IBM - NALAIYATHIRAN PROJECT

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

PROJECT REPORT DOCUMENTATION

TEAM ID: PNT2022TMID35256

Sl. No	Name	Roll No
1	ASHWIN MENON (Leader)	2019103510
2	MORAREDDY RAHUL REDDY	2019103036
3	VIGNESH KUMAR	2019103594
4	KRISHNA TEJA	2019103032

PROJECT REPORT FORMAT:

1. **Project Introduction**
   - 1.1 Project Overview
   - 1.2 Purpose

2. **Literature Survey**
   - 2.1 Existing Problem
   - 2.2 References
   - 2.3 Problem Statement Definition

3. **Ideation and Proposed Solution**
   - 3.1 Empathy Map And Canvas
   - 3.2 Ideation And Brainstorming
   - 3.3 Proposed Solution
   - 3.4 Problem Solution Fit

4. **Requirement Analysis**
   - 4.1 Functional Requirement
   - 4.2 Non - Functional Requirement

5. **Project Design**
   - 5.1 Data Flow Diagrams
   - 5.2 Solution and Technical Architecture
   - 5.3 User Stories

6. **Project Planning & Scheduling**
   - 6.1 Sprint planning & Estimation
   - 6.2 Sprint Delivery Schedule
   - 6.3 Reports from JIRA

7. **Coding and Solutioning(Features added along with code)**
   - 7.1 Feature 1
   - 7.2 Feature 2

1. INTRODUCTION

1.1 PROJECT OVERVIEW

A computerized application called an inventory management system (IMS) assists firms in keeping track of and managing their inventories. Businesses can lower their inventory expenses and increase supply accuracy and timeliness by using an IMS. Businesses use an IMS for a variety of things, including suppliers, finished goods, and raw materials. The majority of IMS applications have tools for shipping, receiving, and order management. Numerous systems additionally have reporting features including monthly business reviews and sales and operations planning reports (S&OP) (MBR). Some systems are also created for particular sectors of the economy, such industry and healthcare. Retail IMS systems can automate the ordering and tracking of goods and are designed for small enterprises with little funding and resources. producers and substantial wholesalers often purchase more robust IMS systems that are designed specifically for their needs.

An inventory management system is in charge of making sure that customers may purchase the appropriate product in the right amount at the right time and price. An effective IMS will assist increase productivity, save expenses, and lessen the chance of obsolescence or excess inventory. It can be difficult to implement an IMS since many firms find it difficult to manage their inventories effectively without a committed resource. However, a clear plan and effective implementation techniques can help to guarantee a positive result. Understanding the advantages and disadvantages of the system is the first step in putting a new inventory management system into place.

1.2 PURPOSE

An inventory management system's major objective is to assist businesses in keeping track of the number, location, and state of all goods. Decisions about resource allocation and when to place new product orders can then be determined to use this information. Systems for inventory management can also assist businesses in reducing the amount of inventory they have, which can save money and boost earnings.

Inventory needs will increase as your business expands. The complexity of your inventory will increase as products arrive from various vendors and warehouses. It will be tough and time-consuming to manage your inventory manually, which will make it impossible for you to keep enough stock on hand to meet client demand and expand your business.

2 . LITERATURE SURVEY

2.1 EXISTING PROBLEM

The inability to track inventory in real time is one issue that plagued the majority of systems. This is due to a lack of integration between the systems and the point-of-sale system. This indicated that the inventory was not continuously updated. Sales and profits were lost as a result of this.

2.2 REFERENCES

- Research paper on Inventory management system
- Inventory management efficiency analysis: A case study of an SME company
- A Study of Inventory Management System - Case Study
- Informative Review on Inventory Control System
- Improvement of Inventory Management System Processes by an Automated Warehouse Management
- Study of Smart Inventory Management System Based on the Internet of Things (IOT)
- Research and Design of the Intelligent Inventory Management System Based on RFID

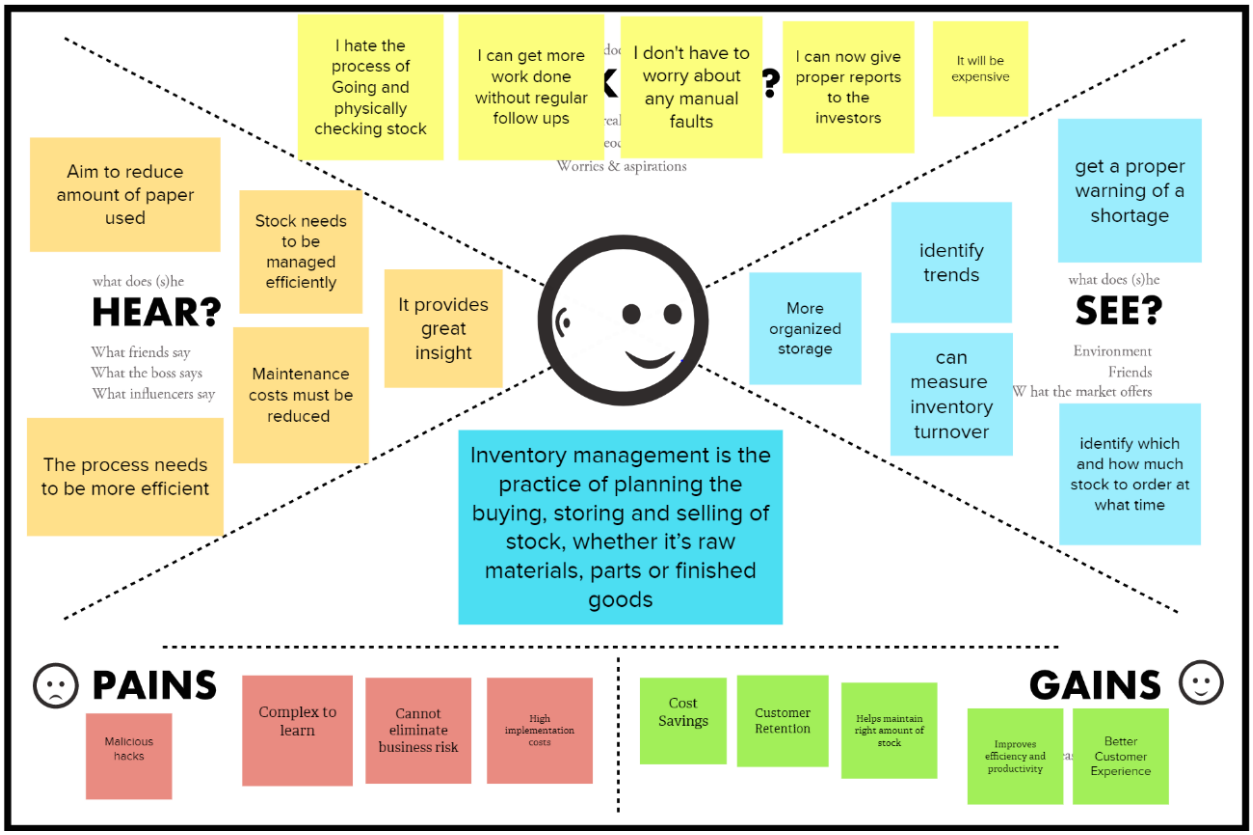
2.3 PROBLEM STATEMENT DEFINITION

This project is aimed at developing a desktop-based application called Inventory Management System for managing the inventory of any organization. This system can be used to store the details of the inventory, stock maintenance and update the inventory based on the sales details, and generate inventory reports weekly or monthly based.

I am	Retailers and Customers
I'm trying to	Have more insights on stocks and their availability to increase productivity
But	Manual management of the stocks are difficult and existing systems aren't much flexible
Because	Too much stock items cause bigger problems and current systems are obsolete
Which makes me	Want to create better inventory management system and increase the accuracy and flexibility of the vendors

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP AND CANVAS



3.2 IDEATION & BRAINSTORMING

Team Gathering, Collaboration and Select the Problem Statement

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

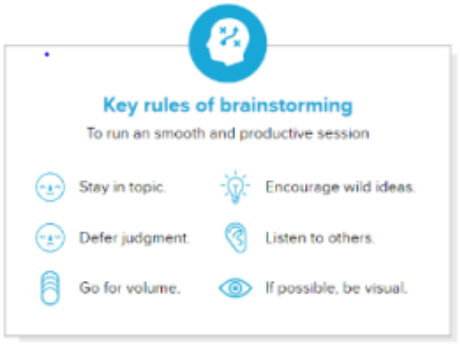
1 Define your problem statement

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

How might we design an inventory management system?



Brainstorm

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

10 minutes

Ashwin

- Send Mail when minimum stock limit is reached
- Enhanced user interface
- Periodic Analysis of Sales Reports
- Managing customer feedback
- Maintain proper product categories
- Tax Calculations

Rahul

- Analyze low and high selling products
- Display graphs to show clear picture
- Provide product insights
- Enabling customer return policy
- Monitor products for cost changes
- Managing multiple orders

Krishna Teja

- Avoid overstocking of products
- Enable remote access of software
- Managing customer account
- Maintain records for the product
- Enabling Multiple Payment Options
- Occasional discounts for the products

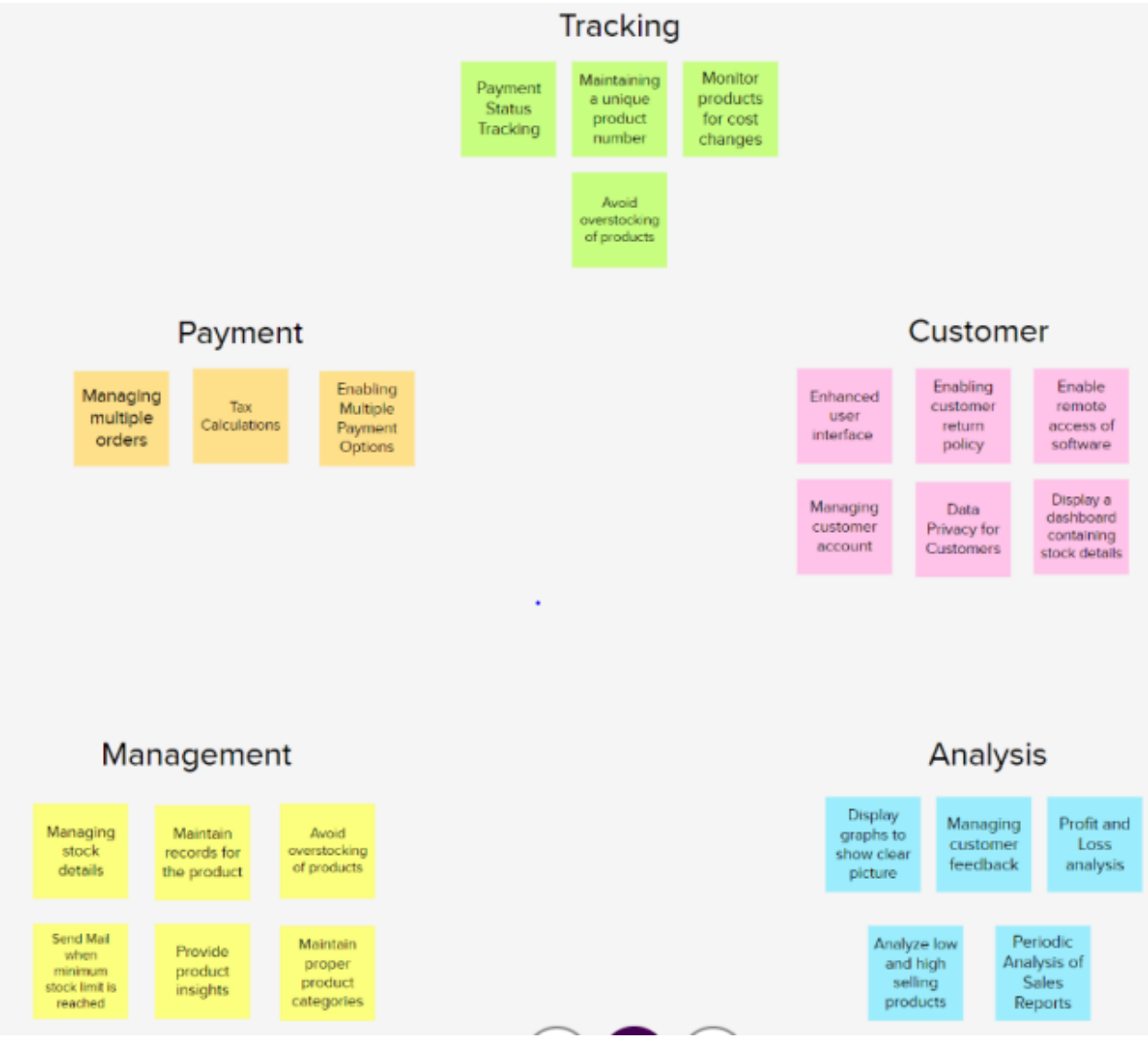
Vignesh

- Data Privacy for Customers
- Maintaining a unique product number
- Payment Status Tracking
- Managing stock details
- Display a dashboard containing stock details
- Profit and Loss analysis

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



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

3.3 PROPOSED SOLUTION

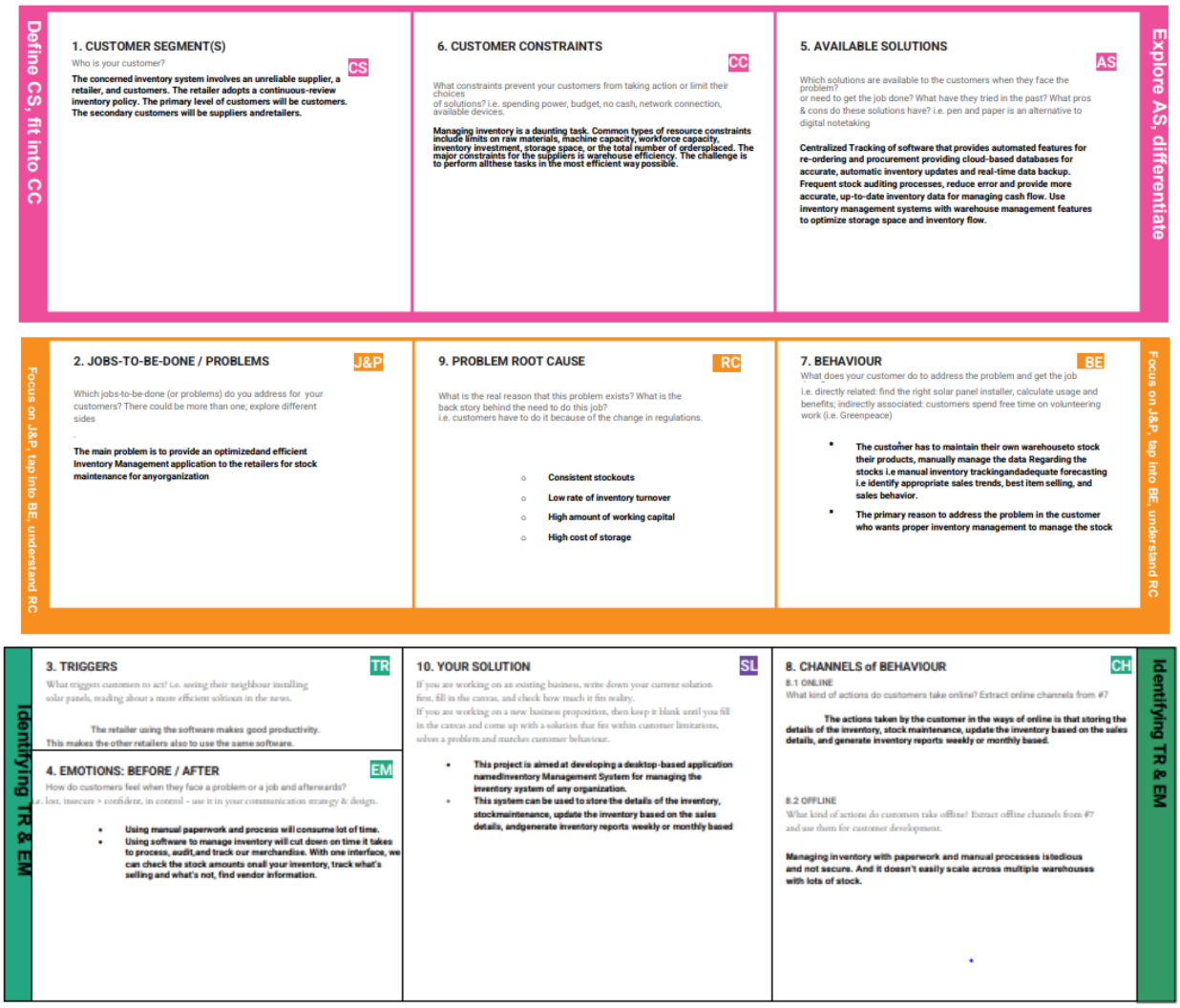
S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	How to provide a systematic system that not only records data but also allows for easier arrangement of the inventory mainly from the retailer's end?
2.	Idea / Solution description	The application mainly focuses on helping the retailers track and manage stocks of products. The System will ask the retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. After they login in, they can update the details of a stock that they possess and also add a product/stock with relevant details. 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.

3.	Novelty / Uniqueness	Providing a user-friendly environment to maintain the stock by -> Display of Dashboard containing stock details ->Report on weekly or monthly basis Apart from the standard features of the inventory management system like handling products, warehouses, locations we also plan to include the feature of sales prediction using regression and the previous sales data within our application.
4.	Social Impact / Customer Satisfaction	This system can have a positive impact on social life. This system improves the Management of resources and reduces excess inventory and thus reduces the wastage of products. It is also easy to use and can arrange the inventory with efficiency. It also improves the relationship with vendors and suppliers and can negotiate better deals with the suppliers by knowing the demand beforehand.
5.	Business Model (Revenue Model)	Retailers can order the right amount and type of stock at the right time with the aid of an inventory management system. It eliminates the unnecessary expense for the retailers.

6.	Scalability of the Solution	A scalable cloud architecture is made possible through virtualization. Unlike physical machines whose resources and performance are relatively set, virtual machines virtual machines (VMs) that we use in IBM cloud are highly flexible and can be easily scaled up or down

3.4 PROBLEM SOLUTION FIT

Project Title: Inventory Management for Retailers **Project Design Phase-I - Solution Fit Template** **Team ID:** PNT2022TMID35256



4. REQUIREMENT ANALYSIS

4.1 SOLUTION & TECHNICAL REQUIREMENTS

Functional Requirements:
Following are the functional requirements of the proposed solution.

	Functional Requirement	Sub-Task
1	User Registration	Mail-in registration /Using a form to register
2	User Confirmation	Call/OTP/Email confirmation
3	Logging in	Enter the essential credentials to access the application (email ID and password)
4	Dashboard	View the products details (Name, description, quantity)
5	Adding items to the Inventory list	The inventory can be updated by users with items they want to purchase.
6	Updating of stock	Increasing the availability of a specific product

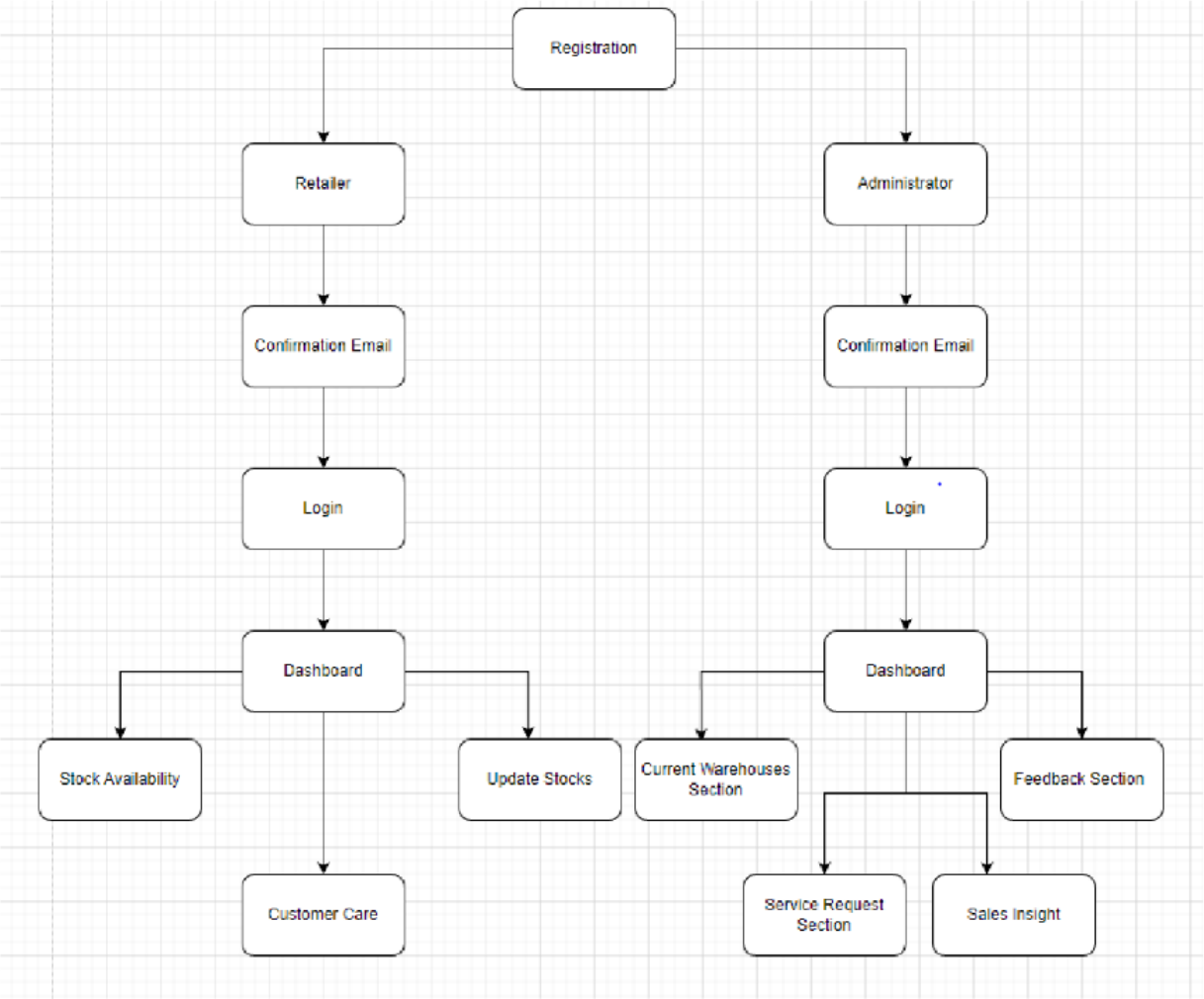
Non-functional Requirements: Following are the non-functional requirements of the proposed solution.

	Non-Functional Requirement	Description
1	Usability	If the system has a steep learning curve, the organisation in need of an inventory management system is unlikely to purchase it. • The user interface is straightforward and simple to use. • The design and colours are consistent. • The websites are mobile-friendly and responsive. • Email delivery must be quick.
2	Security	Security refers to the safety and management of the inventory of a company such that only authorised personnel are allowed to access them. • Login system is used to provide authentication. • Users need to create account and verify it with their email OTP. • Cookie based security is user for authentication on client side.

3	Reliability	• To ensure that the app functions properly even when mistakes occur during runtime, exception handling will be done at the code level. • To ensure sustained operation, many instances of the App would be online.
4	Performance	The efficiency with which various tasks in an inventory management system can be completed determines its performance. • Reduces manpower, costs, and time. When stocks are unavailable, emails will be issued automatically. • Improves the efficiency of the business process. • Enhances the performance of organisations. • Even at minimal bandwidth, it will perform quickly and securely.
5	Availability	The use of IBM DB2 ensures high availability
6	Scalability	An inventory management system's scalability relates to the expansion of its activities. • DB2 is extremely scalable. • By reusing the code, the code is created effectively to allow for the addition of new features with few adjustments. Docker, which is very scalable, is utilised in the IBM Container Registry.

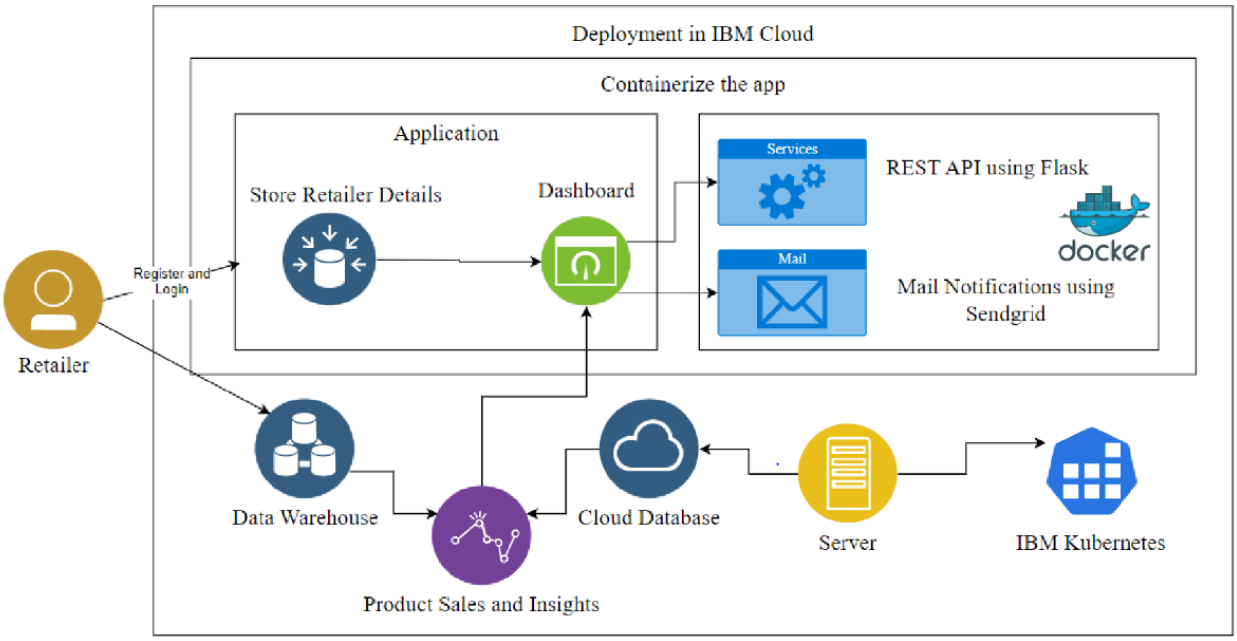
5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 TECHNICAL ARCHITECTURE

Solution Architecture Diagram:



5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	By providing my email address, a password, and a password confirmation, I can register for the application as a user.	I can access my account / dashboard	High	Sprint-1
		USN-2	I can sign up for the application as a user by email.	I can access my account / dashboard	Medium	Sprint-1
	Confirmation	USN-3	I will receive a confirmation email once I have registered for the application.	I can get a confirmation for my email and password and create an authenticated account.	Medium	Sprint-1
	Login	USN-4	By entering the registered email address and password, I can access the application as a user.	I can log onto the application with the verified email and password	High	Sprint-1

	Dashboard	USN-5	As a user, I can view the products which are available.	Once I log on to the application, I can view the inventory.	High	Sprint-2
	Stock Update	USN-6	I can add items to the stock list as a user that aren't listed in the dashboard.	If any of the products are not available, as a user I can update the inventory.	Medium	Sprint-2
	Sales Prediction	USN-7	I can access a sales forecasting tool as a user, which will enable me to more accurately forecast the volume of orders.	The sales prediction tool should forecast the sales so that I, as a User, can order appropriately.	Medium	Sprint-3
Administrator	Request to Customer Care	USN-8	I can contact the Administrator as a user and request any assistance I need with services.	As a user, I can contact Customer Care and get support from them.	Low	Sprint-4
	Give feedback	USN-9	I should be able to report any difficulties I encounter.	As user, I can give my support in my possible ways to the administrator and to the administration.	Medium	Sprint-4

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

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 then confirming my password.	5	High	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-1		USN-2	As a user, I can register for the application through email.	3	Medium	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-1	Confirmation	USN-3	As a user, I will receive a confirmation email once I have registered for the application.	4	Medium	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K

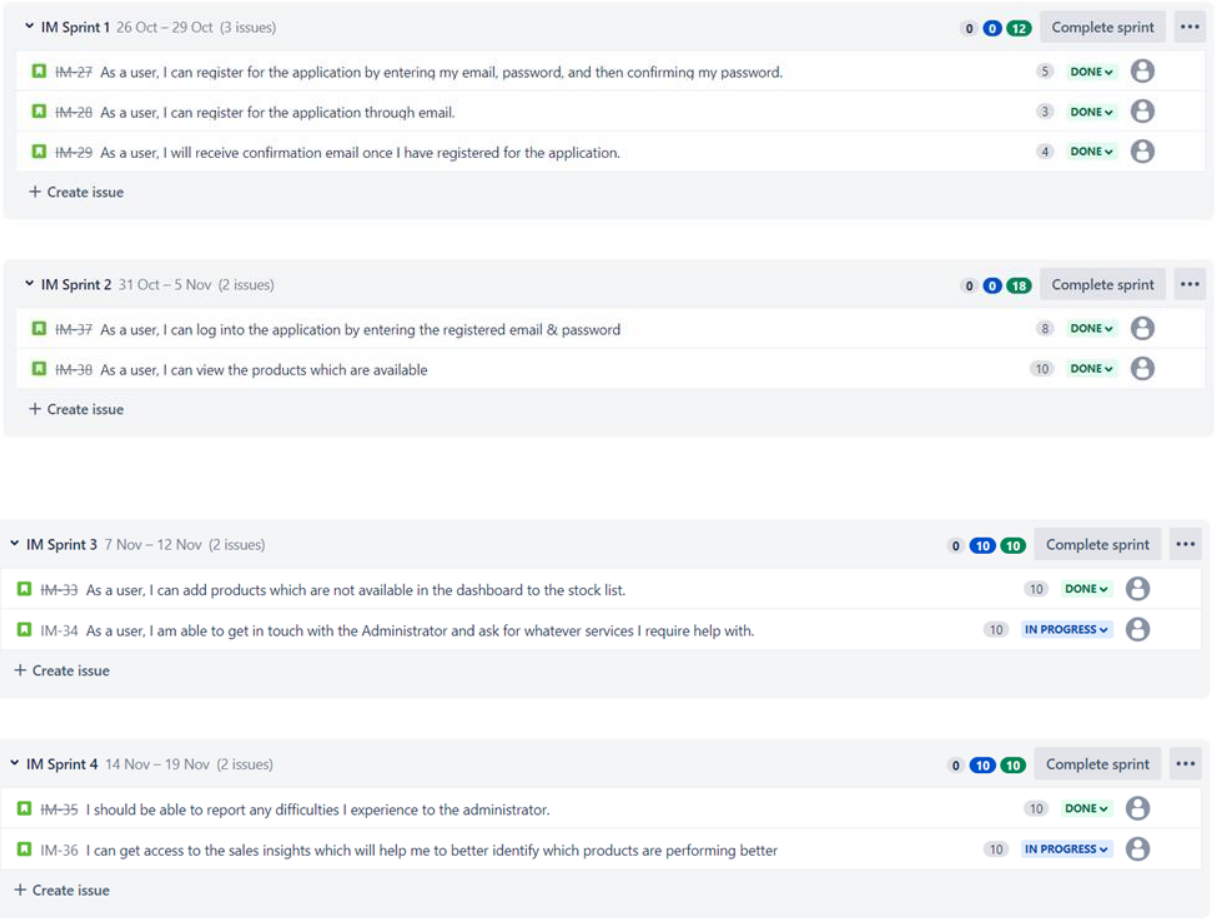
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login	USN-4	As a user, I can log into the application by entering the registered email & password	8	High	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available.	10	High	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-2	Stock Update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list.	10	Medium	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-3	Sales Prediction	USN-7	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	10	High	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-4	Administration	USN-8	As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.	10	Low	Ashwin Menon,Krishna Teja,Rahul Reddy Mora Reddy,Vignesh Kumar K
Sprint-4		USN-9	I should be able to report any difficulties I experience to the administrator.	10	Medium	Ashwin Menon,Krishna Teja,Rahul Reddy MoraReddy,Vignesh Kumar K

6.2 SPRINT DELIVERY SCHEDULE

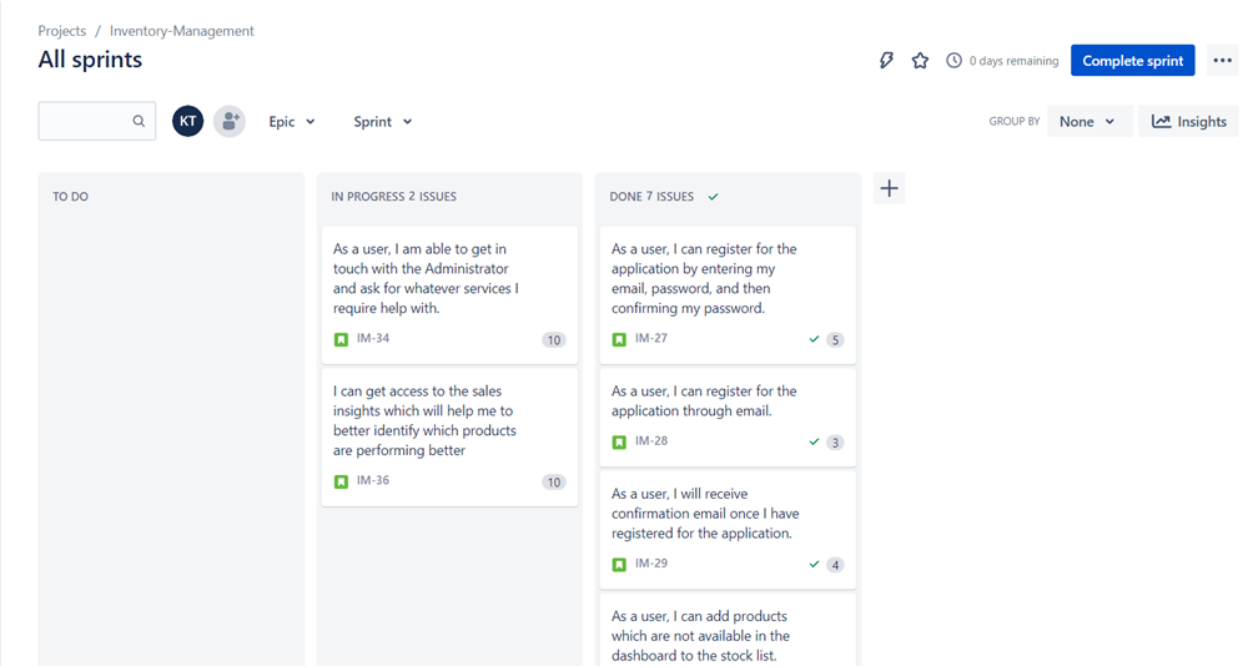
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	4 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	15 Nov 2022
Sprint-3	10	6 Days	07 Nov 2022	12 Nov 2022	10	22 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

6.3 REPORTS FROM JIRA

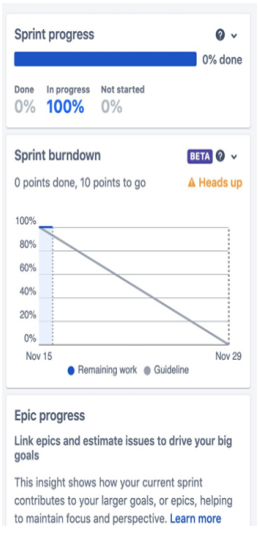
Creation of the backlog (issues) for the project:



Creation of the scrum boards for the project:



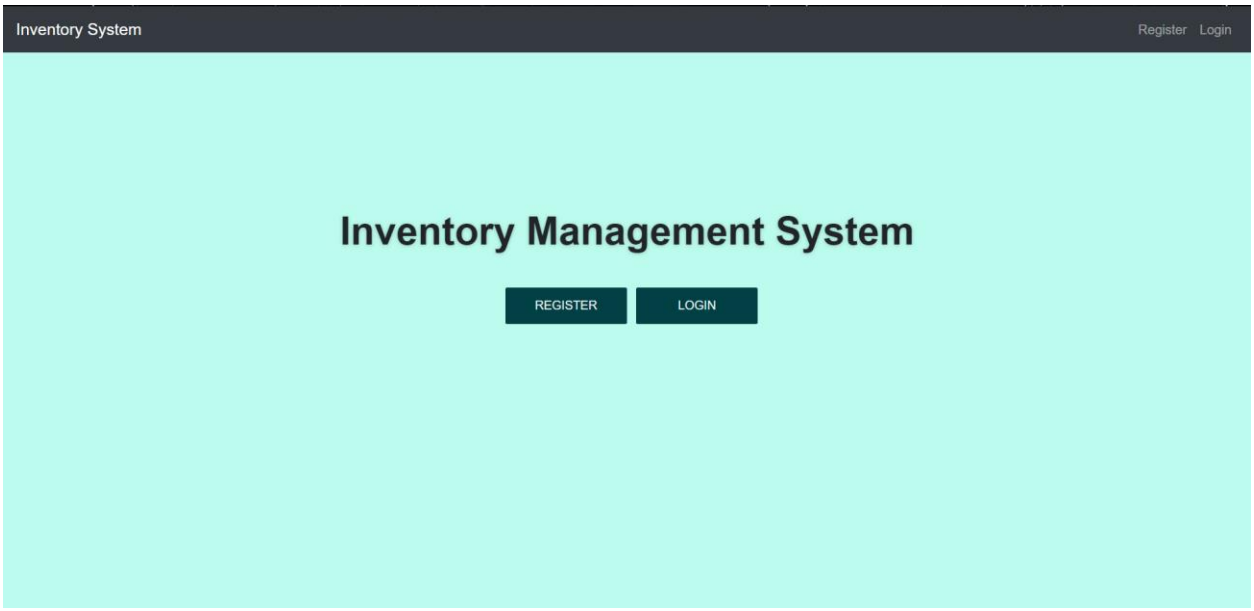
Burndown for the project:



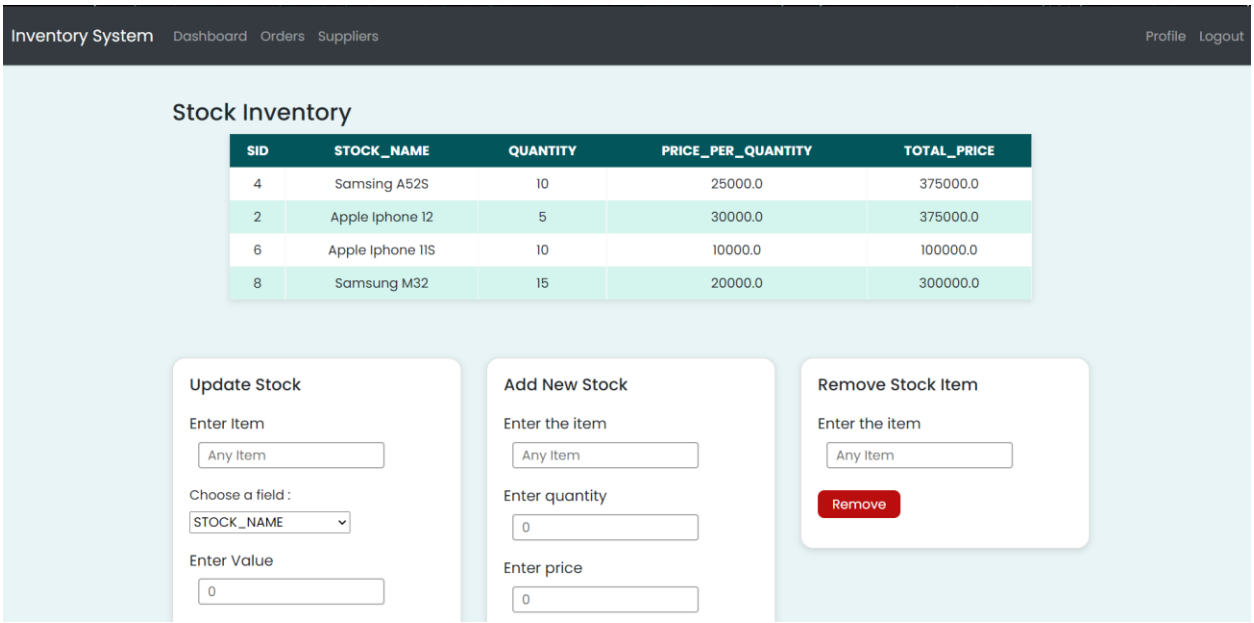
7.CODING & SOLUTIONING

7.1 FEATURE-1

Users can register or login through this dashboard

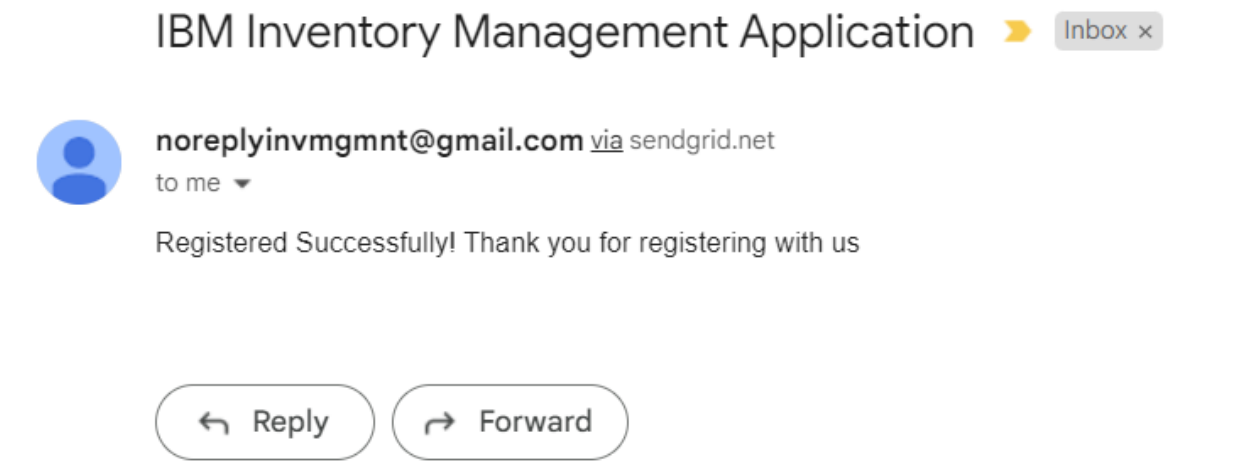


Used flask web framework to create an interactive dashboard



7.2 FEATURE-2

Used sendgrid for autonomous mails



7.3 DATABASE SCHEMA

User Table

Table definition

USERS

No statistics available.

Name	Data type	Nullable	Length	Scale
USERNAME	VARCHAR	N	32	0
EMAIL	VARCHAR	N	52	0
PASSWORD	VARCHAR	N	52	0

Inventory Stock Table

Table definition

STOCK

No statistics available.

Name	Data type	Nullable	Length	Scale
SID	INTEGER	N		0
STOCK_NAME	VARCHAR	Y	350	0
QUANTITY	INTEGER	Y		0
PRICE_PER_QUANTITY	DOUBLE	Y		0
TOTAL_PRICE	DOUBLE	Y		0

Orders Table

Table definition

ORDERS

No statistics available

Name	Data type	Nullable	Length	Scale
OID	INTEGER	N		0
STOCK_ID	INTEGER	Y		0
QUANTITY	INTEGER	Y		0
ODATE	VARCHAR	Y	30	0
DELIVERY_DATE	VARCHAR	Y	30	0
PRICE	DOUBLE	Y		0

Suppliers Table

Table definition

SUPPLIERS

No statistics available.

Name	Data type	Nullable	Length	Scale
ORDER_ID	INTEGER	Y		0
SNAME	VARCHAR	Y	80	0
SLOCATION	VARCHAR	Y	300	0

8.TESTING

8.1 TEST CASES

Testing can be verification and validation or reliability estimation. The primary objective if testing includes:

- To identify defects in the application.
- The most important role of testing is simply to provide information.
- To check the proper working of the application while inserting updating and deleting the entry of the products.

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	TC for Automation(Y/N)	BUG ID	Executed By
HomePage_TC_001	Functional	Home page	Verify user is able to move to the Register page to create a new account	-	1. Enter URL and click go 2. Click on the Register button / Click on the Register link in navbar	http://127.0.0.1	User should be navigated to the Register page	Working as expected	Pass				Ashwin Menon
HomePage_TC_002	Functional	Home page	Verify user is able to move to the Login page to create a new account	-	1. Enter URL and click go 2. Click on the Login button / Click on the Login link in navbar	http://127.0.0.1	User should be navigated to the Login page	Working as expected	Pass				Ashwin Menon
HomePage_TC_003	UI	Home Page	Verify the UI elements in the home page	-	1.Enter URL and click go 2.Verify that below UI elements exist: a.Register button b.Login button	http://127.0.0.1	Page should show below UI elements: a. A blue Register button b. A green Login button	All mentioned elements exist	Pass				Ashwin Menon
LoginPage_TC_001	Functional	Login page	Verify user is able to log into application with their correct credentials	Account must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password	http://127.0.0.1 Username: testusername Password: testpassword	Application should accept user credentials and user should be navigated to the dashboard	Working as expected	Pass				Rahul Reddy
LoginPage_TC_002	Functional	Login page	Verify user is able to log into application with incorrect credentials	Incorrect account details must not exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password	http://127.0.0.1 Username: testuser Password: testpass	Application should show 'Incorrect email or password ' validation message.	Working as expected	Pass				Rahul Reddy
LoginPage_TC_003	UI	Login page	Verify the UI elements in the login page	-	1. Enter URL and click go 2. Click on the Login button 3. Verify that below UI elements exist: a. Username field b. Password field c. Login button d. 'Signup now' link	http://127.0.0.1	Page should show below UI elements: a. Username field b. Password field c. Login button d. 'Signup now' link	All mentioned elements exist	Pass				Vignesh Kumar

RegisterPage_TC_001	Functional	Register page	Verify user is able to register to create a new account and get redirected to the login page	Account must not already exist	1. Enter URL and click go 2. Click on the Register button / Register link in navbar 3. Enter Name, Email, Username, Password, and Confirm Password	http://127.0.0.1 Name: testname Email: test@gmail.com Username: testusername Password: testpassword	New account should get created and user should be redirected to the login page	Working as expected	Pass				Vignesh Kumar
RegisterPage_TC_002	UI	Register page	Verify the UI elements in the login page	-	1. Enter URL and click go 2. Click on the Register button 3. Verify that below UI elements exist: a. Name field b. Email field c. Username button d. Password button e. Confirm Password button	http://127.0.0.1	Page should show below UI elements: a. Name field b. Email field c. Username button d. Password button e. Confirm Password button f. Submit button	All mentioned elements exist	Pass				Vignesh Kumar
DashboardPage_TC_001	Functional	Dashboard page	Verify if the user can update stock, add new stock, and remove stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Stock Updation: a Enter item name, select a field and enter value. b. Click on 'Update' 5. New Stock Addition: a. Enter item name, quantity, and price	http://127.0.0.1 Item name: testitemname value:0 quantity:100 price:100	Stock gets updated, new stock gets added, and stock gets removed	Working as expected	Pass				Krishna Teja
DashboardPage_TC_002	Functional	Dashboard page	Verify if the user cannot update non-existent stock, add already existing stock, or remove non-existent stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Stock Updation: a Enter invalid item name, select a field and enter value. b. Click on 'Update' 5. New Stock Addition:	http://127.0.0.1 Item name: testitemname1 value:0 quantity:100 price:100	Respective error messages get shown	Working as expected	Pass				Krishna Teja
DashboardPage_TC_003	UI	Dashboard page	Verify if the user is able to see a table of products along with functionality for updating, adding, and removing stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Dashboard page is displayed along with UI elements	http://127.0.0.1	Page should show below UI elements: a. A table of products b. Three text fields and 'Update' button under Update Stock c. Three text fields and 'Add Stock' button under Add New Stock	All mentioned elements exist	Pass				Krishna Teja
ProfilePage_TC_001	Functional	Profile page	Verify if the user can update their details and password	-	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Profile in navbar 5. Update user details and password	http://127.0.0.1 Username: testusername Password: testpassword	User details and password should get updated	Working as expected	Pass				Ashwin Menon
ProfilePage_TC_002	UI	Profile page	Verify if the user is able to see their current details, and functionality to update thier details and password	-	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Profile in navbar	http://127.0.0.1	Page should show below UI elements: a. Current user details with username, name, and email b. Two text fields and 'Update' button under 'Update user details' c. Three text fields and 'Update'	All mentioned elements exist	Pass				Ashwin Menon
SuppliersPage_TC_001	Functional	Suppliers page	Verify if the user can update supplier, add new supplier, and delete supplier	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar 5. Supplier Updation: a Enter name, select a field and enter value. b. Click on 'Update'	http://127.0.0.1 Supplier name: testsupplier location: abc Value: 100	Supplier details get updated, new supplier gets added, and a supplier is deleted	Working as expected	Pass				Rahul Reddy
SuppliersPage_TC_002	Functional	Suppliers page	Verify if the user cannot update non-existent supplier, add already existing supplier, or remove non-existent supplier	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar 5. Supplier Updation: a Enter invalid name, select a field and enter value. b. Click on 'Update' 6. New Supplier Addition:	http://127.0.0.1 Supplier name: testsupplier1 location: abc Value: 100	Respective error messages get shown	Working as expected	Pass				Rahul Reddy
SuppliersPage_TC_003	UI	Suppliers page	Verify if the user is able to see a table of suppliers along with functionality for updating, adding, and deleting suppliers	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar	http://127.0.0.1	Page should show below UI elements: a. A table of suppliers b. Two text fields, a dropdown, and 'Update' button under Update Supplier c. Two text fields, a dropdown, and 'Add Supplier' button under	All mentioned elements exist	Pass				Rahul Reddy
OrdersPage_TC_001	Functional	Orders page	Verify if the user can create a new order, update an order, and cancel an order	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar 5. Order Creation: a Enter Stock ID, quantity b. Click on 'Create' 6. Order Updation:	http://127.0.0.1 Stock ID: 12345 Order ID: 123 Quantity: 100 Value: 100	New order gets created, an order gets updated, and an order gets cancelled	Working as expected	Pass				Krishna Teja
OrdersPage_TC_002	Functional	Orders page	Verify if the user cannot update a non-existent order, or cancel a non-existent order	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar 5. Order Updation: a. Enter invalid Order ID, choose a field, and enter value b. Click on 'Update'	http://127.0.0.1 Stock ID: 12346 Order ID: 124 Quantity: 100 Value: 100	Respective error messages get shown	Working as expected	Pass				Vignesh Kumar
OrdersPage_TC_003	UI	Orders page	Verify if the user is able to see a table of orders along with functionality for creating, updating, and cancelling orders	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar	http://127.0.0.1	Page should show below UI elements: a. A table of orders b. Two text fields and 'Create' button under Create Order c. Two text fields, a dropdown, and 'Update' button under Update Order	All mentioned elements exist	Pass				Krishna Teja

8.2 USER ACCEPTANCE TESTING

Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	3	2	2	1	8
Duplicate	0	1	2	1	4
External	1	3	2	1	7
Fixed	4	2	3	15	24
Not Reproduced	0	0	1	1	2
Skipped	1	0	1	1	3
Won't Fix	2	3	2	1	8
Totals	11	11	13	21	56

Test Case Analysis

Section	Total Cases	Not Tested	Fail	Pass
Login	8	0	0	8
Dashboard	19	0	0	19
Db2 Database	9	0	0	9
Flask Application	4	0	0	4

9.RESULTS

9.1 PERFORMANCE METRICS

NFT - Risk Assessment

S.No	Project Name	Scope/feature	Functional Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Volume Changes	Risk Score	Justification
1	Login Authentication	New	Moderate		Moderate		>30 to 50 %	ORANGE	Required feature
2	Transaction Management	New	High		Moderate		>30 to 50 %	RED	Indispensable feature
3	Containerization	New	Low		Moderate		>5 to 10%	ORANGE	Feature to make it deployable

NFT - Detailed Test Plan

S.No	Project Overview	NFT Test approach	Assumptions/Dependencies/Risks	Approvals/SignOff
1	INVENTORY MANAGEMENT SYSTEM FOR RETAILERS	Stress Test	Proper internet Connection User Credentials	Approved

End Of Test Report

S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/Closed/Open)	Approvals/SignOff

10.ADVANTAGES & DISADVANTAGES

10.1 Advantages

- Manage multiple warehouses.
- Reduce business cost.
- Greater productivity.
- Improve supply chain.
- Reduce cost overselling.

10.2 Disadvantages

- This application is not suitable for those organizations where there is a large quantity of product and different levels of warehouses.
- This software application is able to generate only simple reports.
- Single admin panel is only made. It is not suitable for large organizations.

11. CONCLUSION

In conclusion, the Inventory Management System for merchants is a straightforward web application ideal for SMEs. It has every component a fundamental inventory management system needs to function, and enterprises employ it. Our team has been successful in creating an application that allows us to update, insert, and delete items as needed. Our staff is adamant that, despite some restrictions, the adoption of this system will undoubtedly be advantageous to enterprises.

12.FUTURE SCOPE

A system for keeping track of inventory changes, valuing items, and planning for future inventory levels are some of the additional uses for an inventory system. The inventory value at the end of each period serves as the basis for the financial reporting on the balance sheet. By evaluating the change in inventory, the company can determine the cost of items sold during the period: As a result, the company can get ready for upcoming inventory needs.

13.APPENDIX

13.1 SOURCE CODE

app.py:

```
      from flask import Flask, render_template, flash, redirect, url_for,
session, request, logging
      from wtforms import Form, StringField, TextAreaField, PasswordField,
validators, SelectField, IntegerField
      import ibm_db
      from functools import wraps
      from datetime import datetime, timedelta
      import sendgrid
      import os
      from sendgrid.helpers.mail import Mail, Email, To, Content
      from dotenv import load_dotenv
      load_dotenv()



      app = Flask(__name__)
      app.secret_key = 'ceg1234'
      dsn_hostname = "98538591-7217-4024-b027-
8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud"
      dsn_uid = "hgd72603"
      dsn_pwd = "qUt0VtWTanLWm4bJ"

      dsn_driver = "{IBM DB2 ODBC DRIVER}"
      dsn_database = "bludb"
      dsn_port = "30875"
      dsn_protocol = "TCPIP"
      dsn_security = "SSL"

      dsn = (
          "DRIVER={0};"
          "DATABASE={1};"
          "HOSTNAME={2};"
          "PORT={3};"
          "PROTOCOL={4};"
```

```
          "UID={5};"
          "PWD={6};"
          "SECURITY={7};"

"SSLServerCerificate=DigiCertGlobalRootCA.crt").format(dsn_driver,
dsn_database, dsn_hostname, dsn_port, dsn_protocol, dsn_uid,
dsn_pwd,dsn_security)

      try:
          conn = ibm_db.connect(dsn,"","")
      except:
          print("Unable to connect: ", ibm_db.conn_error())

      SUBJECT = "IBM Inventory Management Application"

      def sendgridmail(user,TEXT):
          try:
              print("Helo")
              content = Content("text/plain",TEXT)
              message = Mail(
                  from_email=os.environ.get('SENDGRID_FROM_EMAIL'),
                  to_emails=user,
                  subject=SUBJECT,
                  html_content=content)
              print("Hello1")
              sg =
sendgrid.SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
              print("Hello2")
              response = sg.send(message)
              print("Hello3")
              print(response.status_code)
              print(response.body)
              print(response.headers)

          except Exception as e:
              print("Hello4")
              print(e)


      @app.route('/')
```

```
    def index():
        return render_template('home.html')

    #Register Form Class
    class RegisterForm(Form):
        name = StringField('Name', [validators.Length(min=1, max=50)])
        username = StringField('Username', [validators.Length(min=1,
max=25)])
        email = StringField('Email', [validators.length(min=6, max=50)])
        password = PasswordField('Password', [
            validators.DataRequired(),
            validators.EqualTo('confirm', message='Passwords do not
match')
        ])
        confirm = PasswordField('Confirm Password')

    #Register new User
    @app.route('/register', methods=['GET','POST'])
    def register():
        form = RegisterForm(request.form)
        if request.method == 'POST' and form.validate():
            email = form.email.data
            username = form.username.data
            password = str(form.password.data)

            sql = "SELECT * FROM users WHERE email=?"
            prep_stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(prep_stmt, 1, email)
            ibm_db.execute(prep_stmt)
            account = ibm_db.fetch_assoc(prep_stmt)
            print(account)
            if account:
                error = "Account already exists! Log in to continue !"
            else:
                insert_sql = "INSERT INTO users
(email,username,password) values(?,?,?)"
                prep_stmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(prep_stmt, 1, email)
                ibm_db.bind_param(prep_stmt, 2, username)
                ibm_db.bind_param(prep_stmt, 3, password)
```

```
                ibm_db.execute(prep_stmt)
                sendgridmail(email, "Registered Successfully! Thank you
for registering with us")
                flash(" Registration successful. Log in to continue !")

            return redirect(url_for('login'))
        return render_template('register.html', form = form)

    #User Login
    @app.route('/login', methods = ['GET', 'POST'])
    def login():
        if request.method == 'GET':
            return render_template('login.html')
        else:
            error = None
            account = None
            username = request.form['username']
            password = request.form['password']
            print(username, password)

            sql = "SELECT * FROM users WHERE username=? AND password=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, password)
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            print(account)
        if account:
            session['logged_in'] = True
            session['username'] = username
            flash("Logged in successfully","success")
            return redirect(url_for('dashboard'))
        else:
            error = "Incorrect username / password"
            return render_template('login.html', error=error)


    #Check for if user is logged in
    def is_logged_in(f):
        @wraps(f)
```

```
        def wrap(*args, **kwargs):
            if 'logged_in' in session:
                return f(*args, **kwargs)
            else:
                flash('Unauthorized, Please login', 'danger')
                return redirect(url_for('login'))
        return wrap

    #Main Dashboard Page
    @app.route('/dashboard')
    @is_logged_in
    def dashboard():
        sql = "SELECT * FROM stock"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_assoc(stmt)
        stocks = []
        print(dictionary)
        headings = [*dictionary]
        while dictionary != False:
            stocks.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)
        return render_template('dashboard.html',headings=headings,
data=stocks)

    #User Logout
    @app.route('/logout')
    @is_logged_in
    def logout():
        session.clear()
        flash("Logged out successfully", "success")
        return redirect(url_for('login'))

    #Update Stock Inventory
    @app.route('/inventoryUpdate', methods=['POST'])
    @is_logged_in
    def inventoryUpdate():
        if request.method == "POST":
            try:
                item = request.form['item']
                field = request.form['input-field']
```

```
                value = request.form['input-value']
                print(item, field, value)
                insert_sql = 'UPDATE stock SET ' + field + "= ?" + "
WHERE STOCK_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 STOCK_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
STOCK_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:

                return redirect(url_for('dashboard'))

    # Add to Stock Inventory
    @app.route('/addstocks', methods=['POST'])
    @is_logged_in
    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 stock
(STOCK_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:

                return redirect(url_for('dashboard'))

    #Delete from Stock Inventory
    @app.route('/deletestocks', methods=['POST'])
    @is_logged_in
    def deleteStocks():
        if request.method == "POST":
            print(request.form['item'])
            try:
                item = request.form['item']
                insert_sql = 'DELETE FROM stock WHERE STOCK_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('dashboard'))

    #Update Username
    @app.route('/update-user', methods=['POST', 'GET'])
    @is_logged_in
```

```
    def updateUser():
        if request.method == "POST":
            try:
                email = session['username']
                value = request.form['input-value']
                insert_sql = 'UPDATE users SET username= ? WHERE
username=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, value)
                ibm_db.bind_param(pstmt, 2, email)
                print(pstmt)
                ibm_db.execute(pstmt)
            except Exception as e:
                print(e)
                msg = e

            finally:
                session['username'] = value
                return redirect(url_for('profile'))

    #Update Password
    @app.route('/update-password', methods=['POST', 'GET'])
    @is_logged_in
    def updatePassword():
        if request.method == "POST":
            try:
                email = session['username']
                password = request.form['prev-password']
                curPassword = request.form['cur-password']
                confirmPassword = request.form['confirm-password']
                insert_sql = 'SELECT * FROM  users WHERE username=? 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
username=?'
                    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:

                    return redirect(url_for('profile'))

    #Get Orders
    @app.route('/orders', methods=['POST', 'GET'])
    @is_logged_in
    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)

    #Create new Order
    @app.route('/createOrder', methods=['POST'])
    @is_logged_in
    def createOrder():
        if request.method == "POST":
            try:
                stock_id = request.form['stock_id']
                query = 'SELECT PRICE_PER_QUANTITY FROM stock WHERE SID=
?'
                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
(STOCK_ID,QUANTITY,ODATE,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'))

    #Update Order
    @app.route('/updateOrder', methods=['POST'])
    @is_logged_in
    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
OID=?"
                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'))

    #Cancel Order
    @app.route('/cancelOrder', methods=['POST'])
    @is_logged_in
    def cancelOrder():
        if request.method == "POST":
            try:
                order_id = request.form['order_id']
                query = 'DELETE FROM orders WHERE OID=?'
                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'))

    #Get Suppliers
    @app.route('/suppliers', methods=['POST', 'GET'])
    @is_logged_in
    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)

        sql = "SELECT OID FROM orders"
        stmt = ibm_db.exec_immediate(conn, sql)
```

```
        dictionary = ibm_db.fetch_assoc(stmt)
        order_ids = []
        print("dictionary")
        print(dictionary)
        while dictionary != False:
            order_ids.append(dictionary['OID'])
            dictionary = ibm_db.fetch_assoc(stmt)
        unassigned_order_ids=None
        return render_template("suppliers.html", headings=headings,
data=suppliers, order_ids=order_ids)

    #Update Supplier
    @app.route('/updatesupplier', methods=['POST'])
    @is_logged_in
    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 SNAME=?"
                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'))

    # Add new Supplier
    @app.route('/addsupplier', methods=['POST'])
    @is_logged_in
    def addSupplier():
        if request.method == "POST":
            try:
```

```
                name = request.form['name']
                order_id = request.form.get('order-id-select')
                print(order_id)
                location = request.form['location']
                insert_sql = 'INSERT INTO suppliers
(SNAME,ORDER_ID,SLOCATION) 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'))

    #Delete Supplier
    @app.route('/deletesupplier', methods=['POST'])
    @is_logged_in
    def deleteSupplier():
        if request.method == "POST":
            try:
                item = request.form['name']
                insert_sql = 'DELETE FROM suppliers WHERE SNAME=?'
                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'))

    #Get User's Profile
    @app.route('/profile', methods=['POST', 'GET'])
    @is_logged_in
    def profile():
        if request.method == "GET":
```

```
            email = session['username']
            insert_sql = 'SELECT * FROM users WHERE username=?'
            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)
            return render_template("profile.html", data=dictionary)

if __name__ == '__main__':

    app.run(host="0.0.0.0",port=5000)
```

13.2 GITHUB & PROJECT DEMO LINK

GITHUB Link- https://github.com/IBM-EPBL/IBM-Project-29277-1660123060/tree/main

Application Demo Link - https://drive.google.com/file/d/1RLDUj-UFdYttbHJBOiBUwEUz7Ntt5RLq/view?usp=sharing

Live Application Deployed on Kubernetes Link - http://169.51.204.209:31617/