

**INVENTORY MANAGMENT SYSTEM FOR  
RETAILERS**

**PNT2022TMID34001**

**NALAIYA THIRAN-PROJECT BASED LEARNING ON  
PROFESSIONAL READLINESS FOR INNOVATION,  
EMPLOYNMENT AND ENTERPRENEURSHIP**

**A PROJECT REPORT**

**BY**

Gishalin Rufina(960219104043)

Alice(960219104011)

Arshitha Sherin(960219104022)

Hanan(960219104045)

# TABLE OF CONTENTS

1. **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 & PROPOSED SOLUTION**
  - 3.1 Empathy Map Canvas
  - 3.2 Ideation & Brainstorming
  - 3.3 Proposed Solution
  - 3.4 Problem Solution fit
4. **REQUIREMENT ANALYSIS**
  - 4.1 Functional requirement
  - 4.2 Non-Functional requirements
5. **PROJECT DESIGN**
  - 5.1 Data Flow Diagrams
  - 5.2 Solution & 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 & SOLUTIONING**
  - 7.1 Feature 1
    - 7.1.1. Introduction to Flask
    - 7.1.2.IBM Cloud Object Storage
    - 7.1.3.Coding
  - 7.2 Feature 2
    - 7.2.1.IBM Watson Assistant Service
    - 7.2.2.Docker
  - 7.3 Database Schema
8. **TESTING**
  - 8.1 Test Cases
  - 8.2 User Acceptance Testing
9. **RESULTS**
  - 9.1 Performance Metrics

## **10. ADVANTAGES & DISADVANTAGES**

10.1 Advantages

10.2 Disadvantages

## **11. CONCLUSION**

## **12. FUTURE SCOPE**

## **13. APPENDIX**

Source Code

GitHub & Project Demo Link

# **1. INTRODUCTION**

## **1.1 Project Overview**

Nowadays, organizations, and especially those performing activities in the retail sector, face multiple challenges in the planning and management of their resources. For this sector, having efficient management of human, technological, or material resources refers to the performance that companies characterized by the experience gained in their management could obtain over time. Therefore, the correct inventory management has become essential, especially in organizations dedicated to retail. The determination of the optimal inventory level is a fundamental part of the life of organizations due to the high investment that it represents at the time of its acquisition, administration, and maintenance. According to research, “the role of inventory management is to ensure that stocks of raw material or other supplies, i.e., work-in-progress and finished goods, are kept at levels that provide maximum service levels at minimum costs”. This because the realizable asset occupies a significant percentage within the Total Assets. Hence, its correct ordering and administration imply being able to minimize the risk of contracting results that may put the health of the company at risk. Various technologies have been developed over time for inventory management, going from basic manual reporting to an integrated information system (IS), which can help to “decide how and where orders should be fulfilled to improve service levels while decreasing total costs”. Moreover, these new functionalities can collaborate in the most effective handling of materials and better manage the cycle of purchase - reception - allocation in production

## **1.2.Purpose**

Inventory management being the practice of planning the buying, storing and selling of stock—whether it’s raw materials, parts or finished goods—to ensure the right type and amount of stock is available without holding excess stock and thereby tying up cash. When done correctly, inventory management saves companies money.

The primary purpose of inventory management is to ensure there is enough goods or materials to meet demand without creating overstock, or excess inventory. Accurate inventory management is key to running a successful product business. Tracking stock regularly can help avoid stock errors and other problems.

## 2.LITERATURE SURVEY

### 2.1. Existing problem

When the inventory becomes hard to find, it leads to inventory visibility problems. Lack of visibility is one of the most common inventory management problems. Locating the correct item in the right place as quickly as possible is essential to inventory. If the hard-to-find inventory is part of the supply chain for manufacturing, it can impact the operations of the entire manufacturing process. If the inventory stock is being accessed for shipping and cannot be located, it leads to incomplete or wrong shipments and severely impacts customer satisfaction. Either way inventory visibility problems have a severe impact on the performance of the business and is one of the symptoms of poor inventory management. Another problem is with money that is spent on inventory gets locked in if the items are not used. Overstocking can impact the profitability of a business. This is because more stock is bought than being sold. At the same time, being understocked is also one of the inventory management problems. Understocking can slow down production or even bring it to a halt. Not utilising the available warehouse space is also money wasted. Improper inventory management does not make the best use of all the available warehousing space that the company is paying for or bearing overheads on. So, to overcome these, the inventory management system is being introduced.

### 2.2. References

Sl.no	Title and Author	Year and Publication	Methodology	Advantage	Drawback
1.	Inventory management for retail companies  Rodrigo Arcentales-Carrion, Mario Pena	March, 2021  ResearchGate Publications	Bar code and Radio Frequency Identification, RFID stock counts, Bayesian estimation method, Warehouse Management System (WMS)	Easy to analyze their KPIs, All the optimization algorithms and methodologies mention the importance of having an integrated information system.	High cost of implementation and maintenance

2.	<p>Research paper on Inventory management system</p> <p>Punam Khobragade, Roshni Selokar, Rina Maraskolhe, Prof. Manjusha Talmale</p>	<p>April 2018</p> <p>International Research Journal of Engineering and Technology (IRJET)</p>	<p>Efficient and purposive examination of actualities with a goal deciding the powerful relationship, Poll study among development experts to distinguish their feeling towards stock administration framework in their association.</p>	<p>A straightforward and secure desktop application, displays the one table organization look</p>	<p>Impersonal touch, Production problem.</p>
----	---	---	--	---	--

3.	<p>Sales and Inventory Management System</p> <p>Rahmat Bee Abdul Aleem</p>	<p>September 2013</p> <p>Business Information System University Technology Of Petronas</p>	<p>Rapid application development (rad) methodology, phased development-based methodologies</p>	<p>Increases profitability, improves cash flow, improves decision-making, increases customer satisfaction</p>	<p>Increased space, High implementation costs, business risk.</p>
4.	<p>Inventory Management System</p> <p>Rishabh gupta, Ashish, Aman yadav</p>	<p>April 2022</p> <p>International Journal of Creative Research Thoughts (IJCRT</p>	<p>MERN stack technologies, MERN stands for MongoDB, Express, React,Node.</p>	<p>Easily organizable, simple and easy to use, customer satisfaction</p>	<p>Obsolete Inventory, Storage Costs</p>

5.	Reconstructing inventory management theory  Geoff Buxey	2006  Emerald Group Publishing Limited	Published examples are described in sufficient detail to reveal what these firms actually do, an alternative “top down” approach is proposed.	Low Risk of Shortages, Wholesale Pricing, Fast Fulfillment.	Potential Insurance Costs and Loss, Tying Up Capital
----	---	--	---	---	--

### 2.3. Problem Statement Definition

Creating a problem statement to understand our customer's point of view. The Customer Problem Statement template helps us focus on what matters to create experiences people will love.

A well-articulated customer problem statement allows us to find the ideal solution for the challenges our customers face. Throughout the process, we also will be able to empathize with our customers, which helps get better understanding on how they perceive the product or service.

<b>Problem Statement (PS)</b>	<b>I am (Customer)</b>	<b>I'm trying to</b>	<b>But</b>	<b>Because</b>	<b>Which makes me feel</b>
PS-1	Supplier	To make on-time delivery possible	I'm unable to supply the products demanded	Of the inaccurate data and limited visibility	Daunty
PS-2	Retailer	meet customer demand without running out of stock or carrying excess supply	It seems impossible to keep track of the customer-demanded products	Of lack of a proper software	frustrated
PS-3	Shopper	To purchase goods	The products which I prefer has run out-of-stock	Of unawareness of the retailers	Annoyed

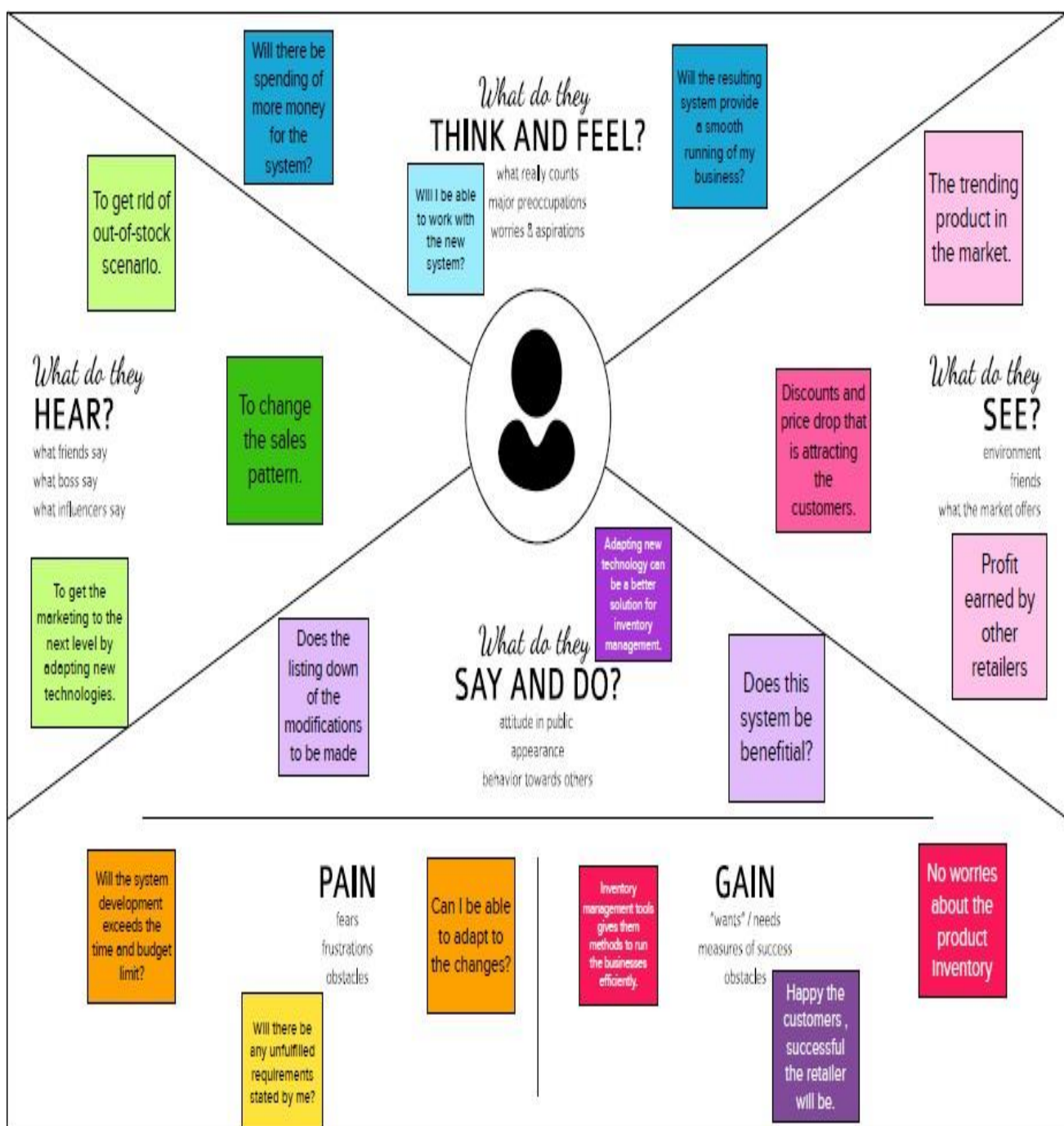


### 3. IDEATION AND PROPOSED SOLUTION

#### 3.1. Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.


It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.



### 3.2. Ideation And Brainstorming

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

Template



## Brainstorm & idea prioritization

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

🕒 10 minutes to prepare

🕒 1 hour to collaborate

👤 2-8 people recommended

➔

### 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 manage inventory, so that the retailers meet customer demand without running out of stock or carrying excess supply?



### Key rules of brainstorming

To run a 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!

Gishalin Rufina

Identifying the products that are on-demand in the market

Easy to operate and manageable software tool to help retailers with inventory management.

Just-in-time inventory management technique can be implemented

Automatic facilities in various aspects like managing the order, delivery, and keeping track of the shipment.

Online mode of placing orders to the suppliers and online mode of payment made possible saving time and energy.

Getting an alert when a product is about to run out-of-stock

Prioritize the Inventory

Use technology that integrates well.

Track all product information.

Allce

Track sales.

Order restocks yourself.

Audit the inventory

Prioritize with ABC analysis.

Sell older Inventory first.

Verify accuracy with regular counts.

Arshitha Sherin

Always track your metrics

Give each variant a dedicated warehouse bin

Use EOQ for optimal order quantities.

Master the lead times.

Use consignment Inventory.

Calculate reorder points.

Hanan

Implement a Just In Time (JIT) inventory system.

Automate tasks with Inventory management software.

Use accurate demand forecasting.

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

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize and categorize important ideas as themes within your mind.

### Techniques

Just-in-time inventory management technique can be implemented

Use technology that integrates well.

Use EOQ for optimal order quantities.

Implement a Just In Time (JIT) inventory system.

Give each variant a dedicated warehouse bin

Use consignment inventory.

### Prioritize goods

Master the lead times.

Identifying the products that are on-demand in the market

Prioritize the inventory

Prioritize with ABC analysis.

Use accurate demand forecasting.

Sell older inventory first.

### Automated features

Getting an alert when a product is about to run out-of-stock

Always track your metrics

Track all product information.

Automate tasks with inventory management software.

Automatic facilities in various aspects like managing the order, delivery, and keeping track of the shipment.

Verify accuracy with regular counts.

### Benefits

Easy to operate and manageable software tool to help retailers with inventory management.

Audit the inventory

Online mode of placing orders to the suppliers and online mode of payment made possible saving time and energy.

Order restocks yourself.

Calculate reorder points.

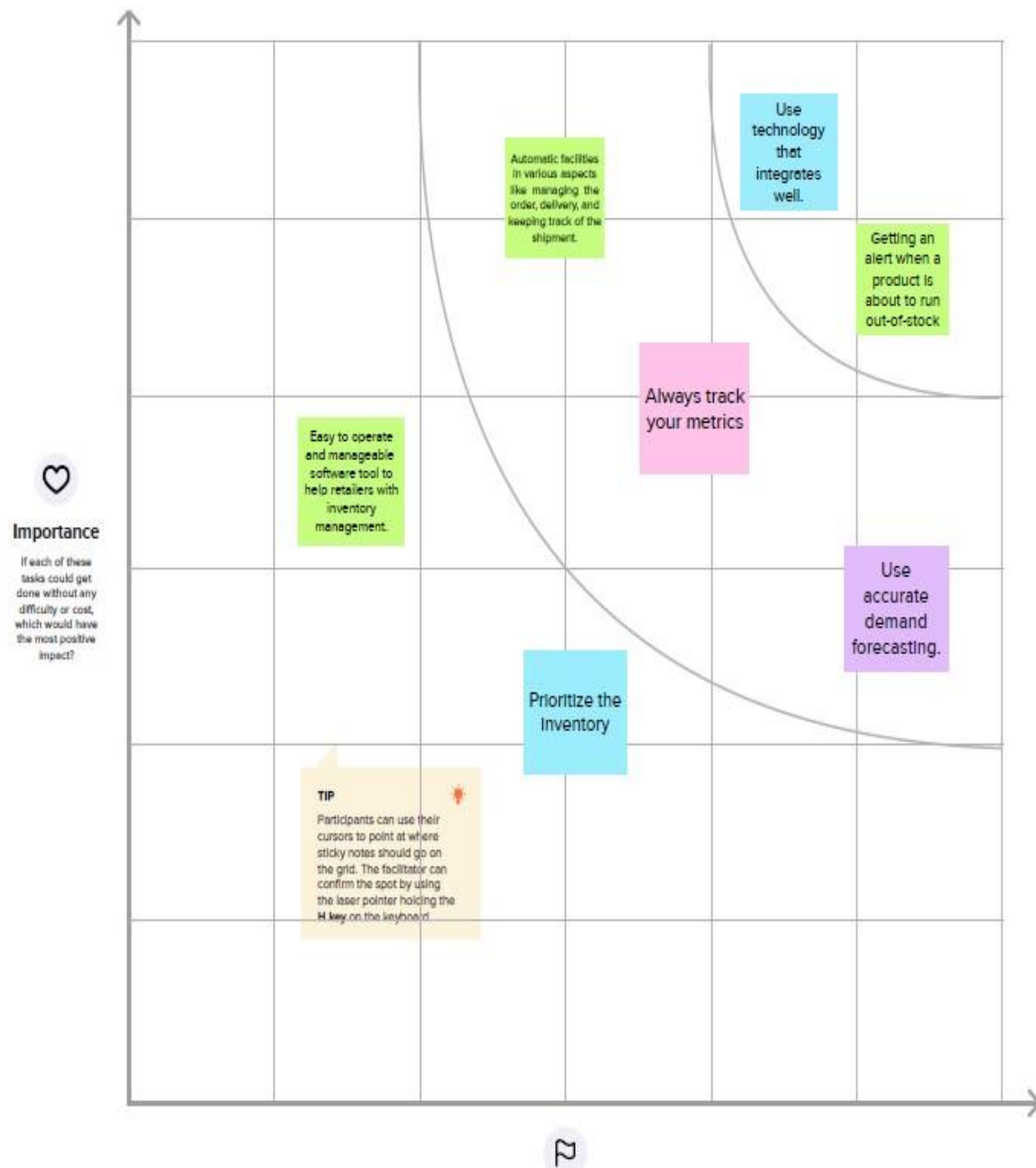
Track sales.

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)	<p>Irrespective of the size of the business, inventory management is one of the most challenging processes in the retail sector. In this industry, the efficiency of inventory management directly impacts customer satisfaction. As retail is a fast-paced, and customer-facing sector, customer satisfaction is core to its business growth. The inventory process involves multiple intricate aspects that drive accurate product delivery. Even a single error in the process can have expensive and long-term consequences. This will eventually affect the company's growth and reputation. The inventory issue refers to the general issue of deciding how much inventory to keep on hand in expectation of possible demand. Loss occurs when a business is unable to meet demand or when commodities are stocked for which there is no demand.</p>
2.	Idea / Solution description	<p>So to provide a solution to this problem of retailers, an inventory management system with easy to operate and access mechanism can be used to track the inventory of a single store or to manage the delivery of stock between several branches of a larger franchise. However, the system merely records sales and restocking data and provides warning of low stock at any location through email at a specified interval.</p>
3.	Novelty / Uniqueness	<ul style="list-style-type: none"><li>➤ The right products at the right time.</li><li>➤ To set automatic reorder points based on preset stock levels and current availability to avoid overselling.</li></ul>

		<ul style="list-style-type: none"> <li>➤ To use multi-location warehouse management features to track and control expanding inventories.</li> <li>➤ To use inventory control processes like blind receiving with barcodes and mobile scanners to prevent human error, inventory manipulation and shrinkage due to theft or negligence.</li> <li>➤ To introduce dashboards with simple interfaces that show real-time inventory data.</li> <li>➤ To use RFID reader and manage products with the RIOT application to generate accurate data.</li> </ul>
4.	Social Impact / Customer Satisfaction	<ul style="list-style-type: none"> <li>➤ When product is returned because it is damaged or dead on arrival, and it is still under warranty, you can arrange with the manufacturer to do an instant swap of the product to keep the customer happy.</li> <li>➤ To reduce the amount of time that products sit on your shelves. When you don't carry extra inventory for extended periods of time, your inventory costs decrease. This is a savings that you can pass on to clients in the form of lower pricing.</li> <li>➤ By being able to give clients accurate inventory information, you improve the image of your company and add one more element to customer retention.</li> <li>➤ To have popular items in stock and ready to instantly fulfill any customer's orders</li> <li>➤ Use inventory management systems with warehouse management features to optimize storage space and inventory flow.</li> </ul>

5.	Business Model (Revenue Model)	<ul style="list-style-type: none"> <li>➤ It helps companies to identify which and how much stock to order at what time.</li> <li>➤ It tracks inventory from purchase to sale of goods.</li> <li>➤ It is suited for situations where a business is expecting to grow.</li> <li>➤ It starts with adopting the right inventory technology to allow for more effective supply chain management, cutting down on costs, reducing waste and reducing the overall carbon footprint.</li> </ul>
6.	Scalability of the Solution	<ul style="list-style-type: none"> <li>➤ Give employees the right inventory tools for the job. They need software to replace manual inventory documentation, and paperless transactions for invoices and purchase orders.</li> <li>➤ Add images with product descriptions in your inventory database to improve purchasing and receiving processes, enhance accuracy and prevent misplaced inventory.</li> <li>➤ Inventory management ensures control over customers demands thereby resulting to customer satisfaction and increase financial performance.</li> <li>➤ Thus the proposed system provides to be a user friendly and makes it cheaply available</li> </ul>

### 3.4. Problem Solution Fit

<b>1. CUSTOMER SEGMENT(S)</b> Inventory Management software can help distributors, whole salers, manufacturers and retailers to optimize their warehouses.	<b>6. CUSTOMER CONSTRAINTS</b> Limits on raw materials, machine capacity, workflow capacity, inventory investment, storage space or the total number of orders placed.	<b>5. AVAILABLE SOLUTIONS</b> Track inventory across multiple locations, automatically manage reorder points, forecast demand and plan production and distribution.
---	---	--

<b>2. JOBS-TO-BE-DONE / PROBLEMS</b> Unclear communication, inadequate access, inefficient warehouse management, overselling, spoiled goods.	<b>9. PROBLEM ROOT CAUSE</b> As retail is a fast-paced and customer-facing sector, customer satisfaction is core to its business growth.	<b>7. BEHAVIOUR</b> From a customer's point of view, it helps them provide better customer services through fast delivery and low shipping charges hence meeting customer expectations.
---	---	--

<b>3. TRIGGERS</b> <span>TR</span> Inventory Management is vital for retailers because the practices helps them increase profits and efficiency.	<b>10. YOUR SOLUTION</b> <span>SL</span> We propose an inventory management system that ensures control over customer's demand without running out-of-stock or carrying excess supply.	<b>8. CHANNELS of BEHAVIOUR</b> <span>CH</span> ONLINE Using cloud based software where you can access all inventory-related information that can make business easier for you.
<b>4. EMOTIONS: BEFORE / AFTER</b> <span>EM</span> Without efficient inventory management, retailers can easily lose track of their stock levels. But after using our technique, it ensure there is enough goods or materials to meet demand without creating overstock or excess inventory.		OFFLINE The simplest way to track inventory is to manually count your inventory every two weeks and compare the numbers verses Sales which is periodic inventory.



## 4.REQUIRMENT ANALYSIS

### 4.1. Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User login	Login with Fingerprints Login with Password Login with Username
FR-4	Periodic stock checking	Cycle counting Physical counting
FR-5	Promotion	Maintain enough stock on hand to meet demand.
FR-6	Markdown	Show product discount
FR-7	Death stock	Return to the vendor for credits
FR-8	Returns Management System	1. Add it to inventory counts 2 Check for damage

### 4.2. Non-Functional Requirements:

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

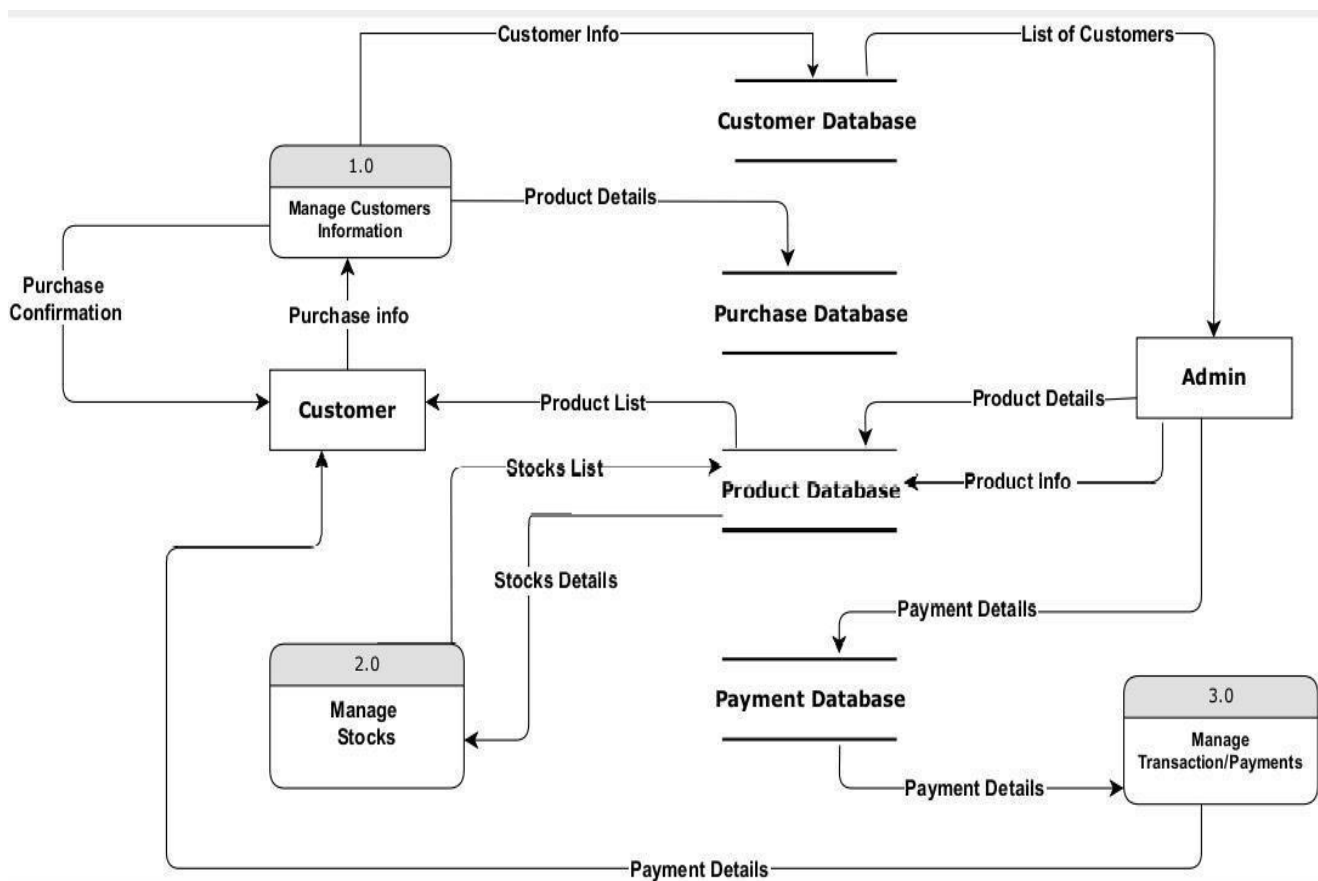
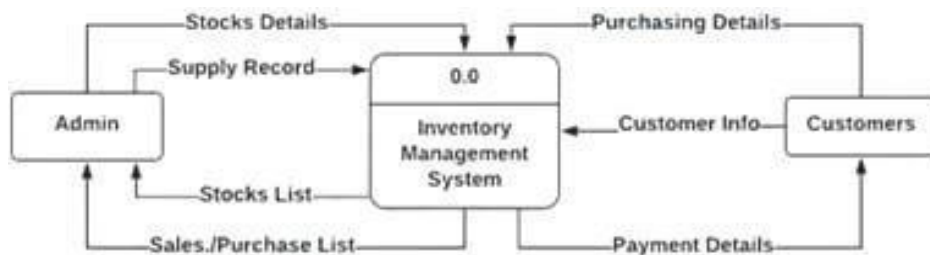
FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	The System must be intuitive and simple in the way It displays a relevant data and relationships; and the menus of the system are easily navigate by the users with buttons that are easy to understand If it takes hours for your staff to learn the software, then it's not worth it. You should remember to choose a solution that simplifies inventory management
NFR-2	<b>Security</b>	Only authorized users can access the system with urname and password of administrator. Inventory is the process of ensuring the safety and optimum management control of stored goods. It is of central importance for optimum warehouse management because the performance of a company stands or tells with the safety and efficiency of a warehouse.

NFR-3	<b>Reliability</b>	Important for several reasons. Your delivery reliability depends on it, but also consider the cost involved. The system must successfully add any recipe, ingredients, vendors, or special occasions given by the user and provide estimations and inventory status in relevance to the newly updated entities. The system must give accurate inventory status to the user continuously. Any inaccuracies are corrected by regularly comparing the actual levels to the levels displayed in the system.
NFR-4	<b>Performance</b>	The goal of inventory performance metrics is to compare actual on-hand dollars versus forecasted cost of goods sold. The system must successfully complete updating the databases, adding new recipes, ingredients, vendors, and occasions every time the user requests such a process. All the functions of the system must be available to the user every time the system is turned on. The calculations performed by the system must comply with the norms set by the user and should not vary unless implicitly changed by the user.
NFR-5	<b>Availability</b>	The software will be available only to administrators of the organization and the product as well as customer details will be recorded by him. Inventory availability refers to whether a specific item is available for customer orders. Additional information provided by retailers may include the quantity available.
NFR-6	<b>Scalability</b>	The inventory management software or app you choose be able to grow as your business does? The last thing you want is to have to manually re-enter all your inventory if you outgrow your current system. The ability of a system to handle growing amount of work.

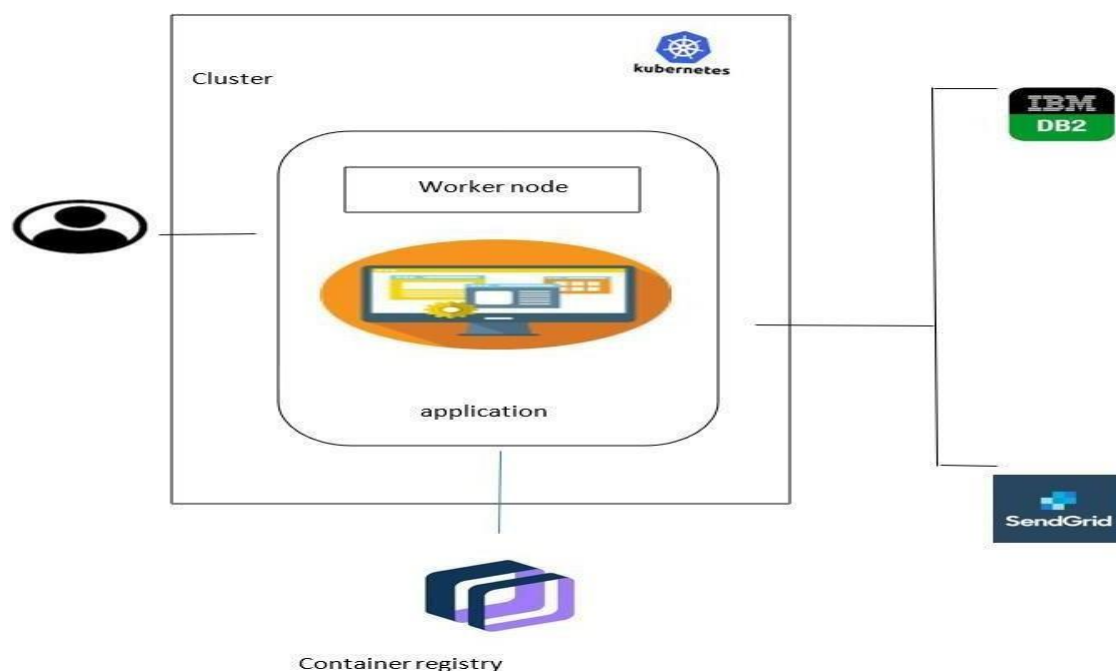
## 5. PROJECT DESIGN

### 5.1. Data Flow Diagram:

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



## 5.2. Solution and Technical Architecture:



S.N o	Component	Description	Technology
1.	User Interface	Web UI, Mobile App, Chatbot	HTML, CSS, avaScript , Jquery etc.
2.	Calculating Product Count	By entering barcode details into the application	Barcode Scanner
3.	Alert and notification	Alerting the retailers regarding the low stock countof theproduct	Send Grid
4.	Cloud Database	Database Service on cloud	IBM DB2
5.	Chat	Chat with Watson assistant	IBM Watson Assistant
6.	File Storage	File Storage Requirements	IBM Object Storage
7.	Infrastructure	Cloud Server Configuration	Cloud Foundry , Kubernetes

S.N o	Characteristics	Description	Technology
1.	Open-Source Frameworks	Styling our page, Python flask microframework	Python Flask, Bootstrap
2.	Security Implementations	For securing our cloud data	SSL Certificates
3.	Scalable Architecture	Three – tier architecture (MVC)	Technology used are, Web server - HTML, CSS, Java script Application server- Python Flask DockerDatabase Server – IBM DB2
4.	Availability	availability of application	Technology used is IBM Load Balancer
5.	Performance	2 requests per seconds, Use of Local Machine Cache Memory	Technology used is IBM Cloud , CDN

### 5.3. User Stories:

A user story is the smallest unit of work in an agile framework. It's an end goal, not a feature, expressed from the software user's perspective. The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register and access the dashboard with Gmail login	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can register and access the dashboard with email and password	High	Sprint-1
	Dashboard	USN-6	As a user, I can view the stock availability status	I can view the stock availability status	High	Sprint-2
		USN-7	As a user, I can view the orders status	I can view the order status	Medium	Sprint-3
		USN-8	As a user, I can view the shipping tracking status	I can view the shipping tracking status	Medium	Sprint-4
	Alerts	USN-9	As a user, I should receive alerts on stock availability if it drops below the set threshold	I should receive prompt alerts on stock availability if it drops below the set threshold	Medium	Sprint-4
Customer (Web user)	Registration	USN-10	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-11	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-12	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2

## 6. PROJECT PLANNING AND SCHEDULING

### 6.1. Sprint Planning and Estimation:

Here is how an agile team plans a new sprint, as part of an existing release plan:

1. Hold a retrospective meeting to discuss the previous sprints and lessons learned.
2. Run a sprint planning meeting to analyze the release plan and update it according to velocity in recent sprints, changes to priorities, new features, or idle time that wasn't planned for in the release.
3. Make sure user stories are detailed enough to work on. Elaborate on tasks that are not well defined, to avoid surprises.
4. Break down user stories into specific tasks. Keep the size of tasks small, no more than one workday.
5. Assign tasks to team members and confirm that they are committed to performing them. In the agile/scrum framework this is done by the Scrum Master.
6. Write the tasks on (physical) sticky cards and hang them up on a large board visible to the entire team. All the user stories in the current sprint should be up on the board.
7. Track the progress of all the tasks on a grid, by recording who is responsible for completing each task, estimated time to complete it, remaining hours, and actual hours used. This time tracking should be updated by all team members and visible to everyone.
8. Track velocity using a burndown chart. During the sprint, use the team's time tracking to calculate a chart showing the number of tasks or hours remaining, vs. the plan. The slope of the burndown chart shows if we are on schedule, ahead, or behind schedule.

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint -1	User Panel	USN-1	The user will login and view the services available. This also include registering for the application by entering the required details.	20	High	Gishalin Rufina, Alice, Arshitha Sherin, Hanan
Sprint -2	Admin Panel	USN-2	The role of the admin is to check out the database on the availability and have further maintenance required on the system.	20	High	Gishalin Rufina, Alice, Arshitha Sherin, Hanan

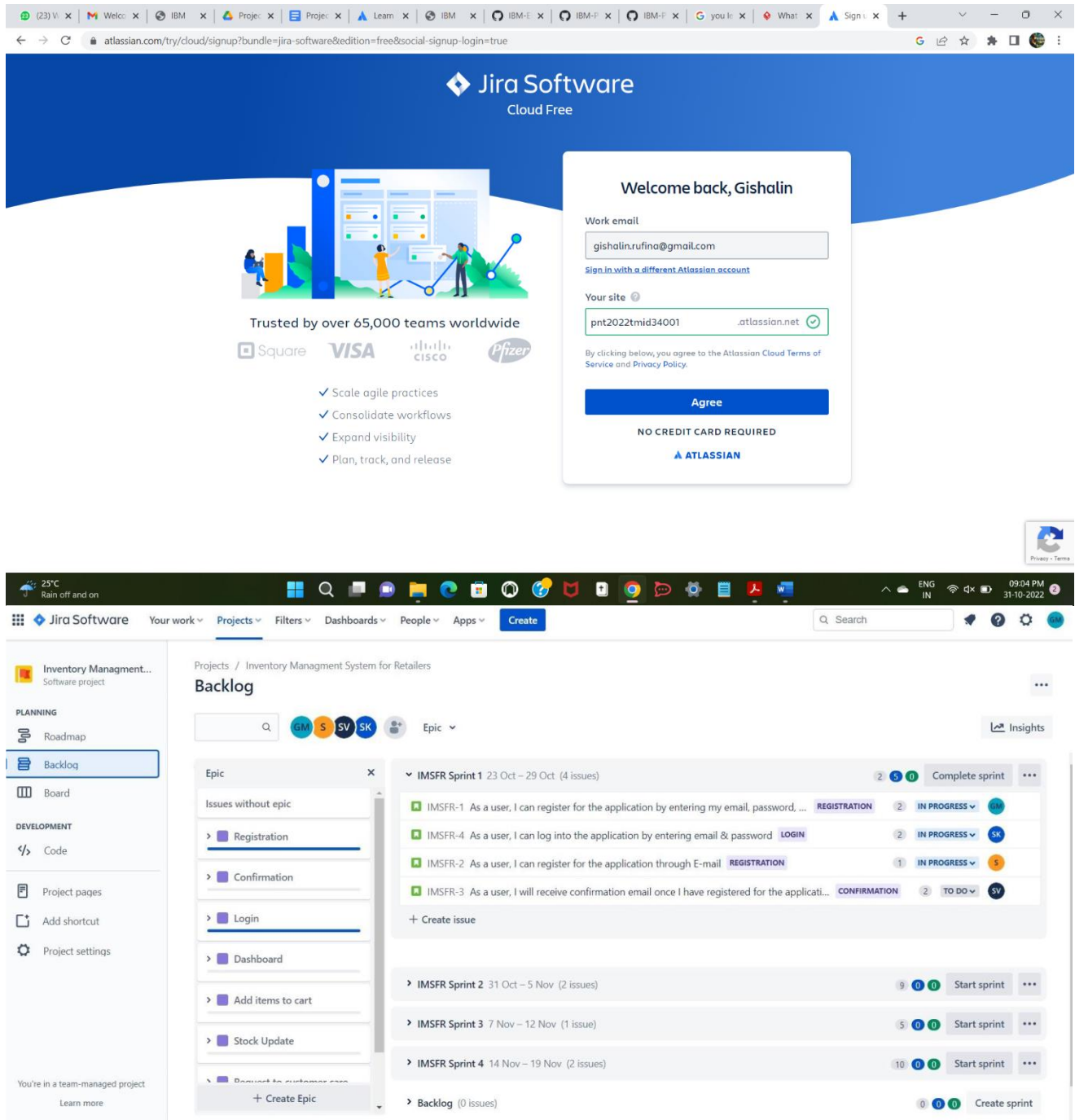
Sprint -3	Chatbot	USN-3	The retailer can easily get the required details through the chatbot.	20	High	Gishalin Rufina, Alice, Arshitha Sherin, Hanan
Sprint -4	Final Delivery	USN-4	Container of application using docker Kubernetes and deployment of the application. Create the documentation and final submit of the application	20	High	Gishalin Rufina, Alice, Arshitha Sherin, Hanan

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

### 6.3.Reports from JIRA

Scrum is one of the most popular frameworks for implementing agile. With scrum, the product is built in a series of fixed-length iterations called sprints that give teams a framework for shipping on a regular cadence. Step-by-step instructions on how to drive a scrum project





Inventory Management...

Software project

PLANNING

Roadmap

Backlog

Board

DEVELOPMENT

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / Inventory Management System for Retailers

Roadmap

Give feedback

Share

Export

Q Search

GM SK SV S

Status category

View settings

	SEP	OCT	NOV	DEC
Sprints		IMSFR-10	IMSFR-11	IMSFR-12
IMSFR-10 Registration				
IMSFR-11 Confirmation				
IMSFR-12 Login				
IMSFR-13 Dashboard				
IMSFR-14 Add items to cart				
IMSFR-15 Stock Update				
IMSFR-16 Request to customer care				
IMSFR-17 Contact Administrator				
+ Create Epic				

Today Weeks Months Quarters

Inventory Management...

Software project

PLANNING

Roadmap

Backlog

Board

DEVELOPMENT

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / Inventory Management System for Retailers

IMSFR Sprint 1

5 days remaining

Complete sprint

Q Search

GM SK S SV

Epic

GROUP BY None

Insights

TO DO 1 ISSUE

As a user, I will receive confirmation email once I have registered for the application

CONFIRMATION

IMSFR-3

2 SV

IN PROGRESS 3 ISSUES

As a user, I can register for the application by entering my email, password, and confirming my password.

REGISTRATION

IMSFR-1

2 GM

As a user, I can log into the application by entering email & password

LOGIN

IMSFR-4

2 SK

As a user, I can register for the application through E-mail

REGISTRATION

IMSFR-2

1 S

IN REVIEW

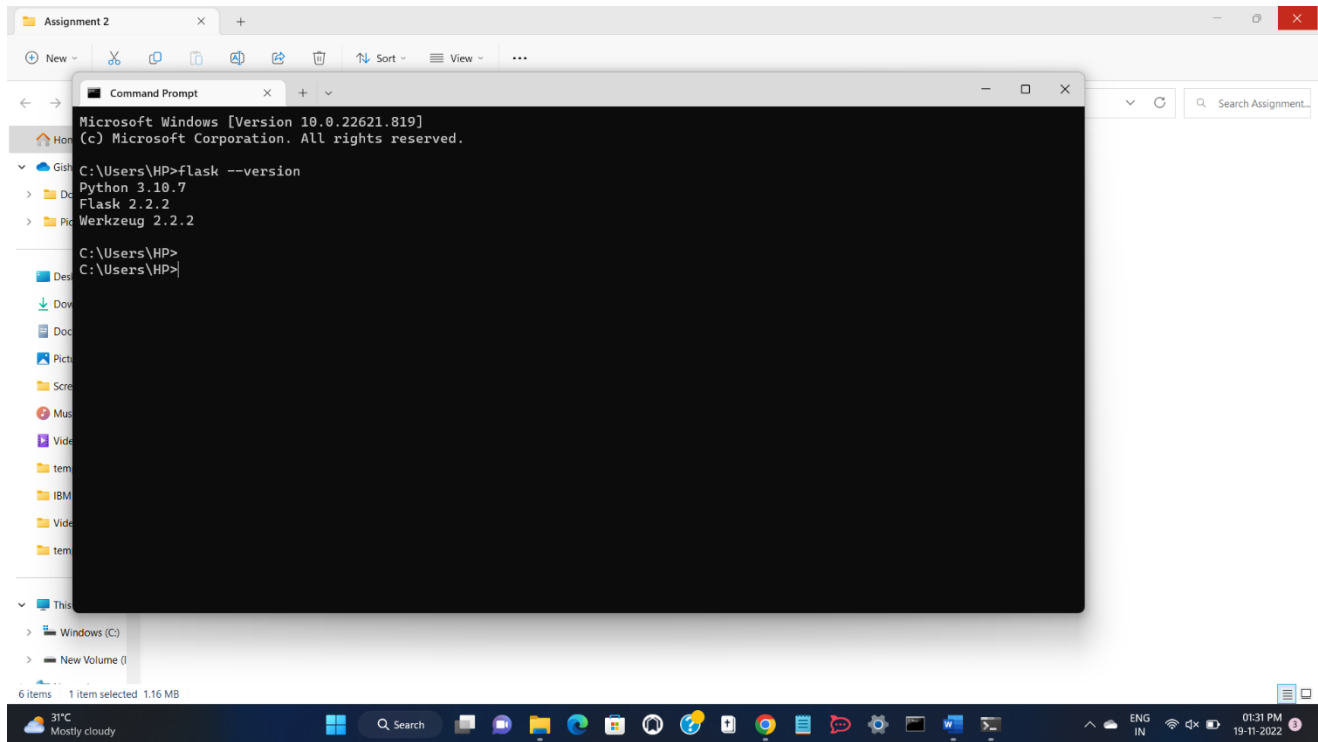
DONE

## 7. CODING & SOLUTIONING

### 7.1 Feature 1

#### 7.1.1.Introduction to flask:

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.



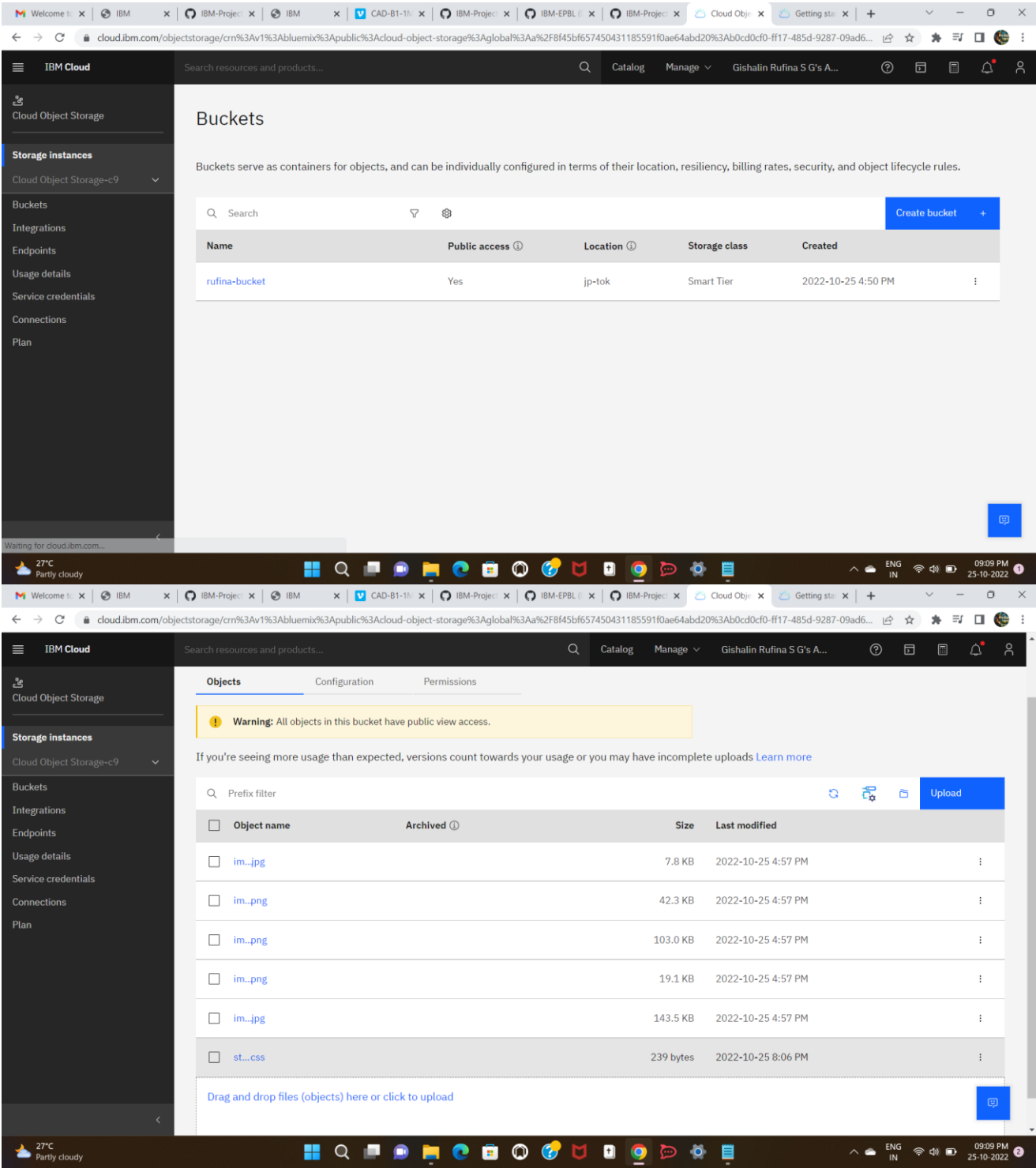
```
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>flask --version
Python 3.10.7
Flask 2.2.2
Werkzeug 2.2.2

C:\Users\HP>
C:\Users\HP>
```

7.1.2.IBM Cloud Object Storage:

IBM Cloud Object Storage is a service offered by IBM for storing and accessing unstructured data. The object storage service can be deployed on-premise, as part of IBM Cloud Platform offerings, or in hybrid form.



### 7.1.3.CODING

#### Index.html

```
{% extends 'base.html' %}
{% block head %}
<title>Login page</title>
{% endblock %}

{% block body %}

<main class="container">
  <div class="mx-auto mt-5 border bg-light login-card " style="width:500px;">
    <h2 class='mx-4 mt-2'>LOGIN</h2>
    <form action="{{url_for('login')}}" method="post">
      <div class="mx-4 mt-2 text-danger">{{ msg }}</div>
      <div class="my-2 mx-4">
        <label for="username">username</label>
        <input type="text" class="form-control" placeholder="adc@gmail.com" name="username"
required />
      </div>
      <div class="my-2 mx-4">
        <label for="password_1">password</label>
        <input type="password" class="form-control" name="password_1" required />
      </div>
      <input type="submit" value="submit" class="btn btn-primary my-4 mt-2 mx-4" />

    </form>
    <p>Don't have an account?<a href="{{ url_for('signup')}}"> Sign Up</a>
  </div>
</main>
</p>

</main>
{% endblock %}
```

#### Signup.html

```
{% extends 'base.html' %} {% block head %}
<title>Signup page</title>
{% endblock %} {% block body %}
<main class="container">
  <div class="mx-auto mt-5 border bg-light" style="width: 500px">
    <h2 class="mx-4 mt-2">REGISTER FORM</h2>
    <div class="mx-4 mt-2 text-danger">{{ msg }}</div>
    <form action="{{url_for('signup')}}" method="post">
      <div class="my-2 mx-4">
        <label for="username">User name</label>
        <input
          type="text"
          class="form-control"
```

```

        placeholder="Ram"
        name="username"
        required
    />
</div>
<div class="my-2 mx-4">
    <label for="email">email</label>
    <input
        type="text"
        class="form-control"
        placeholder="abc@gmail.com"
        name="email"
        required
    />
</div>
<div class="my-2 mx-4">
    <label for="password">password</label>
    <input
        type="password"
        class="form-control"
        placeholder="password"
        name="password"
        required
    />
</div>
<div class="my-2 mx-4">
    <label for="retype">retype password</label>
    <input
        type="password"
        class="form-control"
        placeholder="password"
        name="retype"
        required
    />
</div>

    <input
        type="submit"
        value="submit"
        class="btn btn-primary my-4 my-2 mt-2 mx-4"
    />
    <div class="note mt-3 text-center">
        <!--Register form -->
        <p>already have an account ? please <a href="/login"> login! </a></p>
    </div>
</form>
</div>
</main>
{% endblock%}

```

## Dashboard.html

```
{% extends 'base2.html'%} {% block head %}
<title>Dashboard</title>
{% endblock%} {%block body%}
<h2>Dashboard</h2>
{% include 'table.html' %}
<div class="forms-wrapper">
  <form action="{{url_for('UpdateStocks')}}" method="post">
    <h3>Update Stock</h3>
    <div class="field">
      <label class="custom-label" for="item"> Enter Item</label>
      <input class="text-inputs" type="text" name="item" placeholder="milk" />
    </div>
    <div class="field">
      <label for="input-field">Choose a field :</label>
      <select name="input-field" id="field">
        <option value="NAME">NAME</option>
        <option value="PRICE_PER_QUANTITY">PRICE_PER_QUANTITY</option>
        <option value="QUANTITY">QUANTITY</option>
      </select>
    </div>
    <div class="field">
      <label class="custom-label" for="input-value"> Enter Value</label>
      <input
        class="text-inputs"
        type="text"
        name="input-value"
        placeholder=" "
      />
    </div>
    <button class="submit-button">Update</button>
  </form>

  <form action="{{url_for('addStocks')}}" method="post">
    <h3>Add New Stock</h3>
    <div class="field">
      <label class="custom-label" for="item"> Enter the item</label>
      <input class="text-inputs" name="item" type="text" placeholder="juice" />
    </div>
    <div class="field">
      <label class="custom-label" for="quantity"> Enter quantity</label>
      <input
        class="text-inputs"
        type="number"
        name="quantity"
        placeholder="200"
      />
    </div>
    <div class="field">
      <label class="custom-label" for="price"> Enter price</label>
```

```

        <input class="text-inputs" type="number" name="price" placeholder="25" />
    </div>
    <button class="submit-button">Add Stock</button>
</form>
<form action="{{url_for('deleteStocks')}}" method="post">
    <h3>Remove stocks</h3>
    <div class="field">
        <label class="custom-label" for="item"> Enter the item</label>
        <input class="text-inputs" name="item" type="text" placeholder="juice" />
    </div>
    <button class="submit-button red-button">Remove</button>
</form>
</div>

{% endblock%}

```

## Suppliers.html

```

{% extends 'base2.html'%} {% block head %}
<title>Suppliers</title>
{% endblock%} {%block body%}
<h2>Suppliers</h2>
{% include 'table.html' %}
<div class="forms-wrapper">
    <form action="{{url_for('UpdateSupplier')}}" method="post">
        <h3>Update Supplier</h3>
        <div class="field">
            <label class="custom-label" for="name"> Enter Name</label>
            <input
                class="text-inputs"
                type="text"
                name="name"
                placeholder="Supplier name"
            />
        </div>
        <div class="field">
            <label for="input-field">Choose a field :</label>
            <select name="input-field" id="field">
                <option value="NAME">NAME</option>
                <option value="LOCATION">LOCATION</option>
            </select>
        </div>
        <div class="field">
            <label class="custom-label" for="input-value"> Enter Value</label>
            <input
                class="text-inputs"
                type="text"
                name="input-value"
                placeholder=" "
            />
        </div>
    </form>

```

```

    <button class="submit-button">Update</button>
</form>

<form action="{{url_for('addSupplier')}}" method="post">
    <h3>Add New Supplier</h3>
    <div class="field">
        <label class="custom-label" for="name"> Enter the Supplier</label>
        <input
            class="text-inputs"
            name="name"
            type="text"
            placeholder="Supplier name"
        />
    </div>
    <div class="field">
        <label class="custom-label" for="quantity"> Enter Order ID : </label>
        <select name="order-id-select" id="field">
            {% for order_id in order_ids %}
                <option value="{{ order_id }}">{{order_id}}</option>
            {% endfor %}
        </select>
    </div>
    <div class="field">
        <label class="custom-label" for="location"> Enter Location</label>
        <input
            class="text-inputs"
            type="text"
            name="location"
            placeholder="Location"
        />
    </div>
    <button class="submit-button">Add Stock</button>
</form>

<form action="{{url_for('deleteSupplier')}}" method="post">
    <h3>Delete Supplier</h3>
    <div class="field">
        <label class="custom-label" for="name"> Enter the name</label>
        <input
            class="text-inputs"
            name="name"
            type="text"
            placeholder="Supplier Name"
        />
    </div>
    <button class="submit-button red-button">Delete</button>
</form>
</div>

{% endblock%}

```



SignUp with us!

Firstname:

Middlename:

Lastname:

Gender :  
☐ Male  
☐ Female  
☐ Other

Phone : +91

Address

Email:

Sign in with us!

Username :

Password :

Login

## 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
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

```

```
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;SSLServerCertifi
cate=DigiCertGlobalRootCA.crt;UID=znd71133;PWD=SNOsh2PpbXd6Ew8V", "", "")
```

```
app = Flask(__name__)
app.secret_key = '123'
```

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

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

    if request.method == 'POST':
        un = request.form['username']
        pd = request.form['password_1']
        print(un, pd)
        sql = "SELECT * FROM users WHERE Email =? AND Password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, un)
        ibm_db.bind_param(stmt, 2, pd)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
```

```

if account:
    session['loggedin'] = True
    session['id'] = account['Email']
    userid = account['Email']
    session['username'] = account['Username']
    msg = 'You have logged in successfully!'

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

```

```
@app.route('/signup', methods=['POST', 'GET'])
```

```
def signup():
```

```
    mg = "
```

```
    if request.method == "POST":
```

```
        username = request.form['username']
```

```
        email = request.form['email']
```

```
        pw = request.form['password']
```

```
        sql = 'SELECT * FROM users WHERE Email =?'
```

```
        stmt = ibm_db.prepare(conn, sql)
```

```
        ibm_db.bind_param(stmt, 1, email)
```

```
        ibm_db.execute(stmt)
```

```
        acnt = ibm_db.fetch_assoc(stmt)
```

```
        print(acnt)
```

```
    if acnt:
```

```
        mg = ' This account already exists!!'
```

```
    elif not re.match(r'^@]+@[^@]+\.[^@]+', email):
```

```
        mg = 'Please enter the a valid email address'
```

```
    elif not re.match(r'[A-Za-z0-9]+', username):
```

```
        ms = 'name must contain only character and number'
```

```
    else:
```

```
        insert_sql = 'INSERT INTO users (Firstname,Middlename,Lastname,Email>Password) VALUES (?, ?, ?, ?, ?)'
```

```
        pstmt = ibm_db.prepare(conn, insert_sql)
```

```
        ibm_db.bind_param(pstmt, 1, username)
```

```
        ibm_db.bind_param(pstmt, 2, "firstname")
```

```
        ibm_db.bind_param(pstmt, 3, "lastname")
```

```
        # ibm_db.bind_param(pstmt,4,"123456789")
```

```
        ibm_db.bind_param(pstmt, 4, email)
```

```
        ibm_db.bind_param(pstmt, 5, pw)
```

```
        print(pstmt)
```

```
        ibm_db.execute(pstmt)
```

```
        mg = 'You have successfully registered click login!'
```

```
        message = Mail(
```

```
            from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
```

```
            to_emails=email,
```

```
            subject='New SignUp',
```

```
            html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for choosing us.</p>')
```

```
        sg = SendGridAPIClient(
```

```
            api_key=os.environ.get('SENDGRID_API_KEY'))
```

```
        response = sg.send(message)
```

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

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

```
@app.route('/dashboard', methods=['POST', 'GET'])
```

```
@login_required
```

```
def dashBoard():
```

```
    sql = "SELECT * FROM stock_details"
    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 stock_details (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 stock_details 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 stock_details WHERE Name= ?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            total = dictionary['QUANTITY'] * dictionary['PRICE_PER_QUANTITY']
            insert_sql = 'UPDATE stock_details 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 stock_details 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 order_details"
    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 stock_details WHERE Id= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['Price-per_quantity'])
                query = 'INSERT INTO order_details (Stock_id,Quantity,Date,Delivery,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 order_details 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 order_details 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 supplier_details"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['Order_id'])
        dictionary = ibm_db.fetch_assoc(stmt)
```

```
# get order ids from orders table and identify unassigned order ids
```

```
sql = "SELECT Id FROM order_details"
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 supplier-details 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('supplier_details'))

```

```

@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 supplier_details (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('supplier_details'))

```

```

@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM supplier_details 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('supplier_details'))

```

```

@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:

```

```
email = session['id']
insert_sql = 'SELECT * FROM users WHERE Email=?'
pstmt = ibm_db.prepare(conn, insert_sql)
ibm_db.bind_param(pstmt, 1, email)
ibm_db.execute(pstmt)
dictionary = ibm_db.fetch_assoc(pstmt)
print(dictionary)
except Exception as e:
    msg = e
finally:
    # print(msg)
    return render_template("profile.html", data=dictionary)
```

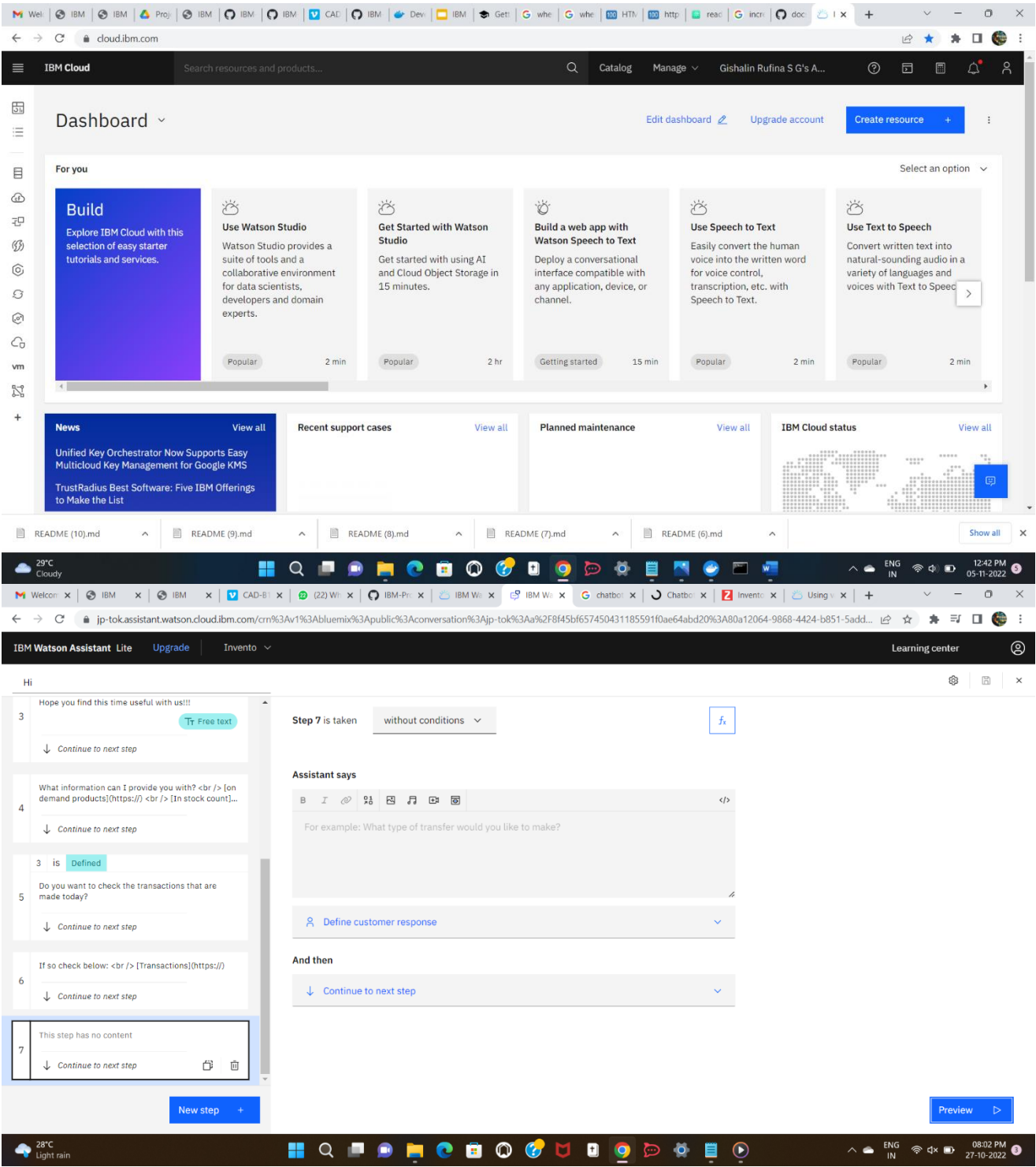
```
@app.route('/logout', methods=['GET'])
@login_required
def logout():
    print(request)
    resp = make_response(render_template("login.html"))
    session.clear()
    return resp
```

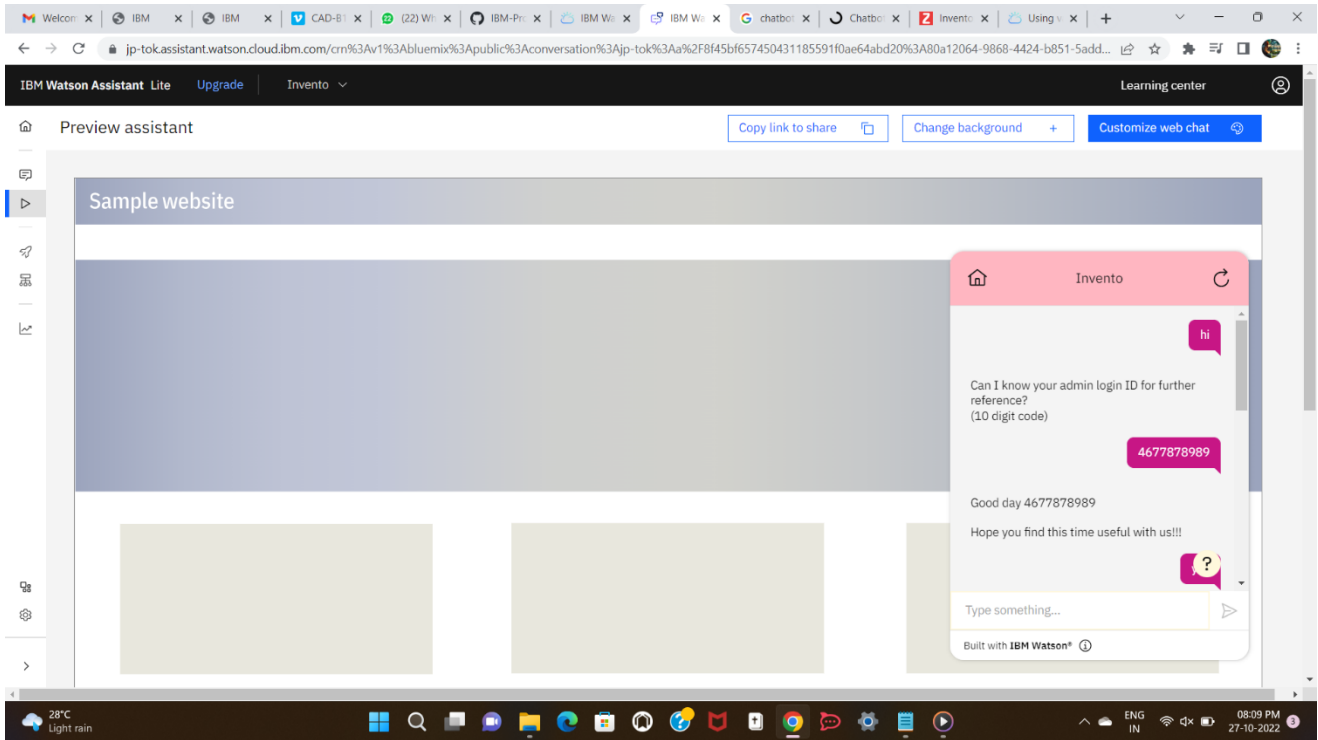
```
if __name__ == '__main__':
    app.run(debug=True)
```

## 7.2 Feature 2

### 7.2.1.IBM Watson Assistant Service:

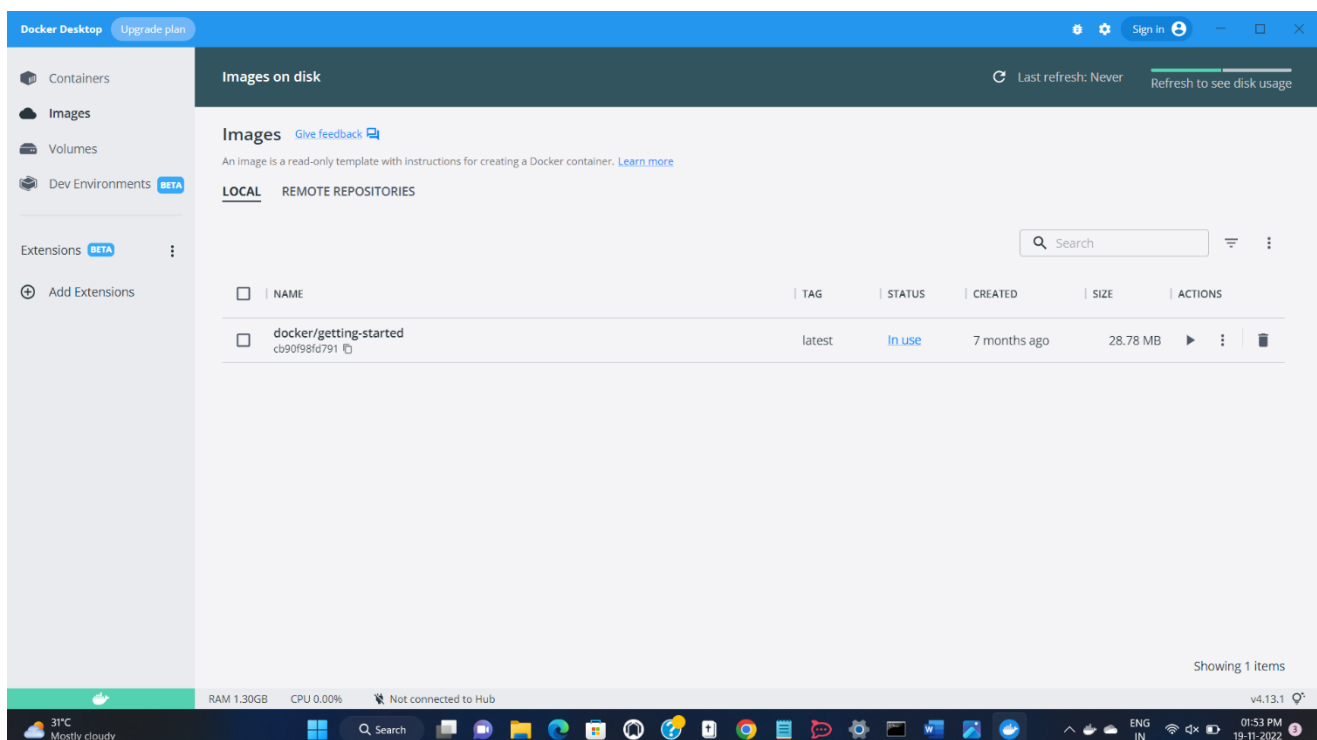
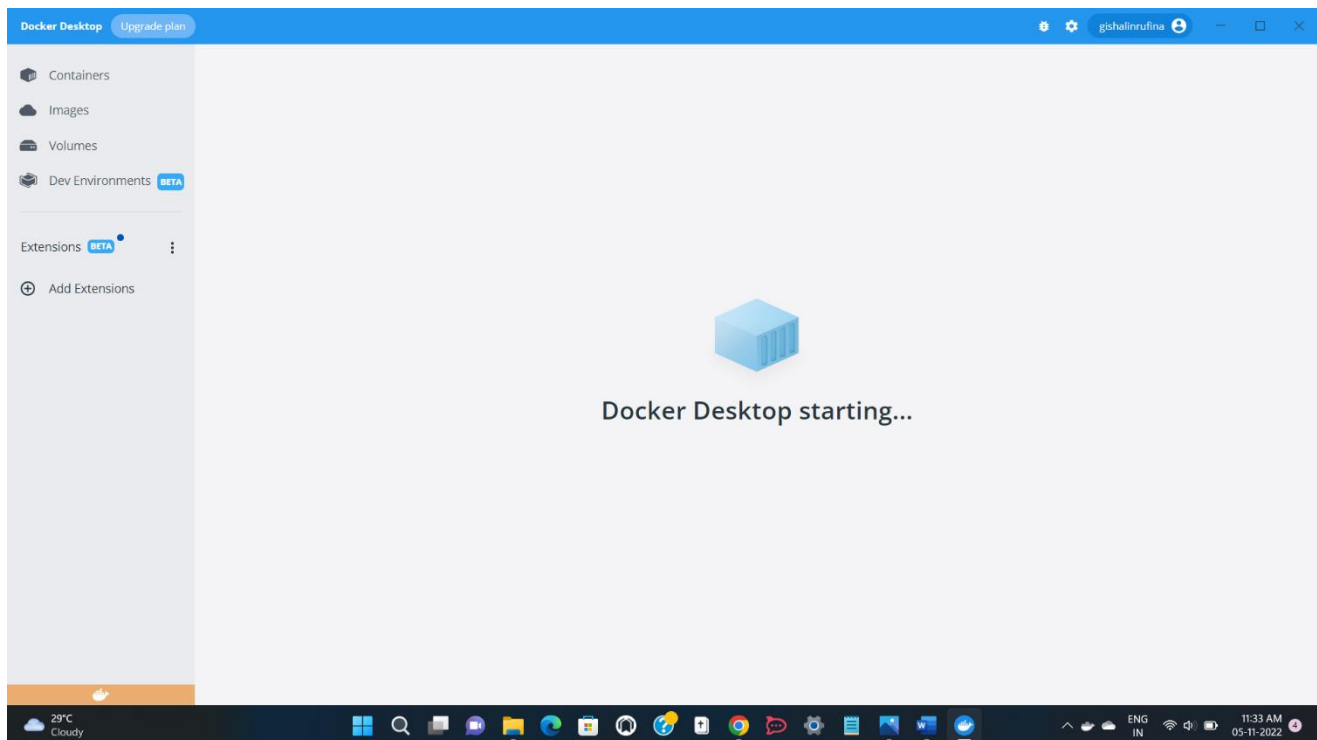
IBM Watson Assistant uses artificial intelligence that understands customers in context to provide fast, consistent, and accurate answers across any application, device, or channel. Remove the frustration of long wait times, tedious searches, and unhelpful chatbots with the leader in trustworthy AI.





### 7.2.2.Docker:

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine.



## 7.3 Database Schema

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTSSequencesApplication objects

Find schemas or tables

Refresh

Schemas

Name	Type	Tables
ZND71133	User	8

Total: 1, selected: 1

Tables

New table

Name	Schema	Properties
ITEM_DETAILS	ZND71133	...
ORDER_DETAILS	ZND71133	...
ORDER_STATUS	ZND71133	...
STOCK_AVAILABILITY	ZND71133	...
STOCK_DETAILS	ZND71133	...
SUPPLIER_DETAILS	ZND71133	...
USERS	ZND71133	...

Total: 8, selected: 0

Inventory UTA re....docxInventory UTA rep....pdfTestcases Report T....xlsxCreate IBM DB2 an....pdf

Show all

31°C Mostly cloudy

Search

IBM Db2 on Cloud

Connections

Connect your apps and clients to IBM Db2 on Cloud

LinuxPowerLinuxMacWindows

Instructions

1. Download Windows driver package

Download Windows driver package from [driver list](#)

File name: ibm\_data\_server\_driver\_package\_win64\_v11.5.exe (104 MB)

2. Install the drivers by running the ibm\_data\_server\_driver\_package\_win64\_v11.5.exe file as an administrator.

3. In The Connection configuration resources section, select whether or not you want to secure your connections by using SSL.

If your application uses its own driver and you want to connect with SSL, download the SSL certificate (DigiCertGlobalRootCA.crt).

For Java apps, use the JDBC string as the database URL in your call to the JDBC getConnection method.

For ODBC apps, add new entries to the db2dsdriver.cfg driver configuration file by running the following commands:

Connection configuration resources

Host name:

815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90i08qkb1od8lcg.databases.appdomain.cloud

With SSL:

Yes

Port number:

30367

Database name:

bludb

User ID:

<user name>

Password:

\*\*\*\*\*

Version:

Compatible with Db2, Version 11.5.0 or later

Download SSL Certificate

JDBC string

Inventory UTA re....docxInventory UTA rep....pdfTestcases Report T....xlsxCreate IBM DB2 an....pdf

Show all

31°C Mostly cloudy

Search

IBM Db2 on Cloud

Connections

Connect your apps and clients to IBM Db2 on Cloud

LinuxPowerLinuxMacWindows

Instructions

1. Download Windows driver package

Download Windows driver package from [driver list](#)

File name: ibm\_data\_server\_driver\_package\_win64\_v11.5.exe (104 MB)

2. Install the drivers by running the ibm\_data\_server\_driver\_package\_win64\_v11.5.exe file as an administrator.

3. In The Connection configuration resources section, select whether or not you want to secure your connections by using SSL.

If your application uses its own driver and you want to connect with SSL, download the SSL certificate (DigiCertGlobalRootCA.crt).

For Java apps, use the JDBC string as the database URL in your call to the JDBC getConnection method.

For ODBC apps, add new entries to the db2dsdriver.cfg driver configuration file by running the following commands:

Connection configuration resources

Host name:

815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90i08qkb1od8lcg.databases.appdomain.cloud

With SSL:

Yes

Port number:

30367

Database name:

bludb

User ID:

<user name>

Password:

\*\*\*\*\*

Version:

Compatible with Db2, Version 11.5.0 or later

Download SSL Certificate

JDBC string

Inventory UTA re....docxInventory UTA rep....pdfTestcases Report T....xlsxCreate IBM DB2 an....pdf

Show all

31°C Mostly cloudy

Search

IBM Db2 on Cloud

Connections

Connect your apps and clients to IBM Db2 on Cloud

LinuxPowerLinuxMacWindows

Instructions

1. Download Windows driver package

Download Windows driver package from [driver list](#)

File name: ibm\_data\_server\_driver\_package\_win64\_v11.5.exe (104 MB)

2. Install the drivers by running the ibm\_data\_server\_driver\_package\_win64\_v11.5.exe file as an administrator.

3. In The Connection configuration resources section, select whether or not you want to secure your connections by using SSL.

If your application uses its own driver and you want to connect with SSL, download the SSL certificate (DigiCertGlobalRootCA.crt).

For Java apps, use the JDBC string as the database URL in your call to the JDBC getConnection method.

For ODBC apps, add new entries to the db2dsdriver.cfg driver configuration file by running the following commands:

Connection configuration resources

Host name:

815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90i08qkb1od8lcg.databases.appdomain.cloud

With SSL:

Yes

Port number:

30367

Database name:

bludb

User ID:

<user name>

Password:

\*\*\*\*\*

Version:

Compatible with Db2, Version 11.5.0 or later

Download SSL Certificate

JDBC string

Inventory UTA re....docxInventory UTA rep....pdfTestcases Report T....xlsxCreate IBM DB2 an....pdf

Show all

cloud.ibm.com/services/dashdb-for-transactions/cn%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%2F8f45b6f657450431185591f0ae64abd20%3A19c82149-cf19-...

IBM Cloud Search resources and products...

Resource list / Db2-id Active Add tags

Manage

Getting started

**Service credentials**

Connections

Service credentials

You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service. [Learn more](#)

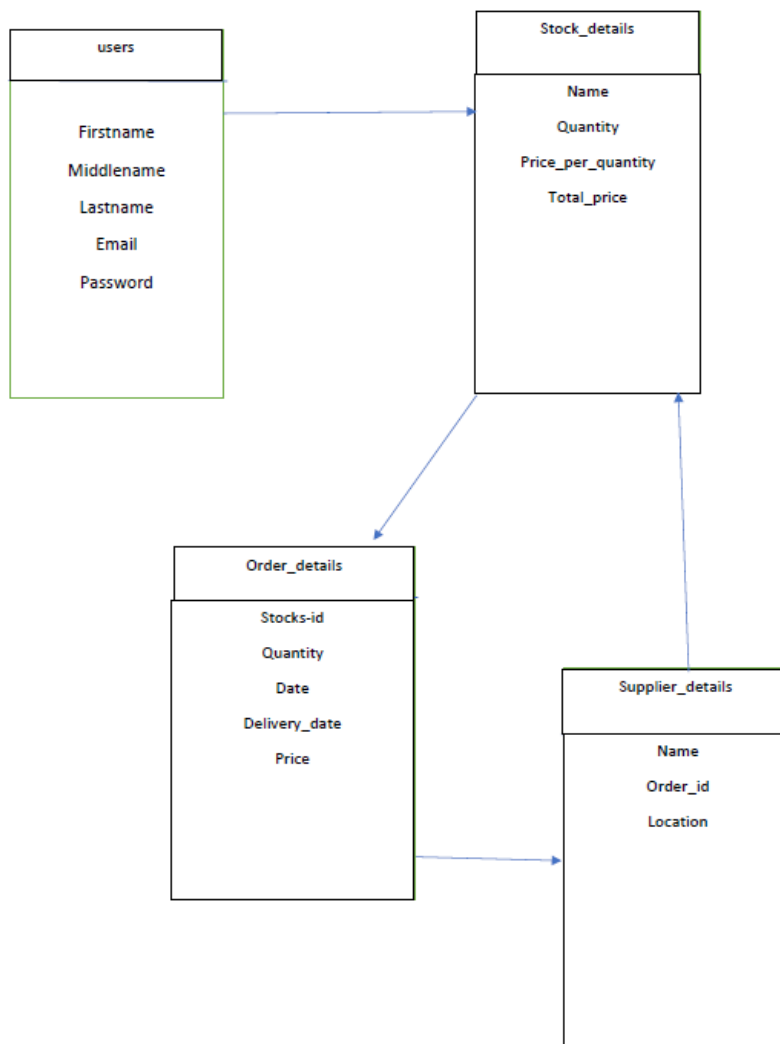
Search credentials...

New credential +

Key name	Date created
Service credentials-1	2022-11-17 6:36 PM

Inventory UTA re...docx Inventory UTA rep...pdf Testcases Report T...xlsx Create IBM DB2 an...pdf Show all

31°C Mostly cloudy Search ENG IN 01:59 PM 19-11-2022



## 8.1 Test Cases

A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly. The purpose of a test case is to determine if different features within a system are performing as expected and to confirm that the system satisfies all related standards, guidelines and customer requirements. The process of writing a test case can also help reveal errors or defects within the system.

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Result	Actual Result	Status
LoginPage_TC_001	Functional	Home Page	Verify user is able to see the Login/Signup popup when user clicked on My account button	1.Enter URL and click go 2.Click on My Account dropdown button 3.Verify login/Signup popup displayed or not	<a href="https://fb0f7bov2d6cndrk8rora.on.drvtw/www.shwetha.com/">https://fb0f7bov2d6cndrk8rora.on.drvtw/www.shwetha.com/</a>	Login/Signup popup should display	Working as expected	Pass
LoginPage_TC_002	UI	Home Page	Verify the UI elements in Login/Signup popup	1.Enter URL and click go 2.Click on My Account dropdown button 3.Verify login/Signup popup with below UI elements: a.email text box b.password text box c.Login button d.New customer? Create account link e.Last password? Recovery password link	<a href="https://fb0f7bov2d6cndrk8rora.on.drvtw/www.shwetha.com/">https://fb0f7bov2d6cndrk8rora.on.drvtw/www.shwetha.com/</a>	Application should show below UI elements: a.email text box b.password text box c.Login button with orange colour d.New customer? Create account link e.Last password? Recovery password link	Working as expected	Pass
LoginPage_TC_003	Functional	Home page	Verify user is able to log into application with Valid credentials	1.Enter URL( <a href="https://shopenzer.com/">https://shopenzer.com/</a> ) and click go 2.Click on My Account dropdown button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: gishalin.rufina@gmail.com password: admin	User should navigate to user account homepage	Working as expected	Pass
LoginPage_TC_004	Functional	Login page	Verify user is able to log into application with InValid credentials	1.Enter URL( <a href="https://shopenzer.com/">https://shopenzer.com/</a> ) and click go 2.Click on My Account dropdown button 3.Enter InValid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: shwethanaidu230@gmail.com password: admin	Application should show 'incorrect email or password' validation message.	Working as expected	Pass
			Verify user is able to log into	1.Enter URL( <a href="https://shopenzer.com/">https://shopenzer.com/</a> ) and click go 2.Click on My Account dropdown button	Username: gishalin.rufina@gmail.com	Application should show 'incorrect email or password' validation	Working as	

[illegible]



## 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	9	4	2	3	20
Duplicate	1	0	3	0	4
External	4	3	0	1	6
Fixed	11	6	4	19	36
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	75

### Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	8	0	0	8
Client Application	20	0	0	20
Security	2	0	0	2



Outsource Shipping	3	0	0	3
Exception Reporting	12	0	0	12
Final Report Output	2	0	0	2
Version Control	1	0	0	1

## 9. RESULTS

### 9.1 Performance Metrics

Performance measurement is the process of collecting, analyzing and/or reporting information regarding the performance of an individual, group, organization, system or component . Definitions of performance measurement tend to be predicated upon an assumption about why the performance is being measured.

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	Inventory Management System for Retail	New	Moderate	No Changes	Moderate		>30 to 50 %	ORANGE	As we have seen the changes
2	Inventory Management System for Retail	New	Low	No Changes	Low		>5 to 10%	GREEN	As we have seen the changes
3	Inventory Management System for Retail	New	High	No Changes	High		>50 to 70%	RED	As we have seen the changes
4	Inventory Management System for Retail	New	Moderate	No Changes	Moderate		>30 to 50 %	ORANGE	As we have seen the changes
NFT - Detailed Test Plan									
S.No	Project Overview	NFT Test approach	Assumptions/Dependencies/Risks	Approvals/SignOff					
1	Inventory Management System for Retail	Scalability	moderate	Gishalin					
End Of Test Report									
S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	(Detected/Closed/Open)	Approvals/SignOff	
1	Inventory Management System for Retail	Scalability	Yes	Good		Increase the number of pods	Closed	Gishalin	

## **10. ADVANTAGES & DISADVANTAGES**

### **10.1 Advantages**

- Better Inventory Accuracy
- Reduced Risk of Overselling
- Cost Savings
- Avoiding Stockouts and Excess Stock
- Better Terms With Vendors and Suppliers
- Increased Profits

### **10.2 Disadvantages**

- Sometimes, the orders are placed at the irregular time periods which may not be convenient to the producers or the suppliers of the materials.
- The items cannot be grouped and ordered at a time since the reorder points occur irregularly.
- If there is a case when the order placement time is very high, there would be two to three orders pending with the supplier each time and there is likelihood that he may supply all orders at a time.

## **11. CONCLUSION**

The Inventory Management System is developed and designed for recording and managing the inventory of an organization. It can also be used for different institution with fewer modification as per requirement. the system can be easily updated as the other institutional requirement may not be integrated on our project . After the continuous effort , testing and debugging the current system is ready to be implemented in an organization. The Inventory management system project that allows user to manage and maintain his/her inventory with ease. The inventory management system has been developed to allow users to add an inventory, delete an inventory, enter inventory quantity and other details, update inventory status and more. The inventory management system has its own intelligently managed support system that allows user to view and manage various inventories added in the system.

## **12. FUTURE SCOPE**

The forces of technology, globalization, and consumer empowerment have profoundly influenced the way that industries have managed inventory in the past thirty years or more.

Globalization has challenged businesses to reposition inventory within its supply chain to take advantage of production economies in remote locations and drive reductions in Cost of Goods Sold. Many organizations have pursued a horizontally integrated supply chain (as opposed to the traditionally vertically-oriented supply chain) as they search for economies of production and delivery. AI will assist businesses in making decisions, especially routine ones, with greater accuracy and with higher levels of sophistication. Computers will learn from experience and will respond to changes with greater levels of quality and certainty. Humans will rely increasingly on AI to make profitable business decisions that balance cost with customer service.

## 13. APPENDIX

Source Code:

### **Display.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>Document</title>
</head>

<body>
  <p>the msg:{{ msg }}</p>
  <p></p>
</body>
```

```
</html>
```

### **Orders.html**

```
{% extends 'base2.html'% } {% block head % }
<title>Orders</title>
{% endblock% } {%block body% }
<h2>Orders</h2>

{% include 'table.html' % }
<div class="forms-wrapper">
  <form action="{ { url_for('createOrder') } }" method="post">
    <h3>Create Order</h3>
    <div class="field">
      <label class="custom-label" for="item">Enter Stock ID</label>
      <input class="text-inputs" type="text" name="stock_id" placeholder="1" />
    </div>
    <div class="field">
      <label class="custom-label" for="item">Enter Quantity</label>
      <input
        class="text-inputs"
        type="number"
        name="quantity"
        placeholder="10"
      />
    </div>
    <button class="submit-button">Create</button>
  </form>

  <form action="{ { url_for('updateOrder') } }" method="post">
    <h3>Update Order</h3>
    <div class="field">
      <label class="custom-label" for="item">Enter Order ID</label>
      <input class="text-inputs" name="item" type="number" placeholder="1" />
    </div>
    <div class="field">
      <label for="custom-label" for="input-field">Choose a field - </label>
```

```

<select name="input-field" id="field">
  <option value="STOCKS_ID">STOCKS_ID</option>
  <option value="QUANTITY">QUANTITY</option>
</select>
</div>
<div class="field">
  <label class="custom-label" for="input-value">Enter Value</label>
  <input class="text-inputs" type="text" name="input-value" />
</div>
<button class="submit-button">Update</button>
</form>
<form action="{ { url_for('cancelOrder') } }" method="post">
  <h3>Cancel Order</h3>
  <div class="field">
    <label class="custom-label" for="item">Enter Order ID</label>
    <input
      class="text-inputs"
      name="order_id"
      type="number"
      placeholder="1"
    />
  </div>
  <button class="submit-button red-button">Cancel</button>
</form>
</div>

```

```
{% endblock%}
```

### Suppliers.html

```

{% extends 'base2.html'%} {% block head %}
<title>Suppliers</title>
{% endblock%} {% block body%}
<h2>Suppliers</h2>
{% include 'table.html' %}
<div class="forms-wrapper">
  <form action="{ { url_for('UpdateSupplier') } }" method="post">
    <h3>Update Supplier</h3>
    <div class="field">
      <label class="custom-label" for="name"> Enter Name</label>
      <input
        class="text-inputs"
        type="text"
        name="name"
        placeholder="Supplier name"
      />
    </div>
    <div class="field">
      <label for="input-field">Choose a field :</label>
      <select name="input-field" id="field">
        <option value="NAME">NAME</option>
        <option value="LOCATION">LOCATION</option>
      </select>
    </div>
    <div class="field">
      <label class="custom-label" for="input-value"> Enter Value</label>
      <input

```

```

        class="text-inputs"
        type="text"
        name="input-value"
        placeholder=" "
    />
</div>
<button class="submit-button">Update</button>
</form>

<form action="{ {url_for('addSupplier')} }" method="post">
<h3>Add New Supplier</h3>
<div class="field">
    <label class="custom-label" for="name"> Enter the Supplier</label>
    <input
        class="text-inputs"
        name="name"
        type="text"
        placeholder="Supplier name"
    />
</div>
<div class="field">
    <label class="custom-label" for="quantity"> Enter Order ID : </label>
    <select name="order-id-select" id="field">
        { % for order_id in order_ids % }
        <option value="{ { order_id } }">{ { order_id } }</option>
        { % endfor % }
    </select>
</div>
<div class="field">
    <label class="custom-label" for="location"> Enter Location</label>
    <input
        class="text-inputs"
        type="text"
        name="location"
        placeholder="Location"
    />
</div>
<button class="submit-button">Add Stock</button>
</form>

<form action="{ {url_for('deleteSupplier')} }" method="post">
<h3>Delete Supplier</h3>
<div class="field">
    <label class="custom-label" for="name"> Enter the name</label>
    <input
        class="text-inputs"
        name="name"
        type="text"
        placeholder="Supplier Name"
    />
</div>
<button class="submit-button red-button">Delete</button>
</form>
</div>

```



```
{% endblock% }
```

### Result.html

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

```
{% block head %}
```

```
<title>Login page</title>
```

```
{% endblock% }
```

```
{% block body% }
```

```
<main class="container ">
```

```
<div class="mx-auto mt-5 border bg-light login-card " style="width:500px;">
```

```
<h2 class='mx-4 mt-2'>Password changed successfully</h2>
```

```
<button class="submit-button">
```

```
<a href="/profile">Go Back</a>
```

```
</button>
```

```
</div>
```

```
</main>
```

```
</p>
```

```
</main>
```

```
{% endblock% }
```

### Table.html

```
{% extends 'base2.html'% } {% block head % }
```

```
<title>Suppliers</title>
```

```
{% endblock% } {% block body% }
```

```
<h2>Suppliers</h2>
```

```
<p>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

```
</p>
```

```
{% include 'table.html' % }
```

```
<div class="forms-wrapper">
```

```
<form action="{ {url_for('UpdateSupplier') } }" method="post">
```

```
<h3>Update Supplier</h3>
```

```
<div class="field">
```

```
<label class="custom-label" for="name"> Enter Name</label>
```

```
<input
```

```
class="text-inputs"
```

```
type="text"
```

```
name="name"
```

```
placeholder="Supplier name"
```

```
/>
```

```
</div>
```

```
<div class="field">
```

```
<label for="input-field">Choose a field :</label>
```

```
<select name="input-field" id="field">
```

```
<option value="NAME">NAME</option>
```

```
<option value="LOCATION">LOCATION</option>
```

```
</select>
```

```
</div>
```

```
<div class="field">
```

```
<label class="custom-label" for="input-value"> Enter Value</label>
```

```
<input
```

```
class="text-inputs"
```

```

        type="text"
        name="input-value"
        placeholder=" "
    />
</div>
<button class="submit-button">Update</button>
</form>

<form action="{ { url_for('addSupplier') } }" method="post">
<h3>Add New Supplier</h3>
<div class="field">
    <label class="custom-label" for="name"> Enter the Supplier</label>
    <input
        class="text-inputs"
        name="name"
        type="text"
        placeholder="Supplier name"
    />
</div>
<div class="field">
    <label class="custom-label" for="quantity"> Enter Order ID : </label>
    <select name="order-id-select" id="field">
        {% for order_id in order_ids %}
        <option value="{ { order_id } }">{ { order_id } }</option>
        {% endfor %}
    </select>
</div>
<div class="field">
    <label class="custom-label" for="location"> Enter Location</label>
    <input
        class="text-inputs"
        type="text"
        name="location"
        placeholder="Location"
    />
</div>
<button class="submit-button">Add Stock</button>
</form>

<form action="{ { url_for('deleteSupplier') } }" method="post">
<h3>Delete Supplier</h3>
<div class="field">
    <label class="custom-label" for="name"> Enter the name</label>
    <input
        class="text-inputs"
        name="name"
        type="text"
        placeholder="Supplier Name"
    />
</div>
<button class="submit-button red-button">Delete</button>
</form>
</div>

{% endblock%}

```

## Style.css

```
@import 'https://fonts.googleapis.com/css?family=Poppins:300,400,500,600,700';
```

```
body {  
  font-family: 'Poppins', sans-serif;  
  background: #fafafa;  
}
```

```
p {  
  font-family: 'Poppins', sans-serif;  
  font-size: 1.1em;  
  font-weight: 300;  
  line-height: 1.7em;  
  color: #999;  
}
```

```
a,  
a:hover,  
a:focus {  
  color: inherit;  
  text-decoration: none;  
  transition: all 0.3s;  
}
```

```
.navbar {  
  padding: 15px 10px;  
  background: #fff;  
  border: none;  
  border-radius: 0;  
  margin-bottom: 40px;  
  box-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1);  
}
```

```
.navbar-btn {  
  box-shadow: none;  
  outline: none !important;  
  border: none;  
}
```

```
.line {  
  width: 100%;  
  height: 1px;  
  border-bottom: 1px dashed #ddd;  
  margin: 40px 0;  
}
```

```
display: flex;  
width: 100%;  
align-items: stretch;  
}
```

```
#sidebar {  
  min-width: 250px;  
  max-width: 250px;  
  background: #48494b;
```

```
color: #fff;
transition: all 0.3s;
}
```

```
#sidebar.active {
margin-left: -250px;
}
```

```
#sidebar .sidebar-header {
padding: 20px;
background: #48494b;
}
```

```
#sidebar ul.components {
padding: 20px 0;
border-bottom: 1px solid #47748b;
}
```

```
#sidebar ul p {
color: #fff;
padding: 10px;
}
```

```
.project-title {
font-size: 20px;
padding-left: 10px;
text-align: center;
}
```

```
#sidebar ul li a {
padding: 10px;
font-size: 1.1em;
display: block;
}
```

```
#sidebar ul li a:hover {
color: #7386d5;
background: #fff;
}
```

```
#sidebar ul li.active > a,
a[aria-expanded='true'] {
color: #fff;
background: #48494b;
}
```

```
a[data-toggle='collapse'] {
position: relative;
}
```

```
.dropdown-toggle::after {
display: block;
position: absolute;
top: 50%;
right: 20px;
```

```
transform: translateY(-50%);  
}
```

```
ul ul a {  
  font-size: 0.9em !important;  
  padding-left: 30px !important;  
  background: #48494b;  
}
```

```
ul.CTAs {  
  padding: 20px;  
}
```

```
ul.CTAs a {  
  text-align: center;  
  font-size: 0.9em !important;  
  display: block;  
  border-radius: 5px;  
  margin-bottom: 5px;  
}
```

```
a.download {  
  background: #fff;  
  color: #48494b;  
}
```

```
a.article,  
a.article:hover {  
  background: #48494b !important;  
  color: #fff !important;  
}
```

```
.login-card {  
  box-shadow: rgba(0, 0, 0, 0.35) 0px 5px 15px;  
  border-radius: 10px;  
  padding: 10px;  
}
```

```
.login-card p {  
  padding-left: 20px;  
}
```

```
.login-card a {  
  color: rgba(84, 84, 220, 0.888);  
}
```

```
#content {  
  width: 100%;  
  padding: 20px;  
  min-height: 100vh;  
  transition: all 0.3s;  
}
```

```
@media (max-width: 768px) {  
  #sidebar {
```

```

    margin-left: -250px;
}
#sidebar.active {
    margin-left: 0;
}
#sidebarCollapse span {
    display: none;
}
}

```

/\* Table Styles \*/

```

.table-wrapper {
    margin: 10px 70px 70px;
    box-shadow: rgba(99, 99, 99, 0.2) 0px 2px 8px 0px;
}

```

```

.fl-table {
    border-radius: 5px;
    font-size: 16px;
    font-weight: normal;
    border: none;
    border-collapse: collapse;
    width: 100%;
    max-width: 100%;
    white-space: nowrap;
    background-color: white;
}

```

```

.fl-table td,
.fl-table th {
    text-align: center;
    padding: 8px;
}

```

```

.fl-table td {
    border-right: 1px solid #f8f8f8;
    font-size: 16px;
}

```

```

.fl-table thead th {
    color: #ffffff;
    background: #68716e !important ;
}

```

```

.fl-table thead:nth-child(odd) {
    color: #ffffff;
    background: #324960;
}

```

```

.fl-table tr:nth-child(even) {
    background: #f8f8f8;
}
.custom-label {

```

```

font-size: 18px;
font-weight: 400;
}
.field input[type='text'] {
/* width: 100%; */
border: 2px solid #aaa;
border-radius: 4px;
/* margin: 8px 0; */
outline: none;
padding: 2px 10px;
box-sizing: border-box;
transition: 0.3s;
}
.field input[type='number'] {
/* width: 100%; */
border: 2px solid #aaa;
border-radius: 4px;
/* margin: 8px 0; */
outline: none;
padding: 2px 10px;
box-sizing: border-box;
transition: 0.3s;
}

.submit-button {
padding: 5px 10px;
color: white;
background-color: rgb(41, 115, 41);
border: none;
border-radius: 8px;
min-width: 100px;
}
.submit-button a {
color: white;
}
.mg-20 {
margin-top: 20px;
}
.user-deatils h4 {
font-size: 18px;
}
/* .field input[type='text']:focus {
border-color: rgba(59, 67, 75, 0.687);
box-shadow: 0 0 8px 0 rgba(80, 94, 108, 0.667);
} */
.field {
display: flex;
align-items: center;
padding: 10px 0px;
}
.text-inputs {
margin: 0px 10px;
}
/* Responsive */

```

```
@media (max-width: 767px) {  
  .fl-table {  
    display: block;  
    width: 100%;  
  }  
  .table-wrapper:before {  
    content: 'Scroll horizontally >';  
    display: block;  
    text-align: right;  
    font-size: 11px;  
    color: white;  
    padding: 0 0 10px;  
  }  
  .fl-table thead,  
  .fl-table tbody,  
  .fl-table thead th {  
    display: block;  
  }  
  .fl-table thead th:last-child {  
    border-bottom: none;  
  }  
  .fl-table thead {  
    float: left;  
  }  
  .fl-table tbody {  
    width: auto;  
    position: relative;  
    overflow-x: auto;  
  }  
  .fl-table td,  
  .fl-table th {  
    padding: 20px 0.625em 0.625em 0.625em;  
    height: 60px;  
    vertical-align: middle;  
    box-sizing: border-box;  
    overflow-x: hidden;  
    overflow-y: auto;  
    width: 120px;  
    font-size: 13px;  
    text-overflow: ellipsis;  
  }  
  .fl-table thead th {  
    text-align: left;  
    border-bottom: 1px solid #f7f7f9;  
  }  
  .fl-table tbody tr {  
    display: table-cell;  
  }  
  .fl-table tbody tr:nth-child(odd) {  
    background: none;  
  }  
  .fl-table tr:nth-child(even) {  
    background: transparent;  
  }  
}
```



```

}
.fl-table tr td:nth-child(odd) {
    background: #f8f8f8;
    border-right: 1px solid #e6e4e4;
}
.fl-table tr td:nth-child(even) {
    border-right: 1px solid #e6e4e4;
}
.fl-table tbody td {
    display: block;
    text-align: center;
}
}

.forms-wrapper {
    display: flex;
    /* align-items: center; */
    justify-content: space-around;
}

.red-button {
    background-color: rgb(186, 13, 13);
}

```

### **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
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

```

```

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;S
SLServerCertificate=DigiCertGlobalRootCA.crt;UID=znd71133;PWD=SNOsh2PpbXd6Ew8V", "", "")

```

```

app = Flask(__name__)
app.secret_key = '123'

```

```

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

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

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

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

```
@app.route('/signup', methods=['POST', 'GET'])
def signup():
    mg = ""
    if request.method == "POST":
        username = request.form['username']
```

```

email = request.form['email']
pw = request.form['password']
sql = 'SELECT * FROM users WHERE Email =?'
stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt, 1, email)
ibm_db.execute(stmt)
acnt = ibm_db.fetch_assoc(stmt)
print(acnt)

if acnt:
    mg = ' This account already exists!!'

elif not re.match(r'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z]+', email):
    mg = 'Please enter the a valid email address'
elif not re.match(r'^[A-Za-z0-9]+', username):
    ms = 'name must contain only character and number'
else:
    insert_sql = 'INSERT INTO users (Firstname,Middlename,Lastname,Email>Password)
VALUES (?, ?, ?, ?, ?)'
    pstmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(pstmt, 1, username)
    ibm_db.bind_param(pstmt, 2, "firstname")
    ibm_db.bind_param(pstmt, 3, "lastname")
    # ibm_db.bind_param(pstmt,4,"123456789")
    ibm_db.bind_param(pstmt, 4, email)
    ibm_db.bind_param(pstmt, 5, pw)
    print(pstmt)
    ibm_db.execute(pstmt)
    mg = 'You have successfully registered click login!'
    message = Mail(
        from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
        to_emails=email,
        subject='New SignUp',
        html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for
choosing us.</p>')

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

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

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

@app.route('/dashboard', methods=['POST', 'GET'])
@login_required
def dashBoard():
    sql = "SELECT * FROM stock_details"
    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 stock_details (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 stock_details 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 stock_details WHERE Name= ?'
        pstmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(pstmt, 1, item)
        ibm_db.execute(pstmt)
        dictionary = ibm_db.fetch_assoc(pstmt)
        print(dictionary)
        total = dictionary['QUANTITY'] * dictionary['PRICE_PER_QUANTITY']
        insert_sql = 'UPDATE stock_details 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 stock_details 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 order_details"
    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 stock_details WHERE Id= ?'
        stmt = ibm_db.prepare(conn, query)
        ibm_db.bind_param(stmt, 1, stock_id)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        if dictionary:
            quantity = request.form['quantity']
            date = str(datetime.now().year) + "-" + str(
                datetime.now().month) + "-" + str(datetime.now().day)
            delivery = datetime.now() + timedelta(days=7)
            delivery_date = str(delivery.year) + "-" + str(
                delivery.month) + "-" + str(delivery.day)
            price = float(quantity) * \
                float(dictionary['Price-per_quantity'])
            query = 'INSERT INTO order_details (Stock_id,Quantity,Date,Delivery,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 order_details 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 order_details 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 supplier_details"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['Order_id'])
        dictionary = ibm_db.fetch_assoc(stmt)

```

```

# get order ids from orders table and identify unassigned order ids

```

```

    sql = "SELECT Id FROM order_details"
    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 supplier-details SET ' + field + " = '" + value + "' WHERE Name='" + item + "'"

```



```

        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('supplier_details'))

@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 supplier_details (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('supplier_details'))

@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM supplier_details 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('supplier_details'))

@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:

```

```

    email = session['id']
    insert_sql = 'SELECT * FROM users WHERE Email=?'
    pstmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(pstmt, 1, email)
    ibm_db.execute(pstmt)
    dictionary = ibm_db.fetch_assoc(pstmt)
    print(dictionary)
except Exception as e:
    msg = e
finally:
    # print(msg)
    return render_template("profile.html", data=dictionary)

```

```

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

```

```

if __name__ == '__main__':
    app.run(debug=True)

```

Github Link:

<https://github.com/IBM-EPBL/IBM-Project-42351-1660660423>