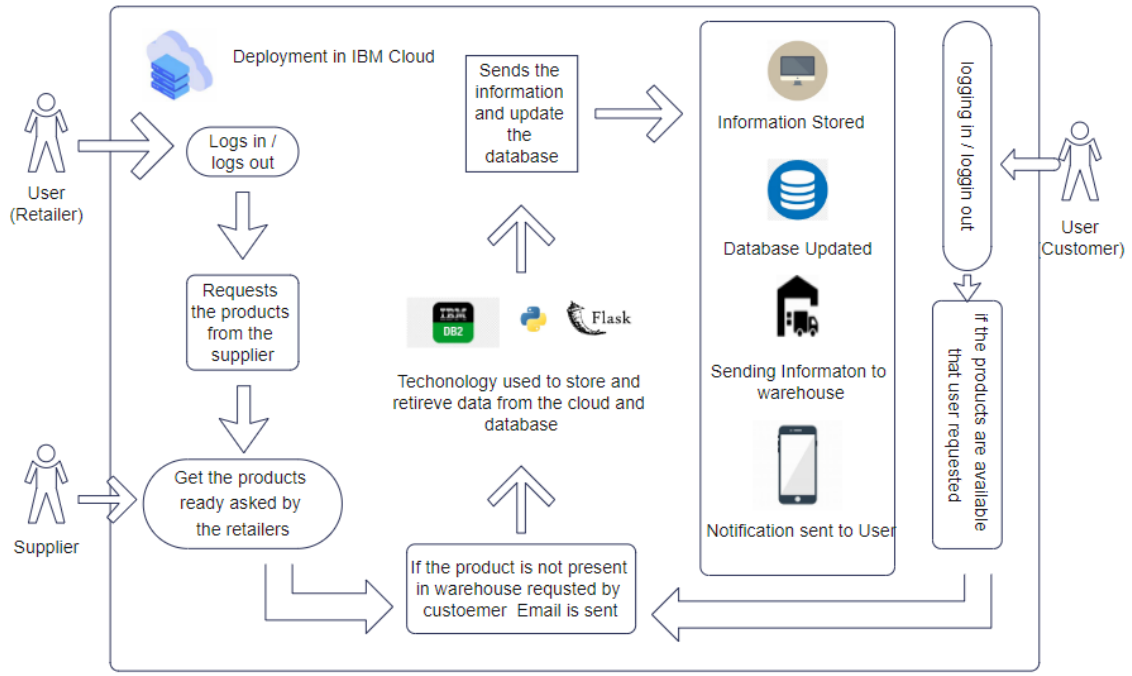


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Technology used

5.3 User Stories

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

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user & Web user)	Registration	USN - 1	As a user, I can register for the application by entering my email, password & confirming my password.	I can access my account / dashboard.	High	Sprint - 1
		USN - 2	As a user, I can see if there is any underflow or overflow. If there is underflow condition, I can order products, otherwise I can eliminate the stock which is not in large usage.	I can see the conditions of stock.	Low	Sprint – 2
		USN - 3	As a user, I can manage sales information and manage transactions/payments.	I can manage sales information and manage transactions/payments.	Medium	Sprint – 2
	Login	USN - 4	As a user, I can login to product database and see the information about product.	I can login to product database and see the information.	High	Sprint – 1
	Dashboard	USN - 5	As a user, I can create a database credentials and enter my stocks.	I can create a database credentials and enter my stocks.	High	Sprint – 4
Customer Care Executive		USN – 6	As a customer care executive, I can solve the log in issues and other issues of the application.	I can provide support or solution at any time 24*7	High	Sprint – 3
Administrator	Application	USN - 7	As a user, I can create alerts to require stocks.	I can manage the alerts on the stocks.	High	Sprint - 4

6. PROJECT PLANNING & SCHEDULING

6.1Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

RR - Ramya. R

RM – Ramya. M

RL – Rithika L

SR – Sri Rishitha Reddy

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	RR, RM
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	3	High	RR, RM
Sprint-1		USN-3	As a user, I can register for the application through Gmail	3	Medium	RR, RM
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	5	High	RR, RM
Sprint-1	Dashboard	USN-5	As a user, I must able to see my details on the dashboard	4	High	RR, RM
Sprint-2		USN-6	As a user, I can make changes on my personal information (updating and deleting information) and able to change my password	3	Medium	RL, SR
Sprint-2	Inventory	USN-7	As a retailer, I should be able to add the products in the app	5	High	RL, SR
Sprint-2		USN-8	As a retailer, I can add or remove products which are required and not	2	Medium	RL, SR
Sprint-2		USN-9	As a retailer, I must get a warning if any product in near of out-of-stock	7	High	RL, SR
Sprint-2		USN-10	As a user, I can view the products available in the inventory	3	Medium	RL, SR
Sprint-3	Orders	USN-11	As a user, I should able to order the products available	6	Medium	RR, SR
Sprint-3		USN-12	As a user, I can request if the product I am looking for is not available	4	Low	RR, SR
Sprint-3		USN-13	As a user, I can place an order through the app and complete the payment in a very secured way	7	High	RR, SR
Sprint-3		USN-14	As a user, I should get the product on time and safe delivery is expected	3	Medium	RR, SR
Sprint-4	Maintenance (account & Warehouse)	USN-15	As a retailer, I can add new location where are the products can be present	4	Medium	RM, RL
Sprint-4			As a retailer, I can monitor the products and their quantity in warehouses	7	High	RM, RL
Sprint-4		USN-16	As an administrator, I can remove the user account temporarily or permanent if the user wishes	5	Low	RM, RL
Sprint-4	Feedback	USN-17	As a customer care team member, I should be able to get feedback from the users	2	Medium	RM, RL, RR, SR
Sprint-4		USN-18	As a customer care team member, I should be available 24/7 to increase customer base and reflect on the queries raised by them	2	Medium	RM, RL, RR, SR

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	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	Due
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	Due
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	Due

Velocity:
Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous" />
  <link rel="stylesheet" href="static/css/style.css" />
  {% block head%}{% endblock %}
</head>

<body>
  <div class="wrapper">
    <!-- Sidebar  -->
    <nav id="sidebar">
      <div class="sidebar-header">
        <h3>Inventory</h3>
      </div>

      <ul class="list-unstyled components">
        <li>
          <a href="/dashboard">Dashboard</a>
        </li>
        <li>
          <a href="/orders">Orders</a>
        </li>
        <li>
          <a href="/suppliers">Suppliers</a>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous" />
  <link rel="stylesheet" href="static/css/style.css" />
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
    crossorigin="anonymous"></script>

  <!-- <link rel="stylesheet" href="{{url_for('static', filename='css/style.css')}}"> -->
  {% block head%}{% endblock %}
</head>

<body>
  <div class="container mt-5">{% block body %} {% endblock %}</div>
</body>

</html>
```

7.2 FEATURE 2

File Edit Selection View Go Run Terminal Help

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

index.html

C: > Users > mramy > Downloads > IBM-Project-25382-1659961247-main > IBM-Project-25382-1659961247-main > Project Development Phase > Sprint 1 > templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Sidebar Menu</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous" />
  <link rel="stylesheet" href="static/css/style.css" />
</head>

<body>
  <div class="wrapper">
    <!-- Sidebar  -->
    <nav id="sidebar">
      <div class="sidebar-header">
        <h3>Inventory</h3>
      </div>

      <ul class="list-unstyled components">
        <li class="active">
          <a>Dashboard</a>
        </li>
        <li>
          <a href="#">Add item</a>
        </li>
        <li>
          <a href="#pageSubmenu">Pages</a>
```

Restricted Mode ⊗ 0 ⚠ 0 Ln 1, Col 1 Spaces: 2 UTF-8 LF HTML

File Edit View Code Window Help app.py - ...\Project Development Phase\Sprint 3

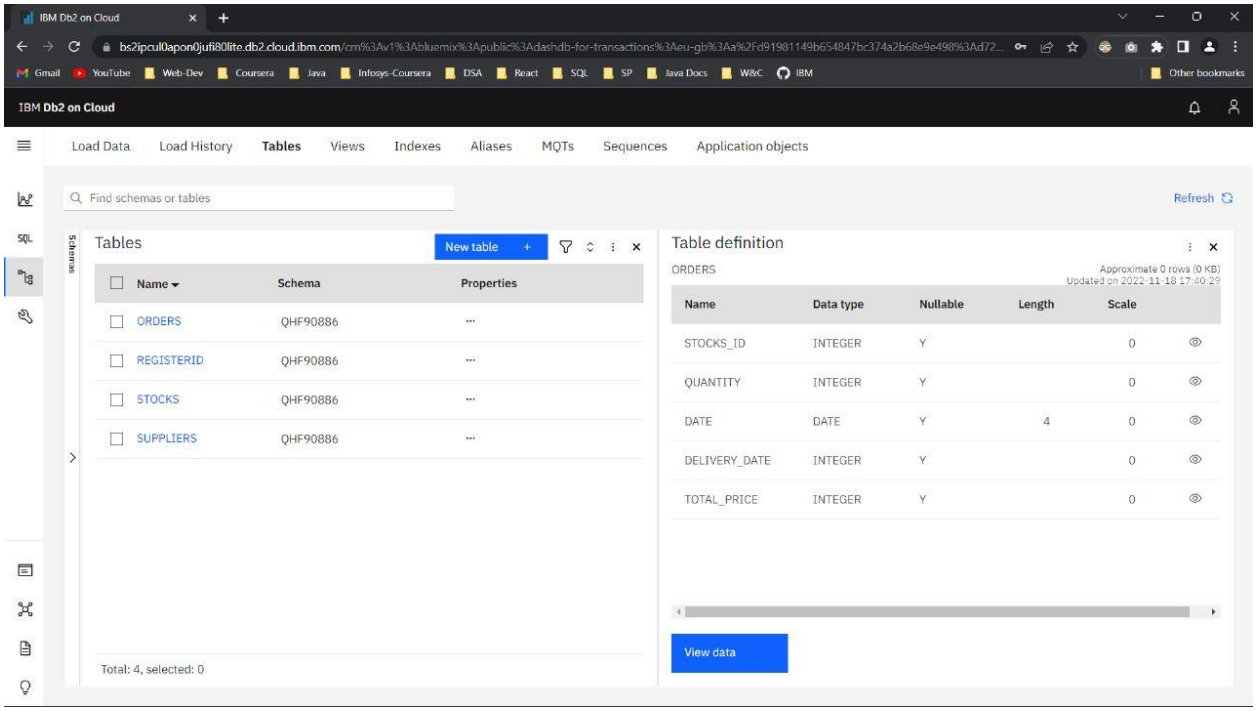
app.py app.py

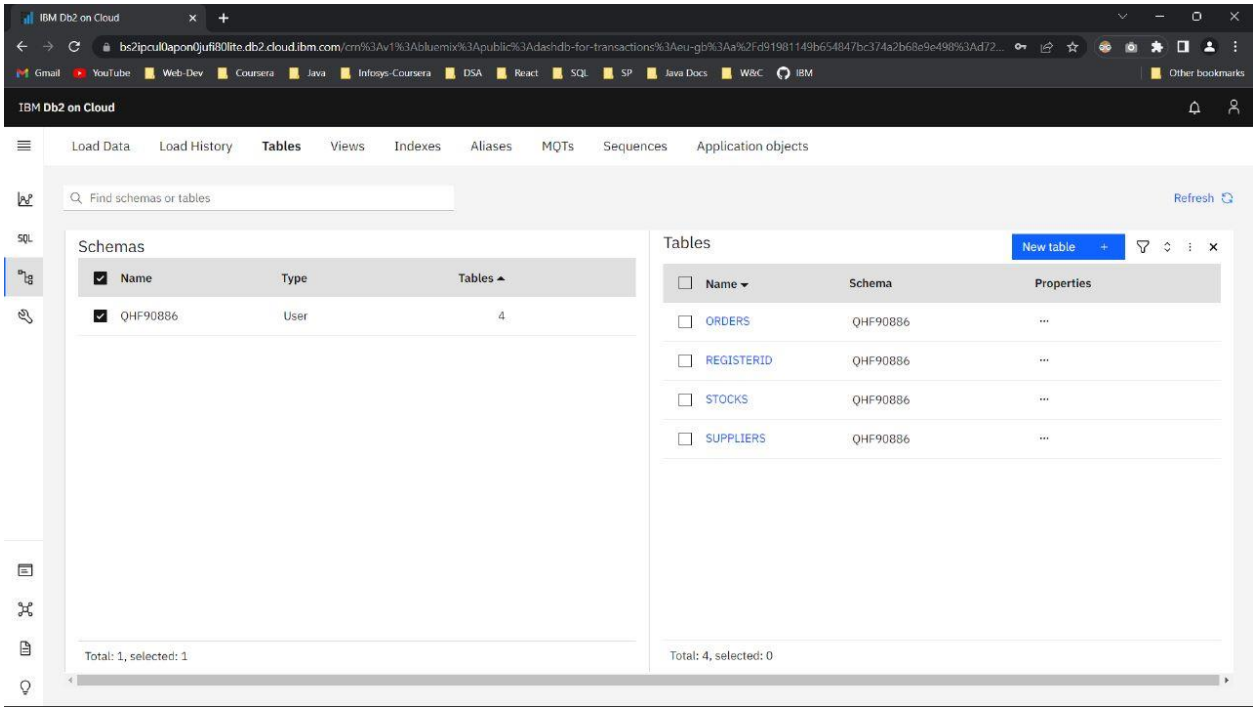
```python
import ...

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864;SECURITY=SSL;SSLServerCertificate=DigiCer

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


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('inventorymanagement1.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM registerid WHERE email="{id}"')
        user = cur.fetchall()
    return render_template("user_info.html", user=user[0])
```

LightEdit mode. Access full IDE 1:1 LF UTF-8 Autosave: off

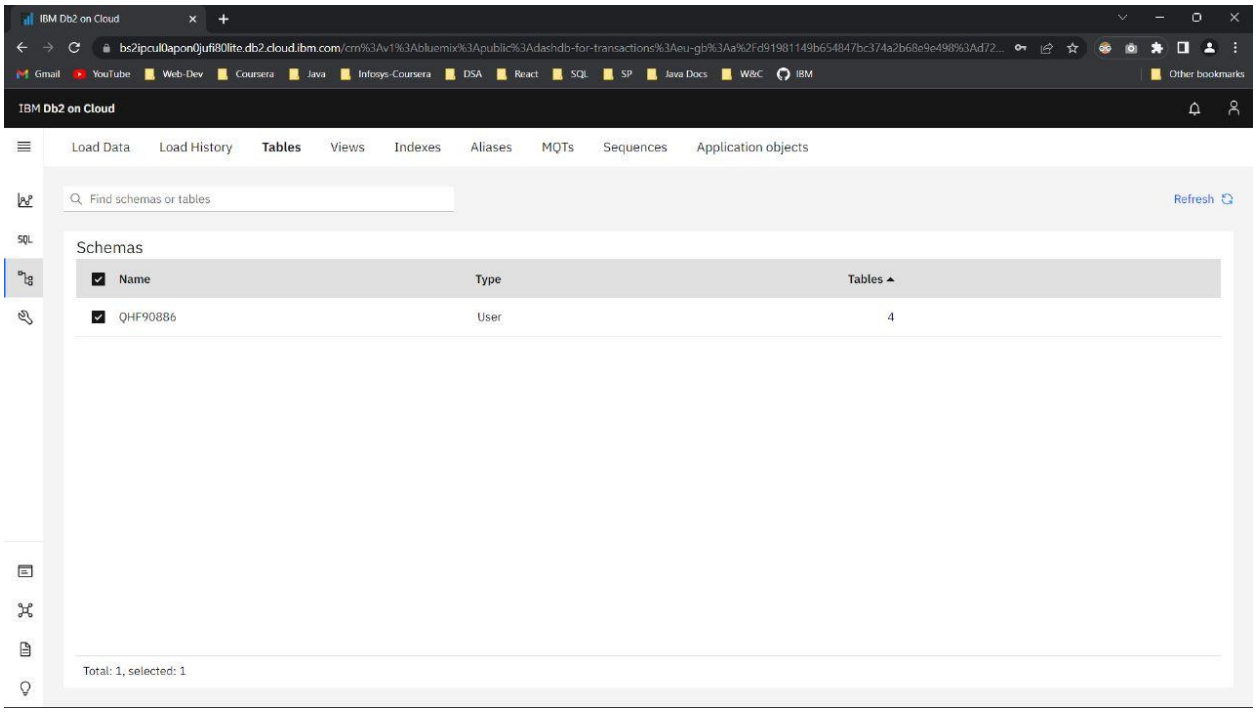
8. TESTING

8.1 TEST CASES



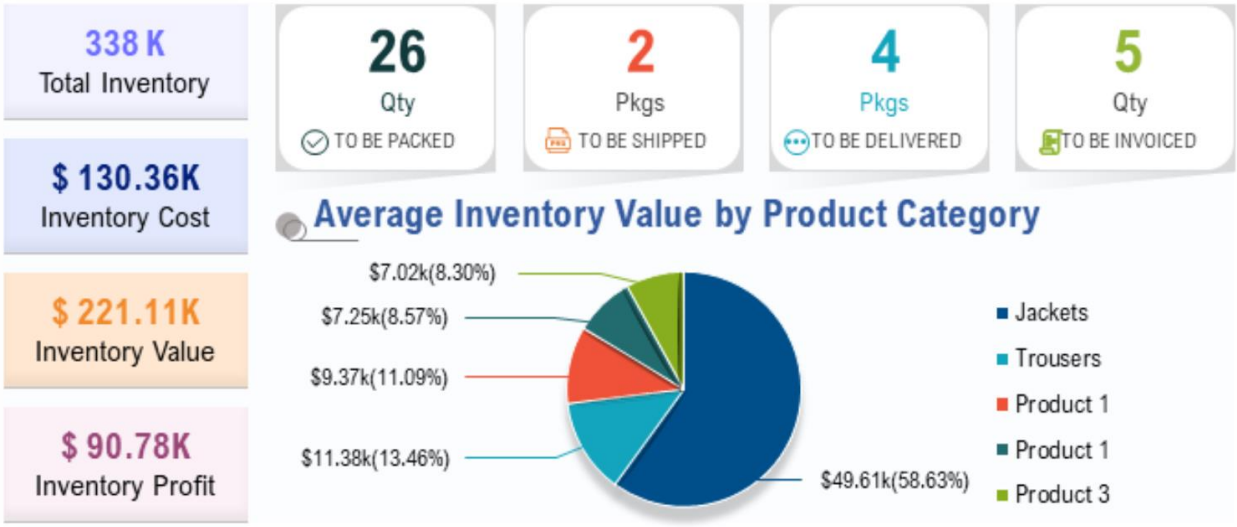


8.2 User Acceptance Testing



9.RESULTS

9.1 PERFORMANCE METRICS:

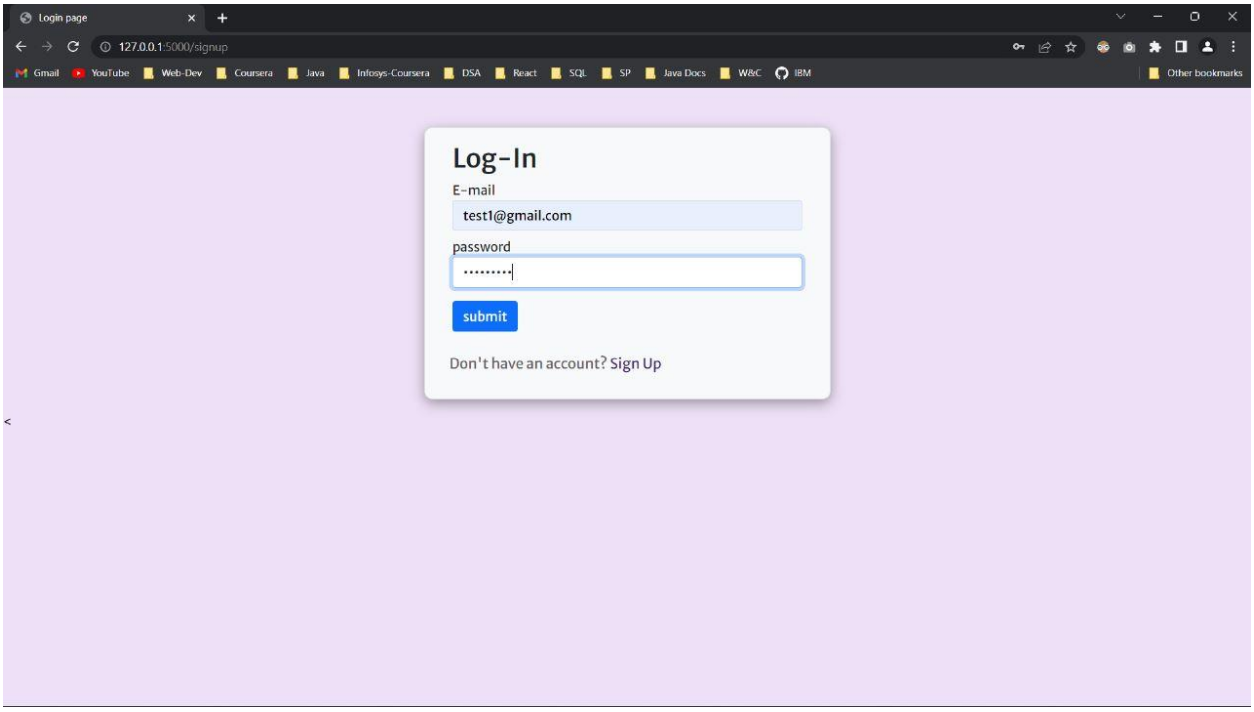


Top Returned Items

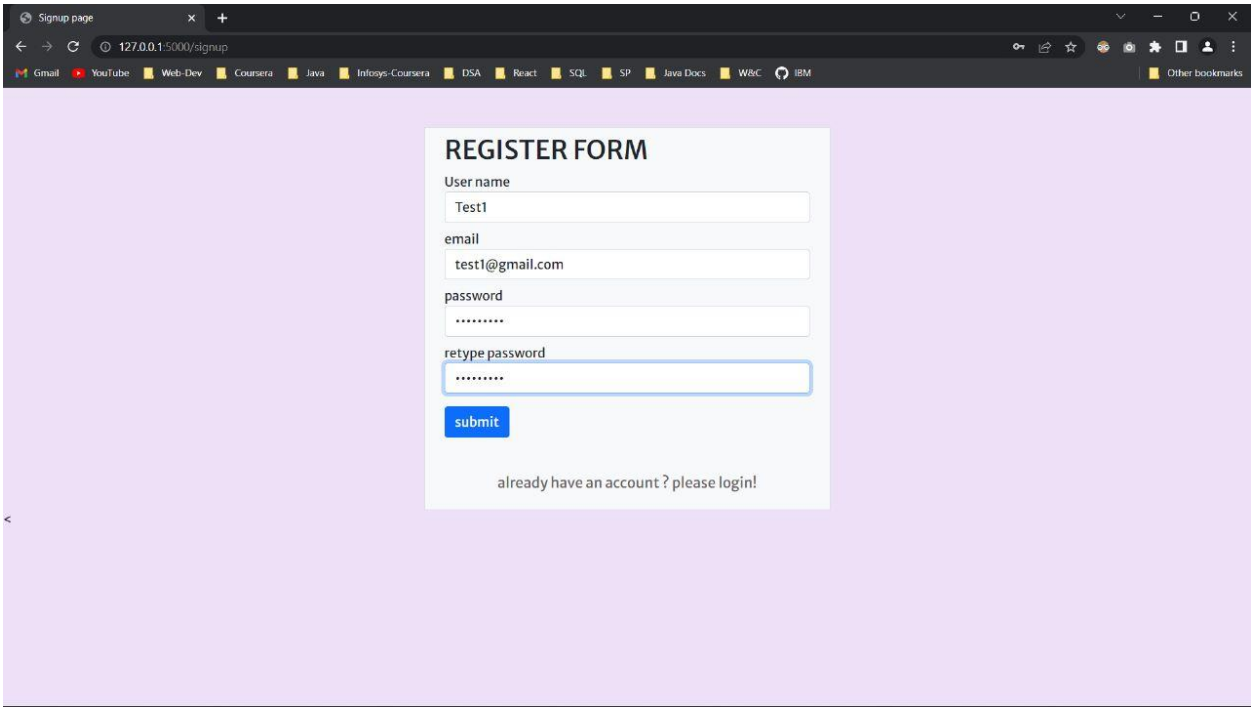
	Damaged	Wrong fit	Wrong color	Add text here
Leggings			6 pcs	3 pcs
Trousers				
Product 1	4 pcs			
Product 2		6 pcs		2 pcs
Product 3	3 pcs	4 pcs	9 pcs	

OUTPUT

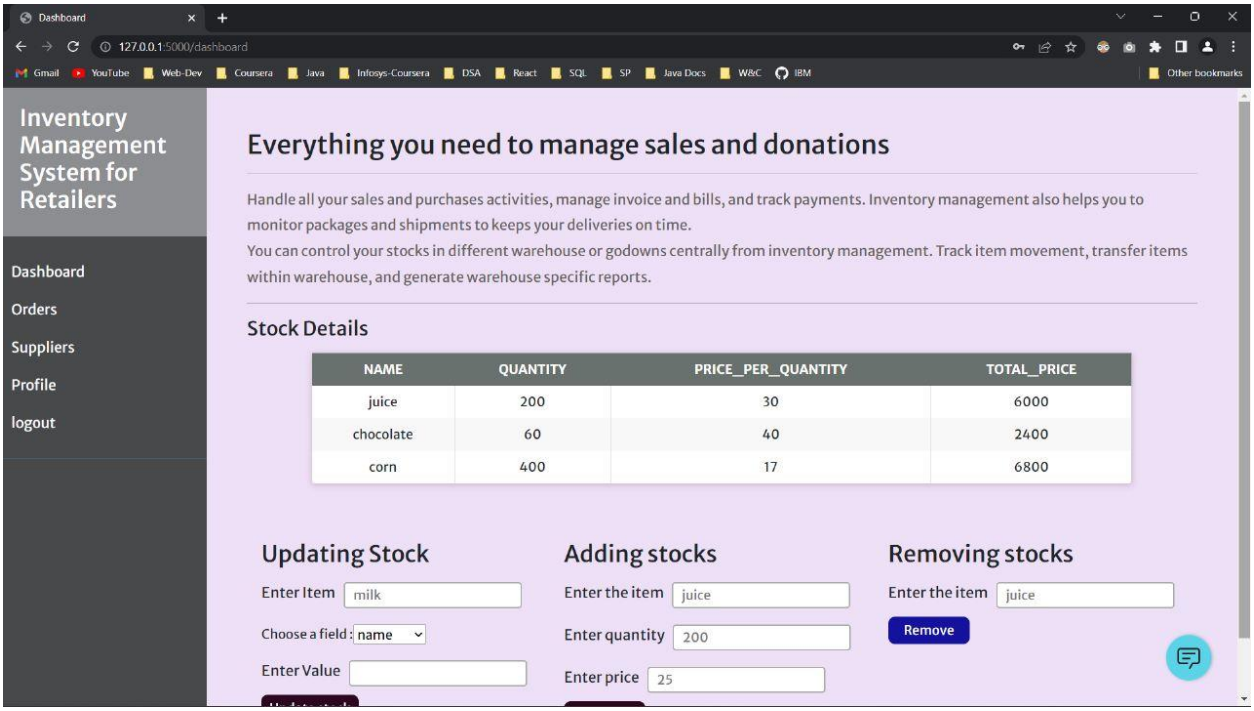
Login.html



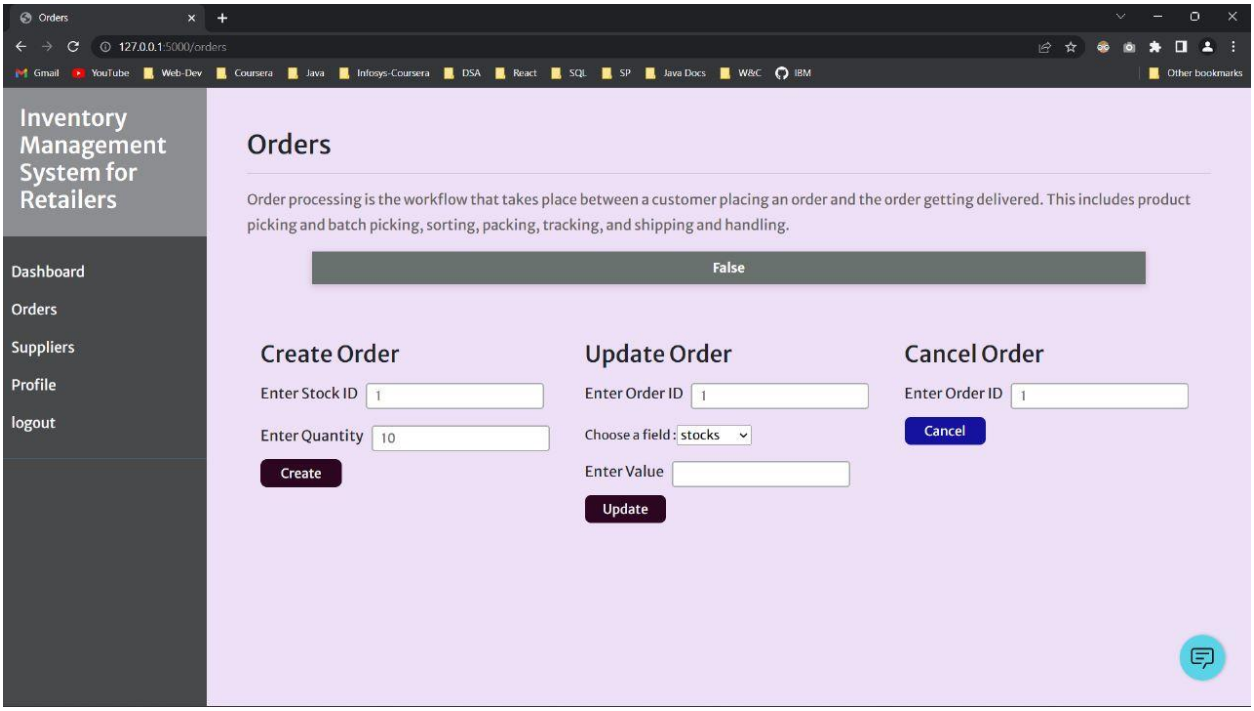
Register.html



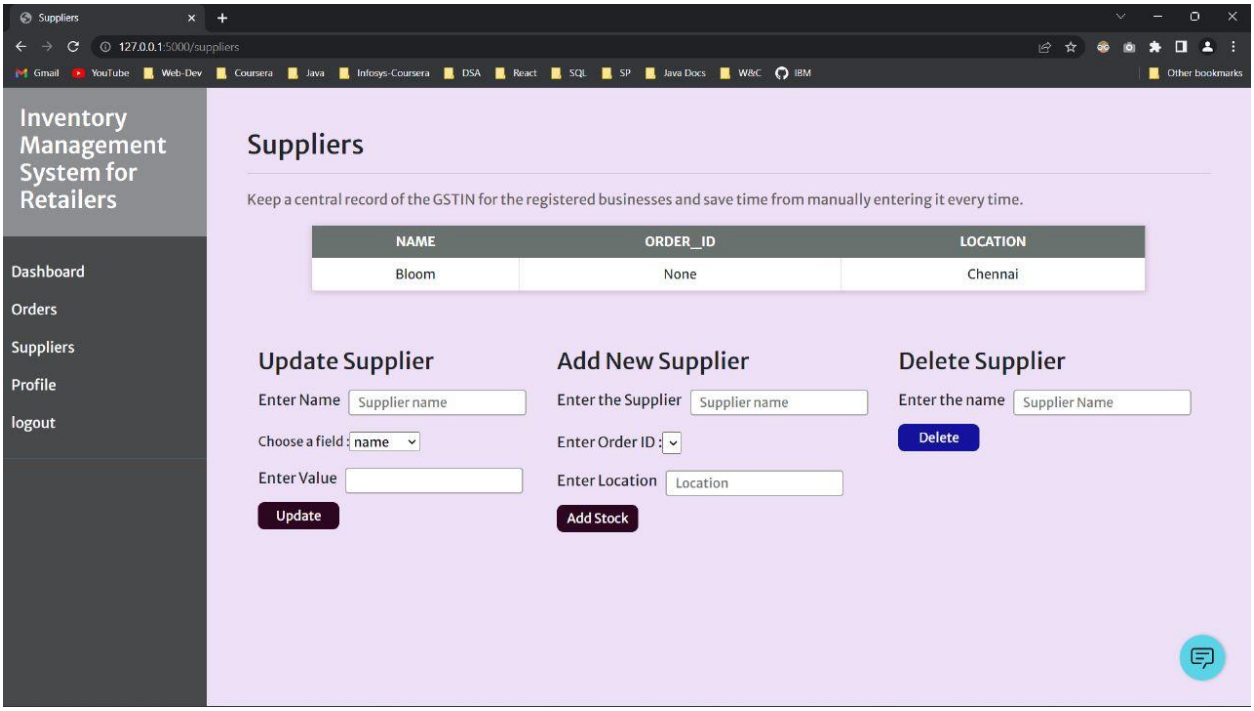
Dashboard.html



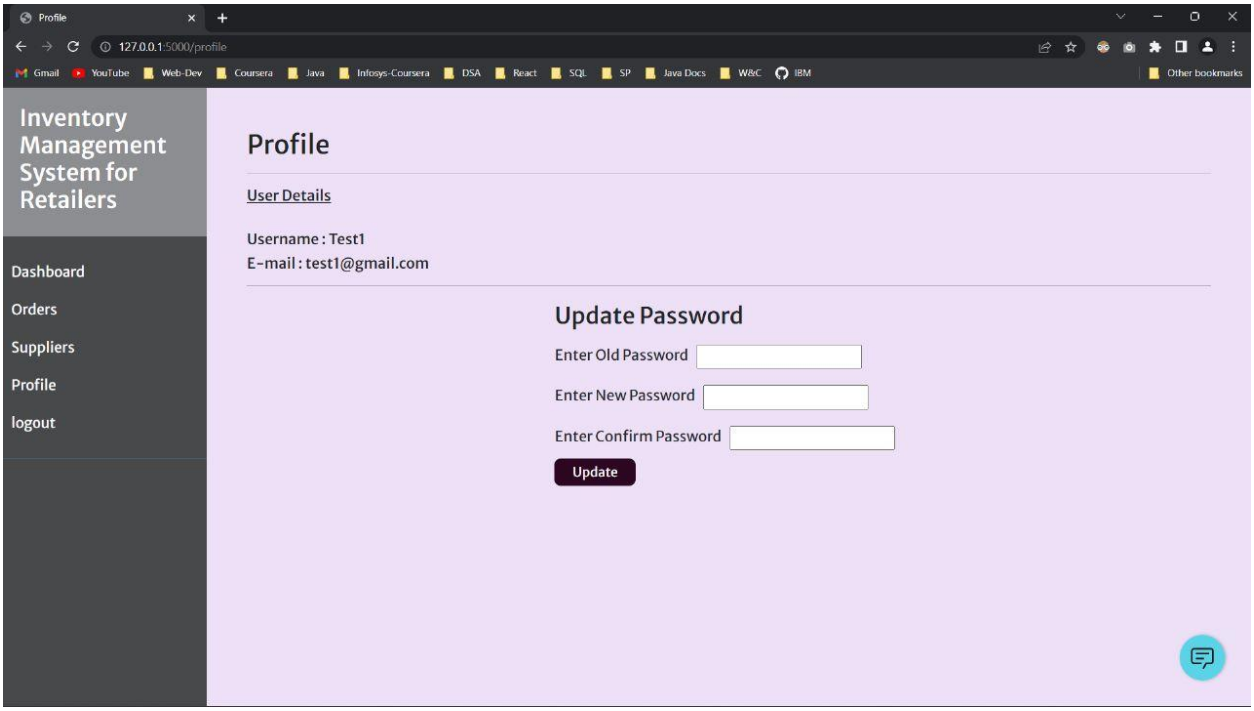
Orders.html



Suppliers.html



Profile.html



10. ADVANTAGES

- Automated inventory management.
- Inventory forecasting for holiday and peak season readiness.
- Prevent stocks outs and overselling.
- Reduce ecommerce's business costs.
- Better inventory planning and forecasting.
- Improving supply chain operation.
- Add new selling channels easily

DISADVANTAGES

- Inconsistent tracking, incomplete data.
- Changing demands.
- Supply chain complexity
- Insufficient order management.
- Overselling is a result of flawed communication.
- Inefficient Warehouse Management
- Inadequate Access

11. CONCLUSION

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

12. FUTURE SCOPE

FORECASTS AND INSIGHTS

- Interactive user interface design
- Online payment system can be added
- Sales and product return system will be added in order to make return of the products
- Lost and breakages
- Making the system flexible in any type

APPENDIX

Source code

App.py

```
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
from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-840d-d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=qhf90886;PWD=Txn7hauNIsZt3TEQ;", ' ' , ' ')

app = Flask(__name__)
app.secret_key = 'julie'
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('inventorymanagement1.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM registerid 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':
        email = request.form['email']
        pd = request.form['password_2']
        print(email, pd)
        sql = "SELECT * FROM registerid WHERE email =? AND password_2=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        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')
            return redirect(url_for('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 registerid 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 registerid (USERNAME,EMAIL,PASSWORD_2) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, username)
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.bind_param(pstmt, 3, pw)
            print(pstmt)
            ibm_db.execute(pstmt)
            mg = 'You have successfully registered click login!'
            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
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 name = ?'
            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'])
```

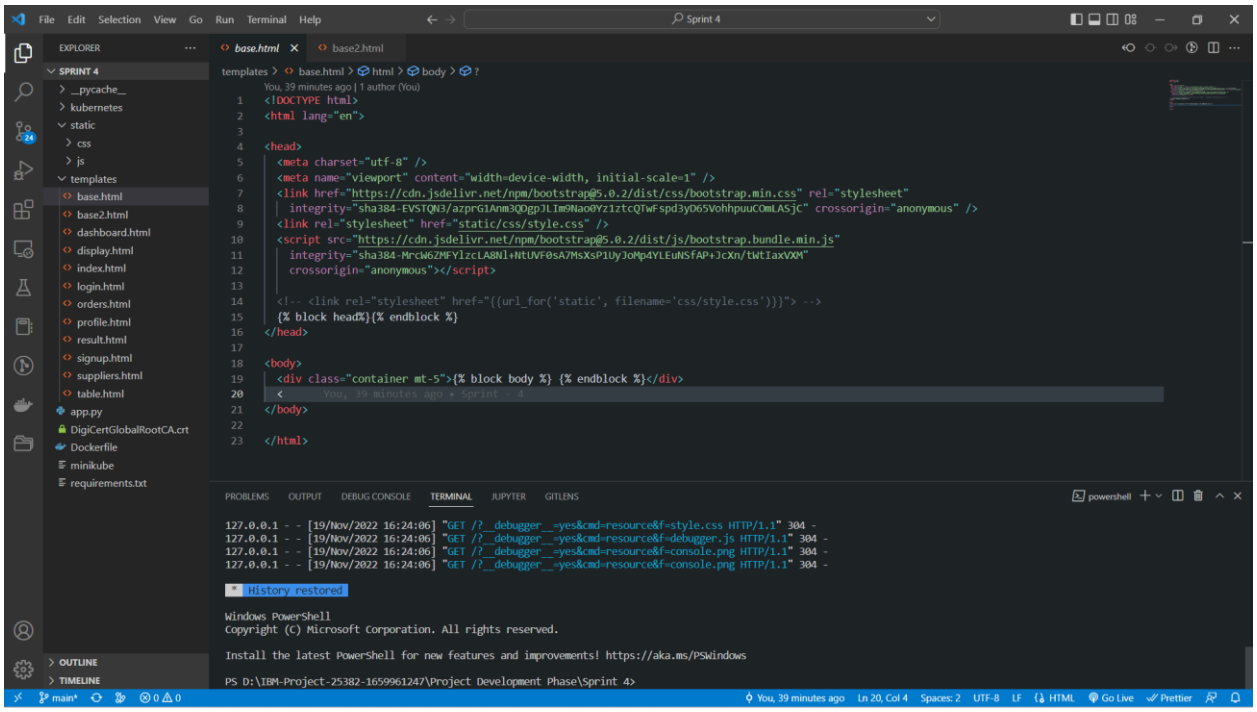
```
@login_required
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        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 STOCKS_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 registerid 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(debug=True)
```

Frontend



Appendix

GitHub & Project Demo Link

GITHUB LINK: https://github.com/IBM-EPBL/IBM-Project-25382-1659961247

PROJECT DEMO LINK:
https://drive.google.com/file/d/1bCtPL3bvUDbZE-BMEpwxSwOHsgUCGeAO/view?usp=sharing