INVENTORY MANAGEMENT FOR RETAIER CLOUD BASED APPLICATION

A MINI PROJECT REPORT

Submitted by

MOHANKUMAR S (AC19UCS071)

KAVYA R (AC19UCS052)

LEKHA M (AC19UCS062)

MOHANKUMAR S (AC19UCS070)

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

ADHIYAMAAN COLLEGE OF ENGINEERING (AUTONOMOUS)
Dr. M.G.R NAGAR, HOSUR- 635 130

ANNA UNIVERSITY: : CHENNAI- 600 025

NOVEMBER 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this main project report **“PROJECT ON INVENTORY MANAGEMENT FOR RETAILERS”** is the Bonafede work of **“MOHANKUMAR S(AC19UCS071), KAVYA(AC19UCS052), LEKHA M(AC19UCS062), MOHANKUMAR S (AC19UCS070)”** who carried out the project under my supervision.

SIGNATURE
Dr. G. FATHIMA, M.E., Ph.D.,
HEAD OF THE DEPARTMENT
PROFESSOR,
Department of CSE,
Adhiyamaan College of Engineering,
(Autonomous) Dr.
M.G.R. Nagar,
Hosur – 635 130.

SIGNATURE
Ms.S. VEERADANYA
MENTOR
ASSISTANT PROFESSOR,
Department of CSE,
Adhiyamaan College of Engineering,
(Autonomous) Dr.
M.G.R. Nagar,
Hosur – 635 130.

Submitted for Main project VIVA-VOCE Examination held on ________________ at **ADHIYAMAAN COLLEGE OF ENGINEERING (AUTONOMOUS), Hosur**.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we thank the almighty for granting us the wisdom, strength and grace to complete the report and for being with us in every step that we look in order to complete the project successfully.

We are grateful to our beloved Principal **Dr. G. RANGANATH, M.E., Ph.D.,** Principal, Adhiyamaan College of Engineering (Autonomous), Hosur, for providing the opportunity to do this work in the premises.

We acknowledge our heartfelt gratitude to **Dr. G. FATHIMA, ME., Ph.D.,** Professor and Head of the Department, Department of Computer Science and Engineering, Adhiyamaan College of engineering (Autonomous), Hosur, for her guidance and valuable suggestions and encouragement throughout this project.

We are highly indebted to **Mrs. S. VEERADANYA., Assistant** Professor, Department of Computer Science and Engineering, Adhiyamaan College of engineering (Autonomous), Hosur, whose immense support, encouragement and valuable guidance made us to complete this project successfully.

We also extend our thanks to Project Coordinator and all Staff Members for their support in completing the project successfully.

Finally, we would like to thank our parents, without their motivations and support would not have been possible for us to complete this project successful.

ABSTRACT

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.

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

TITLE

CHAPTER 1

1. INTRODUCTION

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

Inventory management is the process of tracking and managing inventory in a retail environment. It includes the tracking of inventory levels, orders, and sales. It also involves the management of stock levels, pricing, and promotions. Inventory management is a critical part of retail operations.

It helps retailers to keep track of their inventory, ensure that they have the right products in stock, and manage their stock levels.

It also helps retailers to optimize their stock levels and pricing. Inventory management is a complex process. It requires the use of multiple software applications and data sources.

Retailers need to have a clear understanding of their inventory levels and the products they sell.

They also need to be able to track and manage their inventory in real-time. The first step in inventory management is to track inventory levels.

This can be done manually or through the use of an inventory management system. Inventory management systems are designed to track and manage inventory in real-time.

They provide retailers with the ability to view their inventory levels, stock levels, and sales. They also allow retailers to manage their stock levels and pricing. Inventory management systems can be used to track inventory in a number of ways.

1.1 Project Overview

Inventory management is the process of tracking and managing inventory in a retail environment. It includes the tracking of inventory levels, orders, and sales. It also involves the management of stock levels, pricing, and promotions. Inventory management is a critical part of retail operations. It helps retailers to keep track of their inventory, ensure that they have the right products in stock, and manage their stock levels.

1.2 Purpose

They provide retailers with the ability to view their inventory levels, stock levels, and sales. They also allow retailers to manage their stock levels and pricing. Inventory management systems can be used to track inventory in a number of ways.

It also helps retailers to optimize their stock levels and pricing. Inventory management is a complex process. It requires the use of multiple software applications and data sources.

CHAPTER 2

2. LITERATURE SURVEY

A literature review is a comprehensive summary of previous research on a topic. The literature review surveys scholarly articles, books, and other sources relevant to a particular area of research. The review should enumerate, describe, summarize, objectively evaluate and clarify this previous research. It should give a theoretical base for the research and help you (the author) determine the nature of your research. The literature review acknowledges the work of previous researchers, and in so doing, assures the reader that your work has been well conceived. It is assumed that by mentioning a previous work in the field of study, that the author has read, evaluated, and assimilated that work into the work at hand.

2.1 EXISTING PROBLEM

They introduce Agent technology into domestic storage management and uses he autonomy, reactivity and sociality of Agent to realize the seamless connection among enterprises by defining interaction and cooperation mechanisms among different Agents. This paper mainly designs a storage management system model based on multi-Agent and describes main Agent cooperation processes of the system.

In the design of storage management system model based on multi-Agent in this paper, we use a hierarchical federation multi-Agent system organization structure and the cooperation among Agents is based on improved contract net protocol, which enhances system performance on the whole. Next, we will analyse from Agent performance and system processing efficiency. The autonomy of Agent in the model is mainly manifested.

2.2 REFERENCE

1.Aditya A. Pande, Sabahudin, “Study of Material Management Techniques on Construction project”, International Journal of Informative & Futuristic Research, ISSN: 2347-1697, Vol.2 (3),
May 2015, pp.34793486.

2. Smangele Raphaelle, Gomathy Nathan and Chitra, “Inventory Management. A Case Study”, International Journal of Emerging Research in Management &Technology, ISSN: 2278-9359, Vol.3 (3) June 2014, pp.94-102.

3. Ashwini Patil, Smite V. Pat Askar, “Analysing Material Management Techniques on Construction Project”, International Journal of Engineering and Innovative Technology (IJEIT), Vol.3 (4), Jan 2013, pp.96-100.

4. "Integrations and Apps for Online Inventory Management. Software Trade Gecko". www.tradege cko.com. Retrieved2015-11-24.

2.3 PROBLEM STATEMENT DEFINITION

Retail inventory management is the process of ensuring carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply. Retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information with which to run their businesses.

CHAPTER 3

3. IDEATION & PROPOSED SOLUTION

Ideation is the process of forming ideas from conception to implementation, most often in a business setting. Ideation is expressed via graphical, written, or verbal methods, and arises from past or present knowledge, influences, opinions, experiences, and personal convictions.

PROPOSED SOLUTION

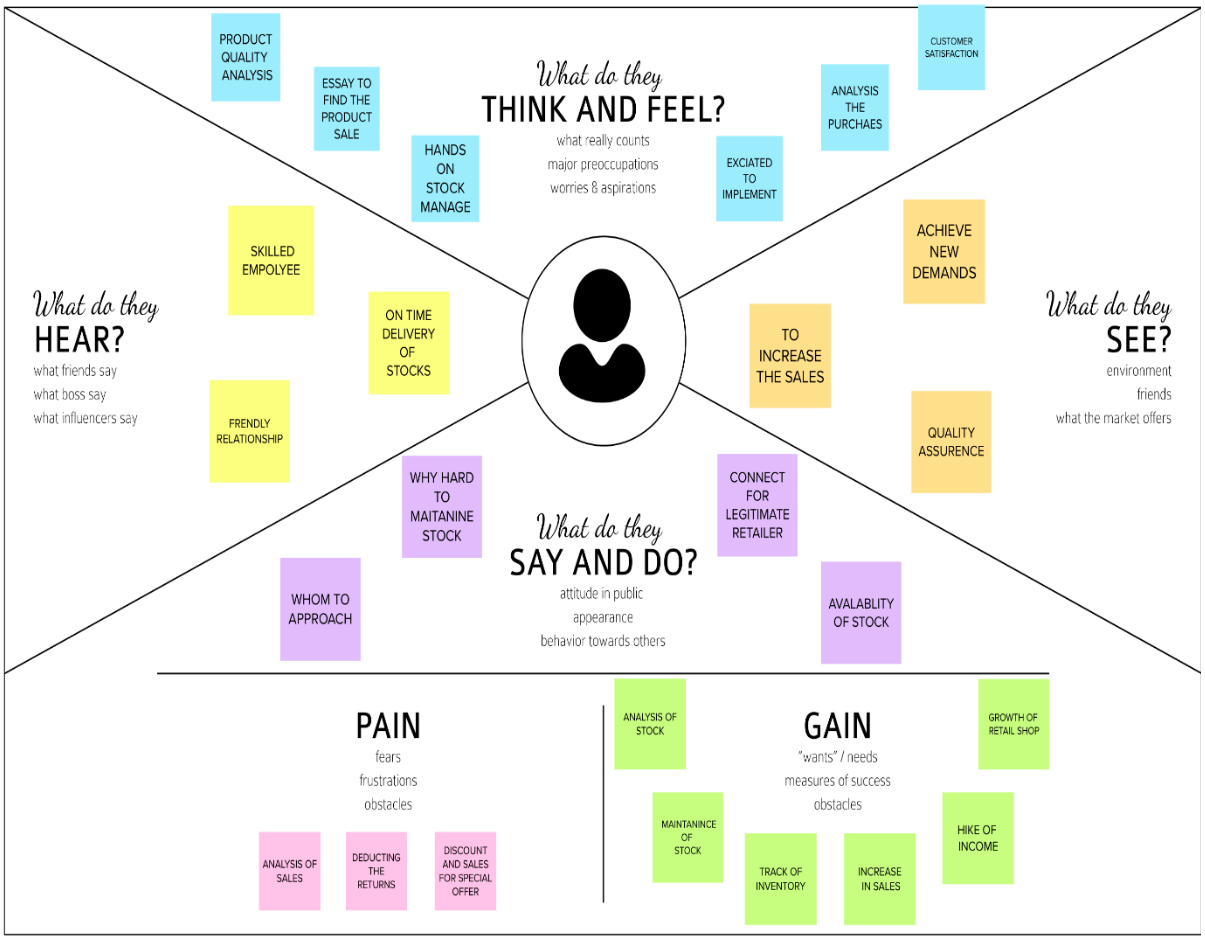
Effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses.

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

- Retailers track and manage stocks.
- Updating the inventory details.
- Adding new stock by submitting essential details.
- System will automatically send an email alert to the retailers if there is no stock found in their accounts.
- Can order new stock.

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative tool teams can use to gain a deeper insight into their customers. Much like a user persona, an empathy map can represent a group of users, such as a customer segment. The empathy map was originally created by Dave Gray and has gained much popularity within the agile community.



what do they think and feel:

- ➢ Product Quality Analysis
- ➢ Easy to find the product sale
- ➢ Hands on stock maintain
- ➢ Exited to Implement
- ➢ Analysis the purchase
- ➢ Customer Satisfaction

what do they See:

- To Increase Sales
- Quality Assurance
- Archive new demand

what do they Say and Do:

- Whom to approach
- How to maintain stock
- Connect to legitimate retailers
- Available of stock

Pains and Gains:

- Analysis of Sales
- Deducting the returns
- Discount and Sales for Special offers
- Analysis the sales
- Maintain of Stock
- Track of Inventory
- Increase Sales
- Hike of Income
- Growth of retail shop

what do they Here:

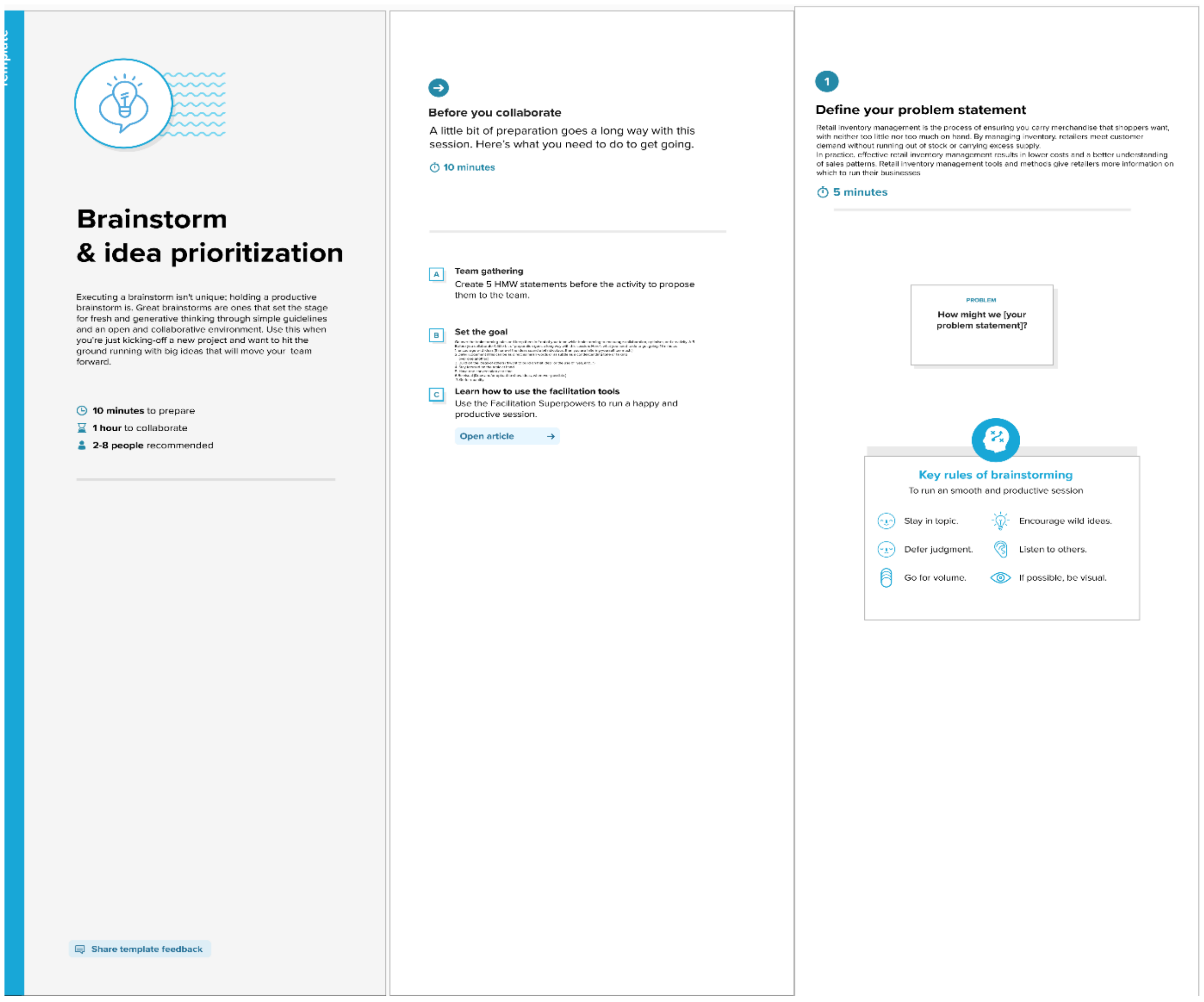
- Skilled Employment
- On time delivery product
- Friendly Relationship

3.1 IDEATION AND BRAINSTROMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity.

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 number of creative solutions.

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



Brainstorm & idea prioritization

Executing a brainstorm isn't unique; holding a productive brainstorm is. Great brainstorms are ones that set the stage for fresh and generative thinking through simple guidelines and an open and collaborative environment. Use this when you're just kicking-off a new project and want to hit the ground running with big ideas that will move your team forward.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Share template feedback

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**
Create 5 HMW statements before the activity to propose them to the team.

B **Set the goal**
[illegible]

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**

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

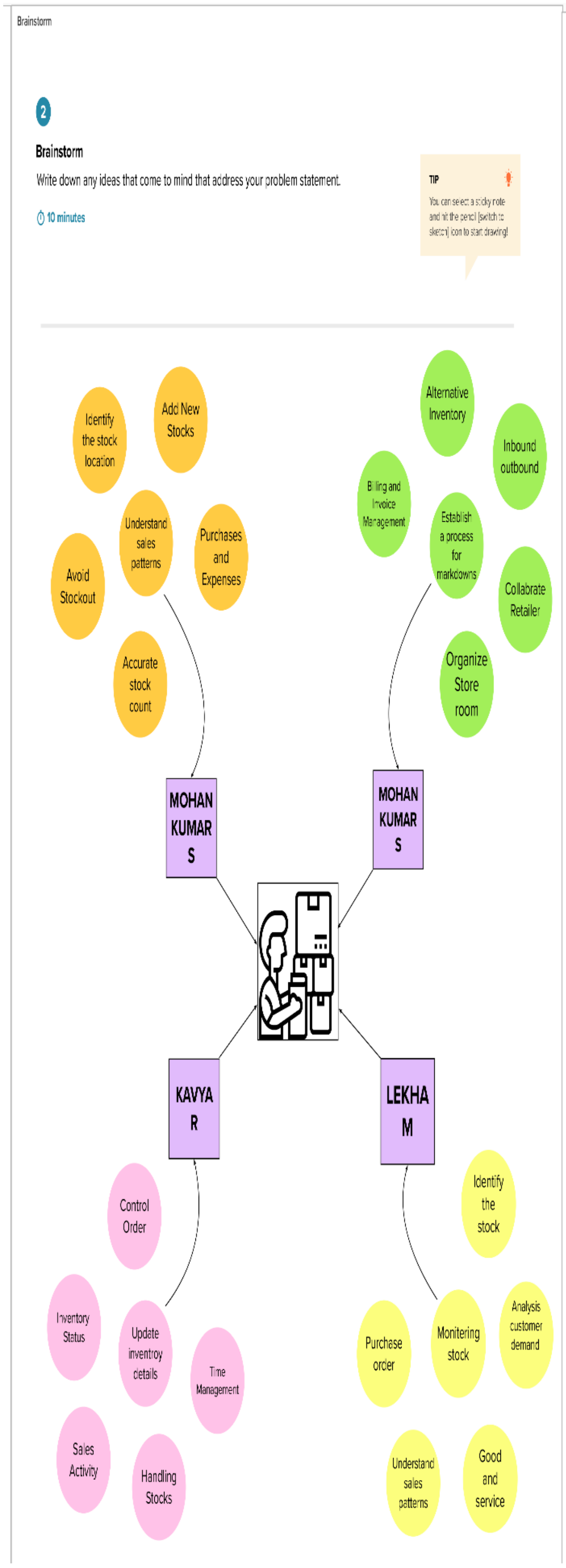
5 minutes

PROBLEM
How might we [your problem statement]?

Key rules of brainstorming
To run an smooth and productive session

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

Step-2: Brainstorm, Idea Listing and Grouping



Brainstorm

2

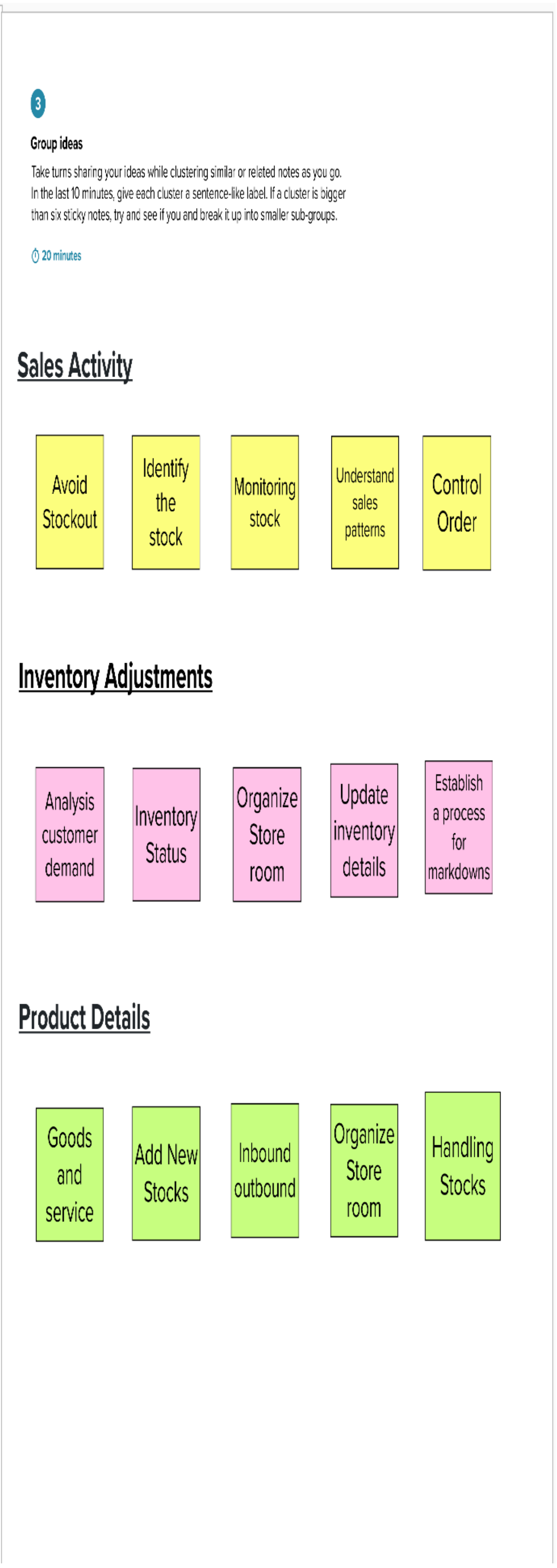
Brainstorm

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!



3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, 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

Sales Activity

Avoid Stockout | Identify the stock | Monitoring stock | Understand sales patterns | Control Order

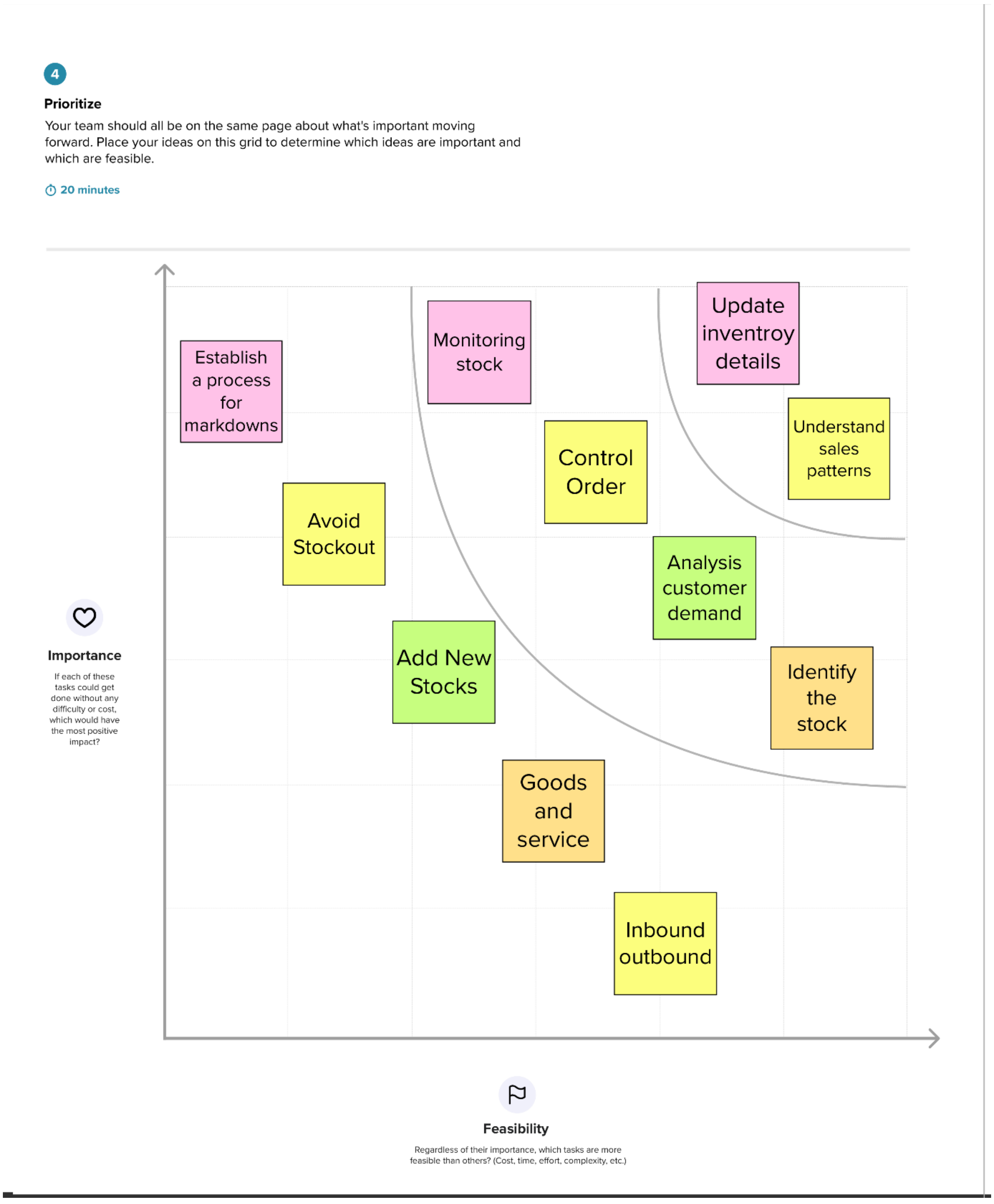
Inventory Adjustments

Analysis customer demand | Inventory Status | Organize Store room | Update inventory details | Establish a process for markdowns

Product Details

Goods and service | Add New Stocks | Inbound outbound | Organize Store room | Handling Stocks

Step-3: Idea Prioritization


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

Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

3.3 PROPOSED SOLUTION

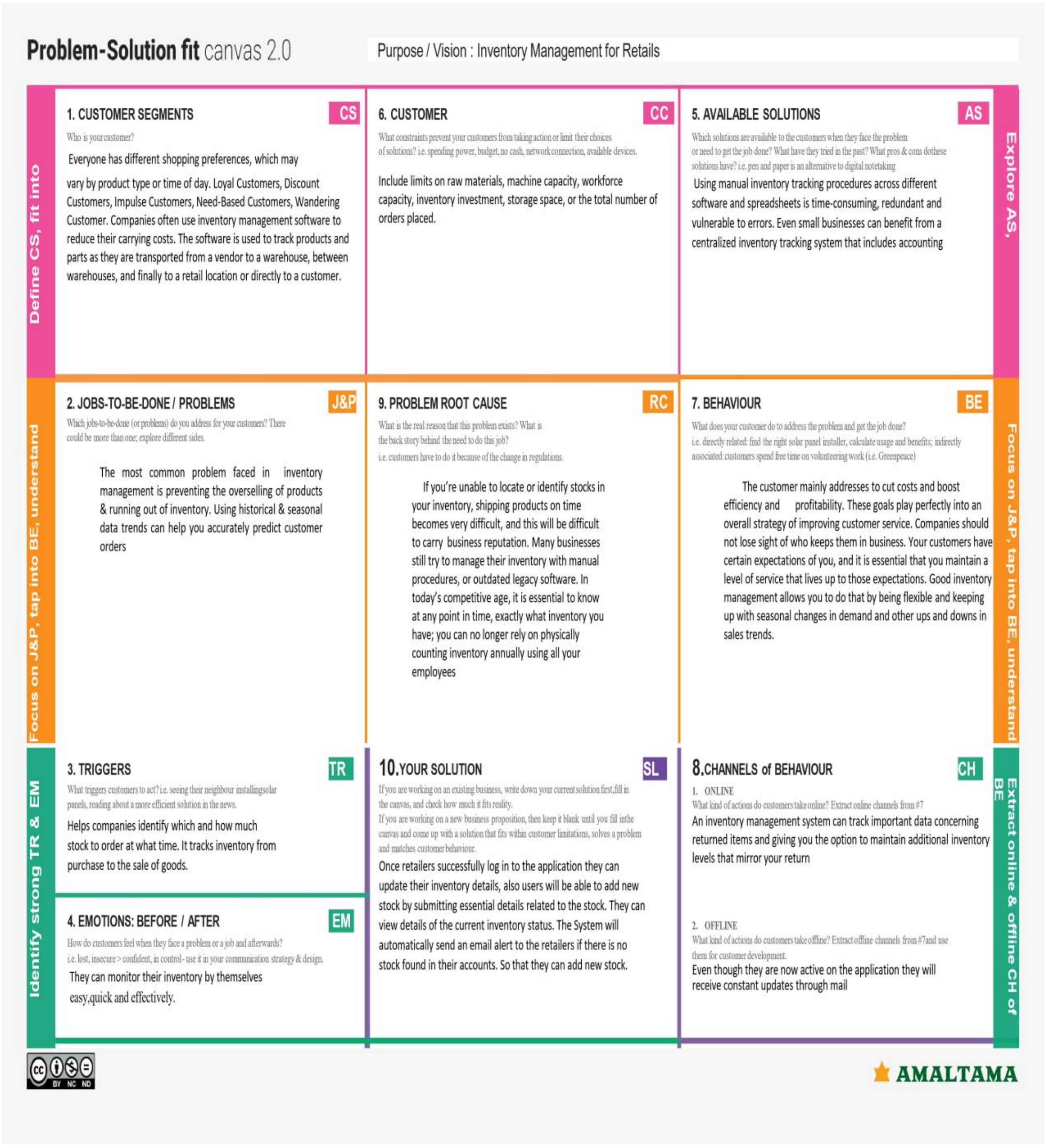
Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. Once retailers successfully log in to the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

S.no	Parameter	Description
1	Problem statement (problem to be solved)	By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.
2	Idea / Solution description	We developed a business model application that provides the inventory details. We built it with a graphical user interface. The applications have been developed to solve the retailer's problem. The application will track the sales pattern of the products in the shop. Managing and updating live stocks, as well as counting the cycle and measurement of products, Users will also be able to add new stocks by submitting essential details related to the stock. If the product is sold out of 75% of its stock, the system will automatically send an email and SMS alert to the user to update the stock, and the retailer can forward the mail to whole sellers to place the order.
3	Novelty / Uniqueness	Our application is developed with many features. They are • Tracking the sales pattern. • Managing and updating live stocks. • Cycle counting of products. • Measurement of products.

4	Social Impact / Customer Satisfaction	Tracking the sales pattern and managing and updating live stocks, counting the cycle and measurement of products. It accelerates retail sales and profits. The retailer can focus on business growth.
5	Business Model (financial Benefit)	• Improve the accuracy of inventory management. • You can save both time and money. • It improves warehouse organisation. • It improves customer retention and engagement. • This ensures more profitability.
6	Scalability of Solution	Analysing inventory details and tracking a sales pattern. Stock management based on customer demand leads to retailer and customer satisfaction.

3.4 PROBLEM SOLUTION FIT

Problem-Solution canvas is a tool for entrepreneurs, marketers and corporate innovators, which helps them identify solutions with higher chances for solution adoption, reduce time spent on solution testing and get a better overview of current situation.



Problem-Solution fit canvas 2.0

Purpose / Vision : Inventory Management for Retails

Define CS, fit into

1. CUSTOMER SEGMENTS CS

Who is your customer?

Everyone has different shopping preferences, which may vary by product type or time of day. Loyal Customers, Discount Customers, Impulse Customers, Need-Based Customers, Wandering Customer. Companies often use inventory management software to reduce their carrying costs. The software is used to track products and parts as they are transported from a vendor to a warehouse, between warehouses, and finally to a retail location or directly to a customer.

6. CUSTOMER CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Include limits on raw materials, machine capacity, workforce capacity, inventory investment, storage space, or the total number of orders placed.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons dothese solutions have? i.e. pen and paper is an alternative to digital notetaking

Using manual inventory tracking procedures across different software and spreadsheets is time-consuming, redundant and vulnerable to errors. Even small businesses can benefit from a centralized inventory tracking system that includes accounting

Explore AS,

Focus on J&P, tap into BE, understand

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

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

The most common problem faced in inventory management is preventing the overselling of products & running out of inventory. Using historical & seasonal data trends can help you accurately predict customer orders

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists? What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

If you're unable to locate or identify stocks in your inventory, shipping products on time becomes very difficult, and this will be difficult to carry business reputation. Many businesses still try to manage their inventory with manual procedures, or outdated legacy software. In today's competitive age, it is essential to know at any point in time, exactly what inventory you have; you can no longer rely on physically counting inventory annually using all your employees

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

The customer mainly addresses to cut costs and boost efficiency and profitability. These goals play perfectly into an overall strategy of improving customer service. Companies should not lose sight of who keeps them in business. Your customers have certain expectations of you, and it is essential that you maintain a level of service that lives up to those expectations. Good inventory management allows you to do that by being flexible and keeping up with seasonal changes in demand and other ups and downs in sales trends.

Focus on J&P, tap into BE, understand

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installingsolar panels, reading about a more efficient solution in the news.

Helps companies identify which and how much stock to order at what time. It tracks inventory from purchase to the sale of goods.

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

They can monitor their inventory by themselves easy,quick and effectively.

10.YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first,fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill inthe canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

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

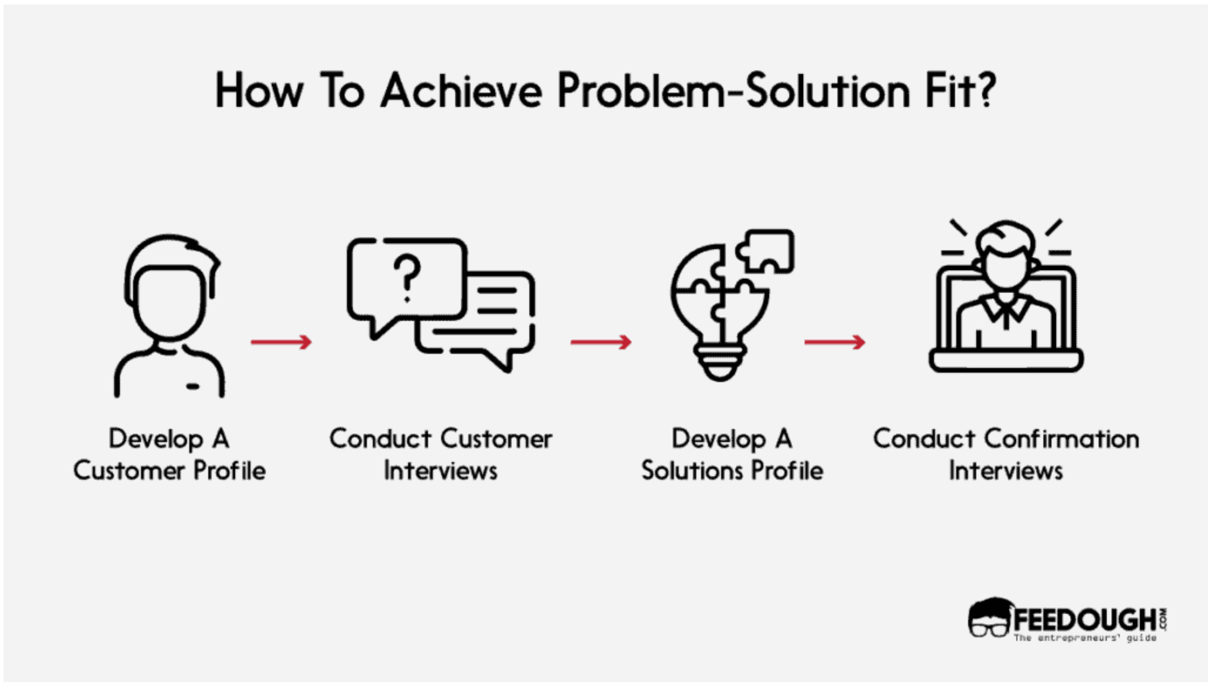
8.CHANNELS of BEHAVIOUR CH

1. ONLINE
What kind of actions do customers take online? Extract online channels from #7
An inventory management system can track important data concerning returned items and giving you the option to maintain additional inventory levels that mirror your return

2. OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7and use them for customer development.
Even though they are now active on the application they will receive constant updates through mail

Extract online & offline CH of BE

AMALTAMA


How To Achieve Problem-Solution Fit?
Develop A Customer Profile
Conduct Customer Interviews
Develop A Solutions Profile
Conduct Confirmation Interviews
FEEDOUGH.COM
The entrepreneurs' guide

4.1 FUNCITIONAL REQUIREMENT

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications.

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	Processing the Transactions	Online payment
FR-4	Authentication	Sent through Email
FR-5	Reporting	Through App (or)Through Email

4.2. NON-FUNCTIONAL REQUIREMENT

Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Tracking the inventory, sales and stock of products.
NFR-2	**Security**	Granting the permission for only authenticated users to access the portal.
NFR-3	**Reliability**	Updating process is fails enables to retrieve the stocks.
NFR-4	**Performance**	Quick access of the product through application.
NFR-5	**Availability**	Arrival of new updates does not impact the application or product details.
NFR-6	**Scalability**	Supports multiple users to access at a time without interference.

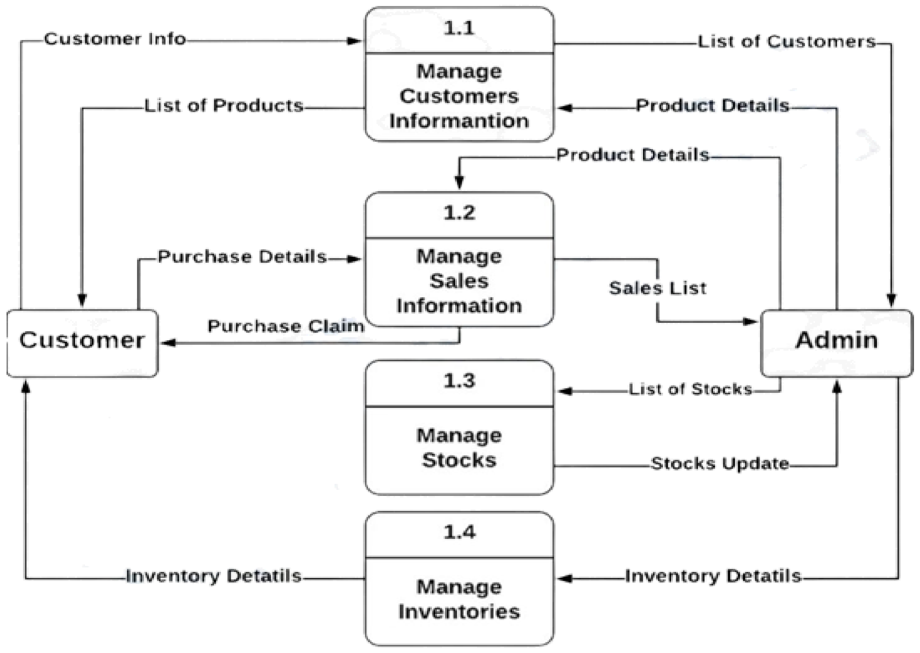
5. PROJECT DESIGN

Project design is an early phase of the project lifecycle where ideas, processes, resources, and deliverables are planned out. A project design comes before a project plan as it's a broad overview whereas a project plan includes more detailed information.

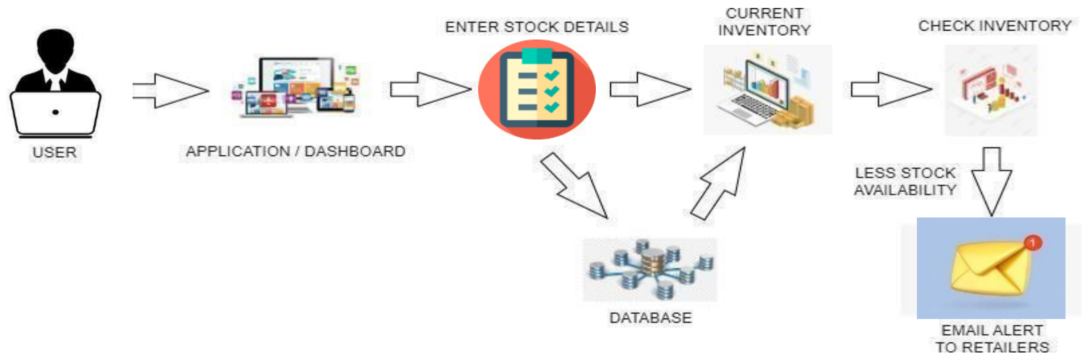
5.1. DATA FLOW DIAGRAMS

A data flow diagram (DFD) is a graphical or visual representation using a standardized set of symbols and notations to describe a business's operations through data movement. They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM).

DATA FLOW DIAGRAMS:



5.2 SOLUTION ARCHITECTURE



5.3 USER STORIES

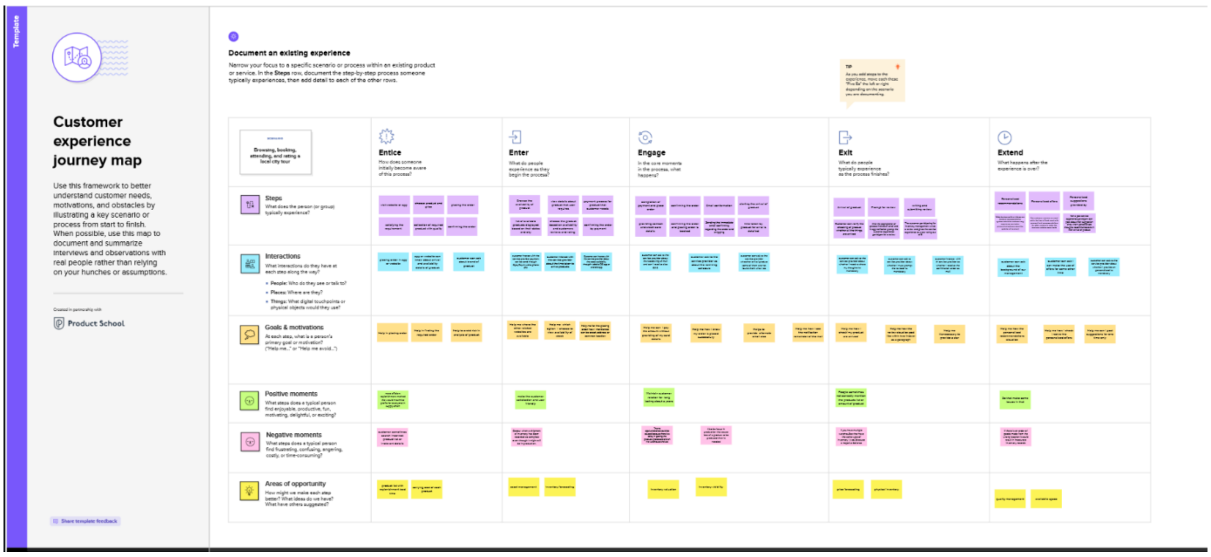
A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer. The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

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		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email and password		High	Sprint-1
User Type	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Acceptance criteria**	**Priority**	**Release**
Customer (Web user)	Login	USN-6	As a user, I can login into application by entering my email and password.	Able to access my account / dashboard	High	Sprint-1
Customer Care Executive		USN-7	It can be used, easily access and responsible	I can access by easily Through application	High	Sprint-1

Administrator		USN-8	As an Administrator I can Update the application	I can fix the bug which Arises for the SFcustomers and users of the application	Medium	Sprint-1

5.4 CUSTOMER JOURNEY MAP

A customer journey map is a visual storyline of every engagement a customer has with a service, brand, or product. The creation of a journey map puts the organization directly in the mind of the consumer, so they can see and understand their customer's processes, needs, and perceptions.



6. PROJECT PLANNING & SCHEDULING

The process of planning primarily deals with selecting the appropriate policies and procedures in order to achieve the objectives of the project. Scheduling converts the project action plans for scope, time cost and quality into an operating timetable.

6.1. SPRINT PLANNING AND ESTIMATION

In Scrum Projects, Estimation is done by the entire team during Sprint Planning Meeting. The objective of the Estimation would be to consider the User Stories for the Sprint by Priority and by the Ability of the team to deliver during the Time Box of the Sprint.

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Story Points**	**Priority**	**Team Members**
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Lekha M MohanKumar S
Sprint-2	Connecting user data to web application	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Kavya R Mohan Kumar S

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Story Points**	**Priority**	**Team Members**
Sprint-3	Connecting web application to object storage	USN-3	As a user, I can register for the application	2	Low	Kavya R Lekha M
Sprint-4	Integrating all the technologies in application	USN-4	As a user, I can register for the application through Gmail	2	Medium	MohanKumar SMohan Kumar S

Testing phase	Analysis of risk	USN-5	As a user, I can log into the application by entering email & password	1	High	Mohan Kumar S Lekha M
	Debugging		Resolving the error		High	Kavya R Mohan Kumar S
	Testing		Testing the application		High	Lekha M MohanKumar S

6.2 SPRINT DELIVERY SCHEDULE

The deliverables of a sprint aren't as predictable as they are for other projects. Sprint participants have produced sketches and drawings, writing, photographs, comic strips, videos and fully coded working prototypes. The answer is whatever's right to answer the problem.

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	5 Nov 2022
Sprint-3	20	6 Days	07Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14Nov 2022	19 Nov 2022	20	19 Nov 2022

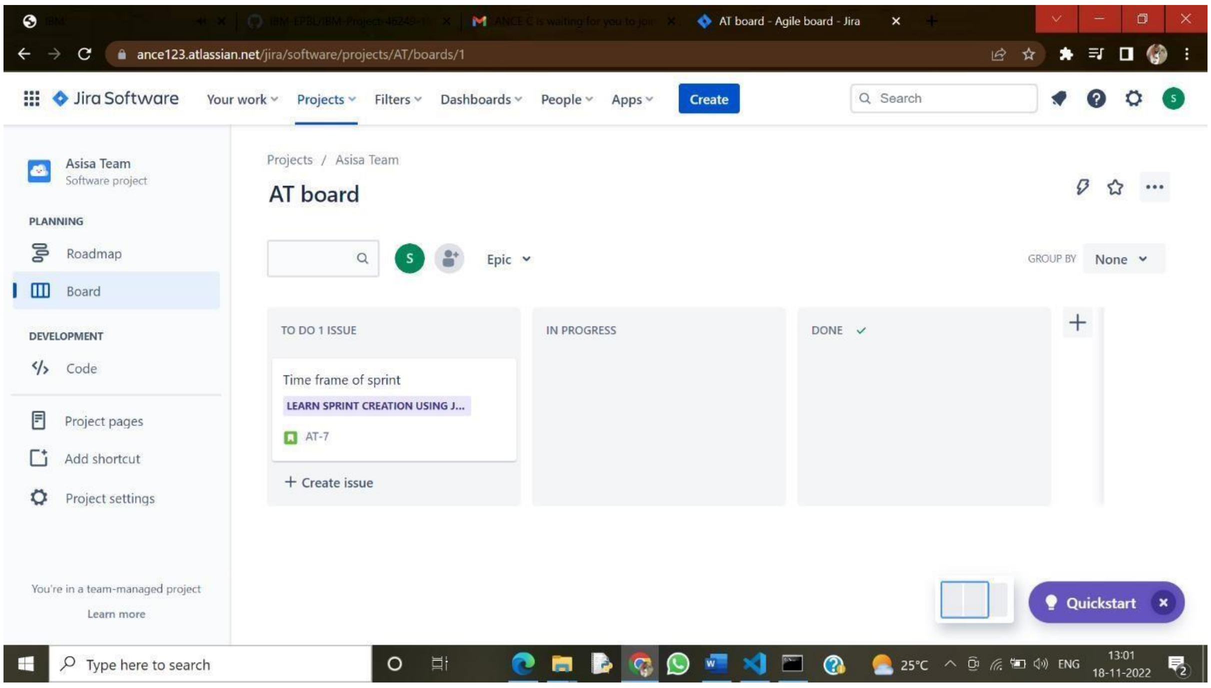
6.3 MILESTONE AND ACTIVITY LIST

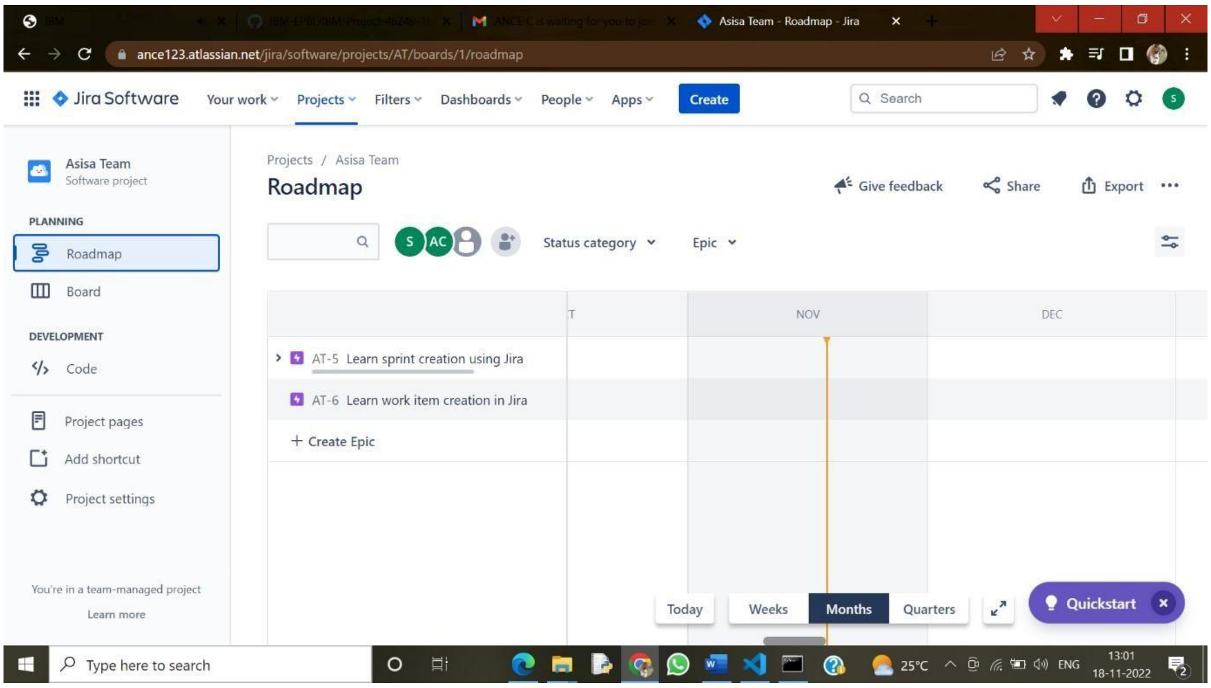
A milestone list is a project management document that identifies all project milestones. A milestone is a significant event or a point in a project. It represents nothing more than a moment in time; hence, when scheduling, milestones should be assigned zero duration.

TITLE	DESCRIPTION	DATE
Literature survey & information gathering	Collect the relevant use cases and refer to existing solutions	19 SEPTEMBER 2022
Prepare empathy map	Prepare Empathy Map canvas and list of problem statements	19 SEPTEMBER 2022
Ideation	List the ideas by organizing the brainstorming session and prioritize the top 3 ideas based on the feasibility & importance	19 SEPTEMBER 2022
Problem solution fit	Prepare problem - solution fit document & solution architecture	07 OCTOBER 2022
Proposed Solution	Preparing the new idea for our problem statement	07 OCTOBER 2022
Customer journey	Prepare the customer journey maps to understand the user interactions & experiences with the application	27 OCTOBER 2022
Solution requirement	Prepare the Functional Requirement Document	27 OCTOBER 2022
Data flow diagrams	Prepare the Data Flow Diagrams	28 OCTOBER 2022

Technology architecture	Prepare Technology Architecture of the solution	29 OCTOBER 2022
Prepare Milestone & activity list	Prepare the Milestone & activity list of the project	30 OCTOBER 2022
Sprint Delivery Plan	Prepare the plan for all the sprints in the project	31 OCTOBER 2022
Project development – delivery of sprint – 1,2,3 & 4	Develop & submit the developed code by testing it	19 November 2022

6.2. REPORT FROM JIRA

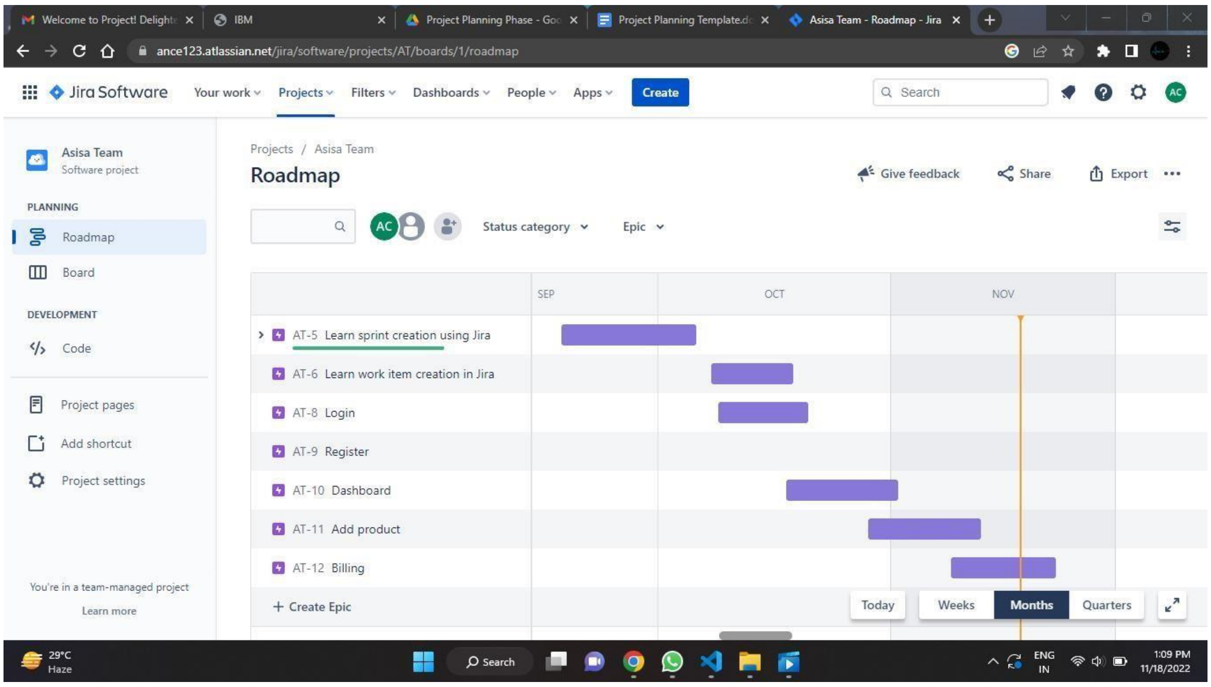



ance123.atlassian.net/jira/software/projects/AT/boards/1/roadmap

Projects / Asisa Team

Roadmap

	NOV	DEC
AT-5 Learn sprint creation using Jira		
AT-6 Learn work item creation in Jira		
+ Create Epic		


ance123.atlassian.net/jira/software/projects/AT/boards/1/roadmap

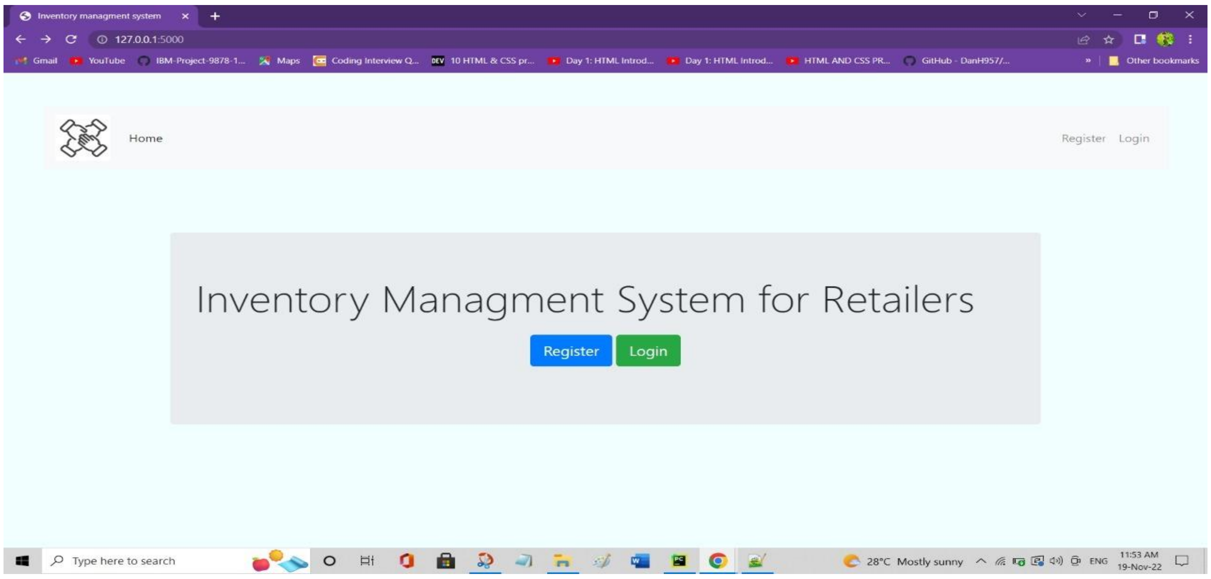
Projects / Asisa Team

Roadmap

	SEP	OCT	NOV
AT-5 Learn sprint creation using Jira			
AT-6 Learn work item creation in Jira			
AT-8 Login			
AT-9 Register			
AT-10 Dashboard			
AT-11 Add product			
AT-12 Billing			
+ Create Epic			

7. CODING & SOLUTIONING

HOME PAGE



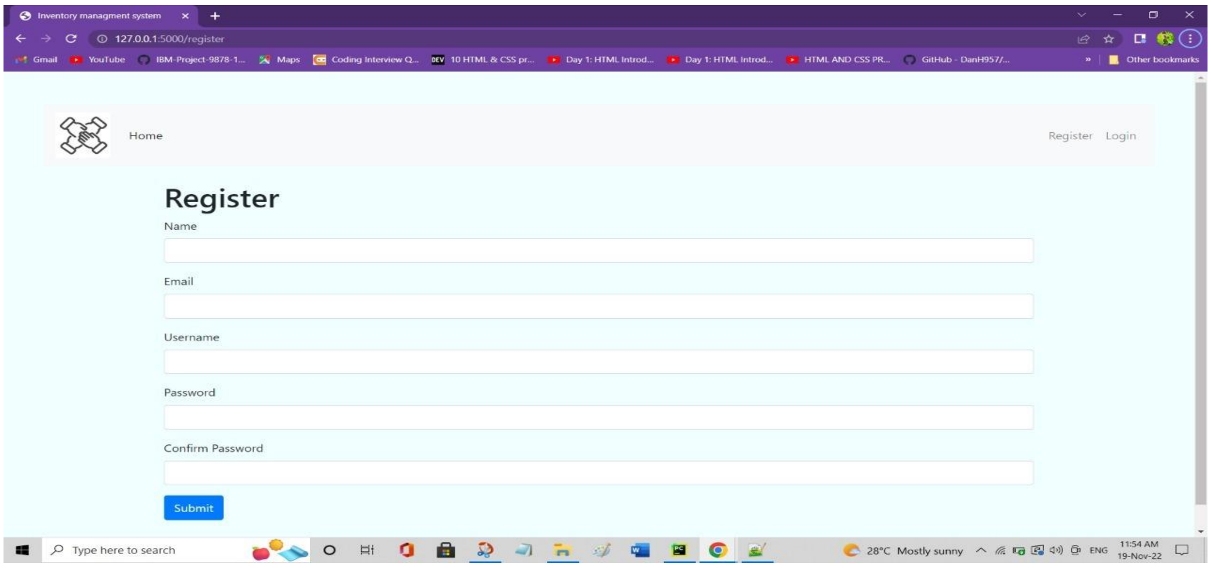
Inventory Managment System for Retailers

Register Login

```
{% extends 'layout.html' %}
{% block body%}
<style>
   body {
     background-image: <link rel="stylesheet" href="{{url_for('static',
filename='css/style.css')}}">
   }
   </style><br><br>
<div class="jumbotron mt-4">
   <h7 class="display-4">Inventory Managment System for
Retailers</h7><br><br>
   {% if session.logged_in == NULL %}
      <center><a href="/register" class="btn btn-primary btn-lg">Register</a>
      <a href="/login" class="btn btn-success btn-lg">Login</a></center>
   {% endif %}<br>
</div>
{% endblock %}
```

7.1. FEARTURE 1

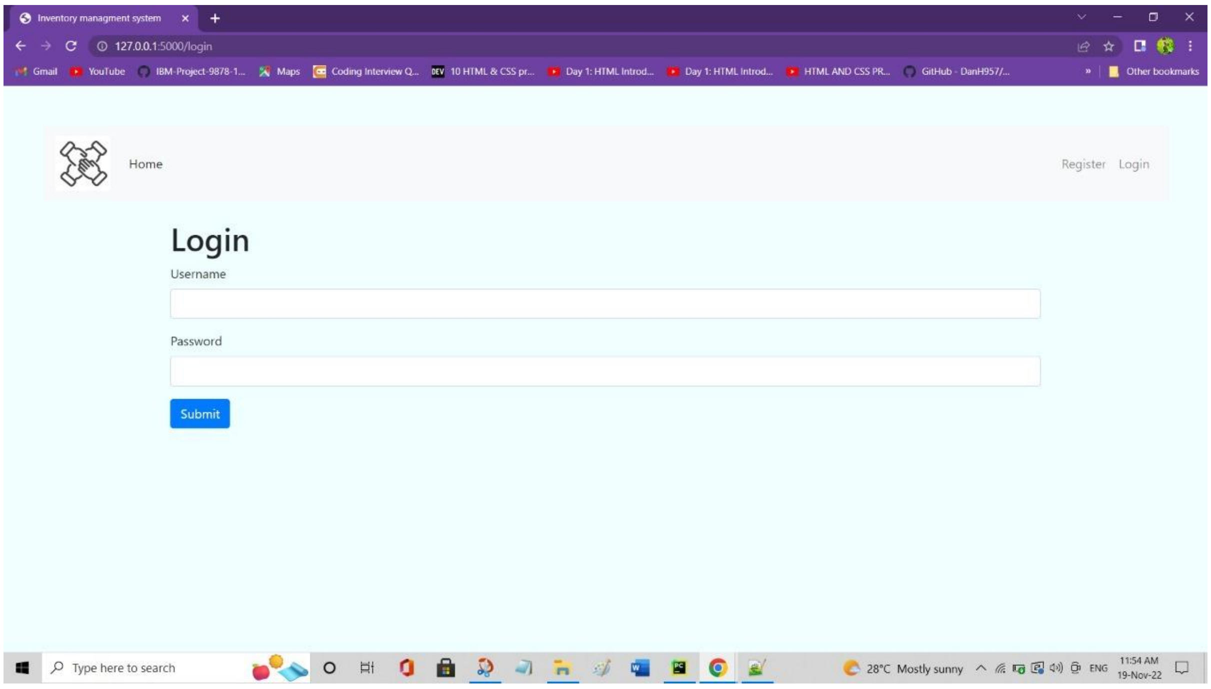
REGISTER PAGE



```
<html>

   <head>

      <meta charset="UTF-8">

      <title> Register </title>

      <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">

   </head>

   <body></br></br></br></br></br>

      <div align="center">

        <div align="center" class="border">

           <div class="header">

              <h1 class="word">Register</h1>
```

```
            </div></br></br></br>
          <h2 class="word">
              <form action="{{ url_for('register') }}" method="post">
                <div class="msg">{{ msg }}</div>
                  <input id="username" name="username" type="text" placeholder="Enter Your Username" class="textbox"/></br></br>
                  <input id="phone_num" name="phone_num" type="text" placeholder="Enter Your PhoneNumber" class="textbox"/></br></br>
                  <input id="email" name="email" type="text" placeholder="Enter Your Email ID" class="textbox"/></br></br>
                  <input type="submit" class="btn" value="Register"></br>
              </form>
          </h2>


        </div>
      </div>
   </body>
</html>
```

LOGIN PAGE

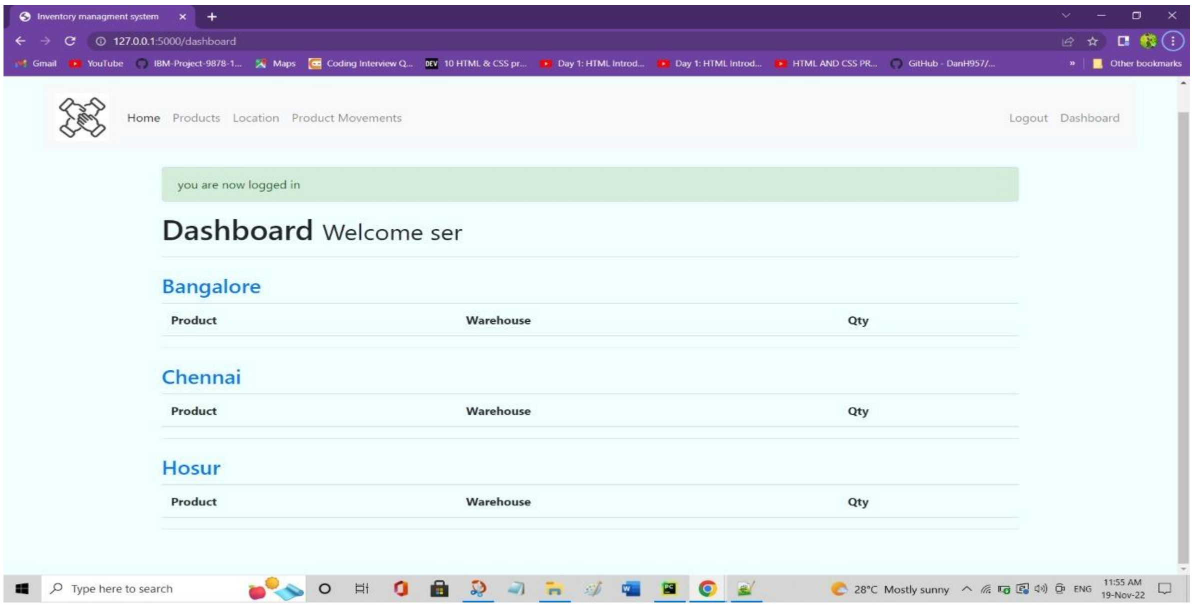


```
<html>

   <head>

      <meta charset="UTF-8">

      <title> Login </title>

      <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

   </head>

   <body></br></br></br></br></br>

      <div align="center">

        <div align="center" class="border">

           <div class="header">
```

```
                <h1 class="word">Login</h1>

              </div></br></br></br>

             <h2 class="word">

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

                  <div class="msg">{{ msg }}</div>

                    <input id="username" name="username" type="text"
placeholder="Enter Your Username" class="textbox"/></br></br>

                    <input id="password" name="password"
type="password" placeholder="Enter Your Password"
class="textbox"/></br></br></br>

                    <input type="submit" class="btn" value="Sign
In"></br></br>

                </form>

             </h2>

             <p class="bottom">Don't have an account?  <a class="bottom"
href="{{url_for('register')}}"> Sign Up here</a></p>

           </div>

         </div>

    </body>

</html>
```

7.2. FEARTURE 2

DASHBOARD



you are now logged in

Dashboard Welcome ser

Bangalore

Product	Warehouse	Qty

Chennai

Product	Warehouse	Qty

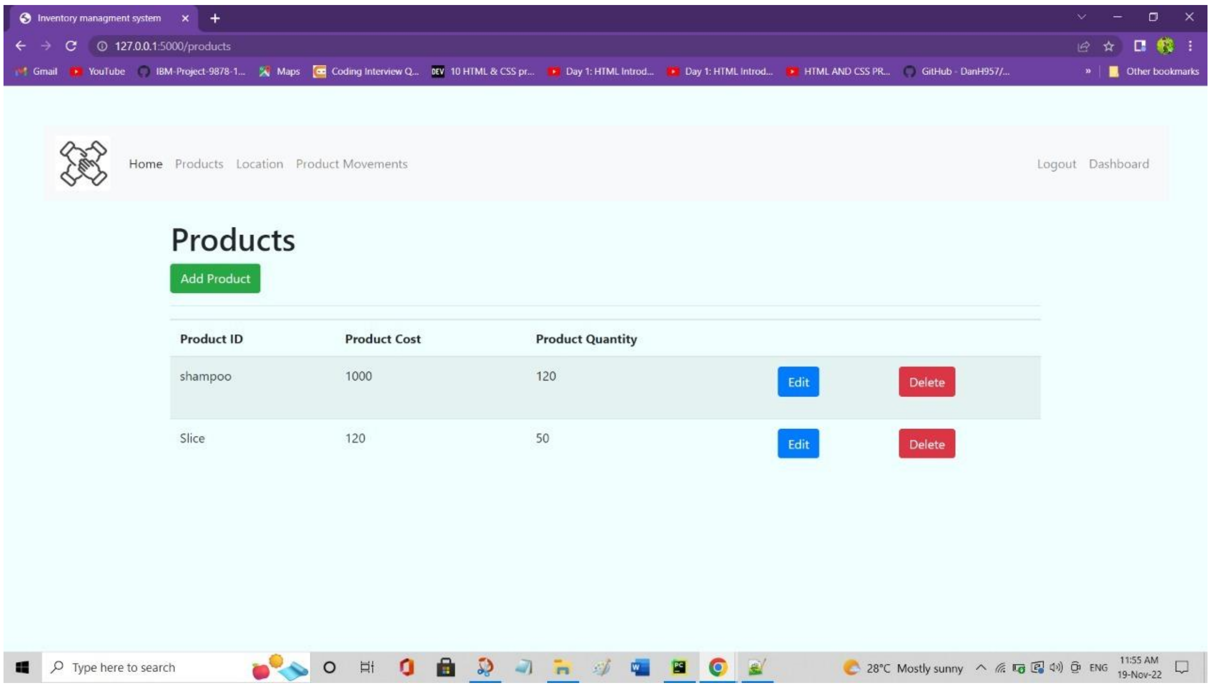
Hosur

Product	Warehouse	Qty

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


{% block body %}
    <h1>Dashboard <small>Welcome {{session.username}}</small></h1>
    <hr>
      {% for location in locations %}
      <div>
       <h3 class="mt-4 text-primary" >{{location}}</h3>
       <table class="table table-striped">
           <thead>
               <tr>
                   <th>Product</th>
                   <th>Warehouse</th>
                   <th>Qty</th>
               </tr>
           </thead>
```

```
            <tbody>
                {% for product in products %}
                   {% if product.LOCATION_ID == location %}
               <tr>
                <td>{{product.PRODUCT_ID}}</td>
                <td>{{product.LOCATION_ID}}</td>
                <td>{{product.QTY}}</td>
               </tr>
                   {% endif %}
               {% endfor %}
            </tbody>
          </table>
          <hr>
          </div>
      {% endfor %}
  {% endblock %}
```

PRODUCT


Inventory managment system
127.0.0.1:5000/products
Home Products Location Product Movements
Logout Dashboard

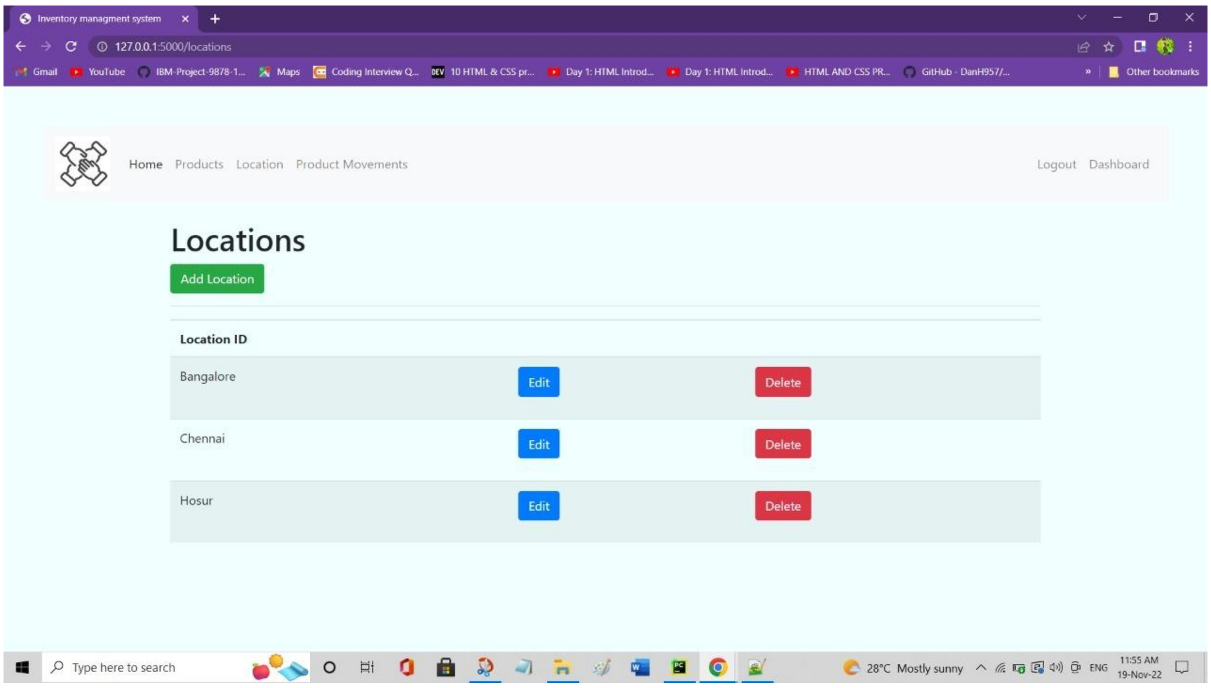
Products

Add Product

Product ID	Product Cost	Product Quantity		
shampoo	1000	120	Edit	Delete
Slice	120	50	Edit	Delete

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

{% block body %}
    <h1>Products</h1>
    <a class="btn btn-success" href="/add_product">Add Product</a>
    <hr>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Product ID</th>
                <th>Product Cost</th>
                <th>Product Quantity</th>
                <th></th>
                <th></th>
            </tr>
        </thead>
        <tbody>
                {% for product in products %}
               <tr>
                <td>{{product.PRODUCT_ID}}</td>
                <td>{{product.PRODUCT_COST}}</td>
                <td>{{product.PRODUCT_NUM}}</td>
                <td><a href="edit_product/{{product.PRODUCT_ID}}" class="btn
btn-primary pull-right">Edit</a></td>
                <td>
                    <form action="{{url_for('delete_product',
id=product.PRODUCT_ID)}}" method="POST">
                        <input type="hidden" name="method" value="DELETE">
                        <input type="submit" value="Delete" class="btn btn-danger">
                </form>
              </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

LOCATION



Locations

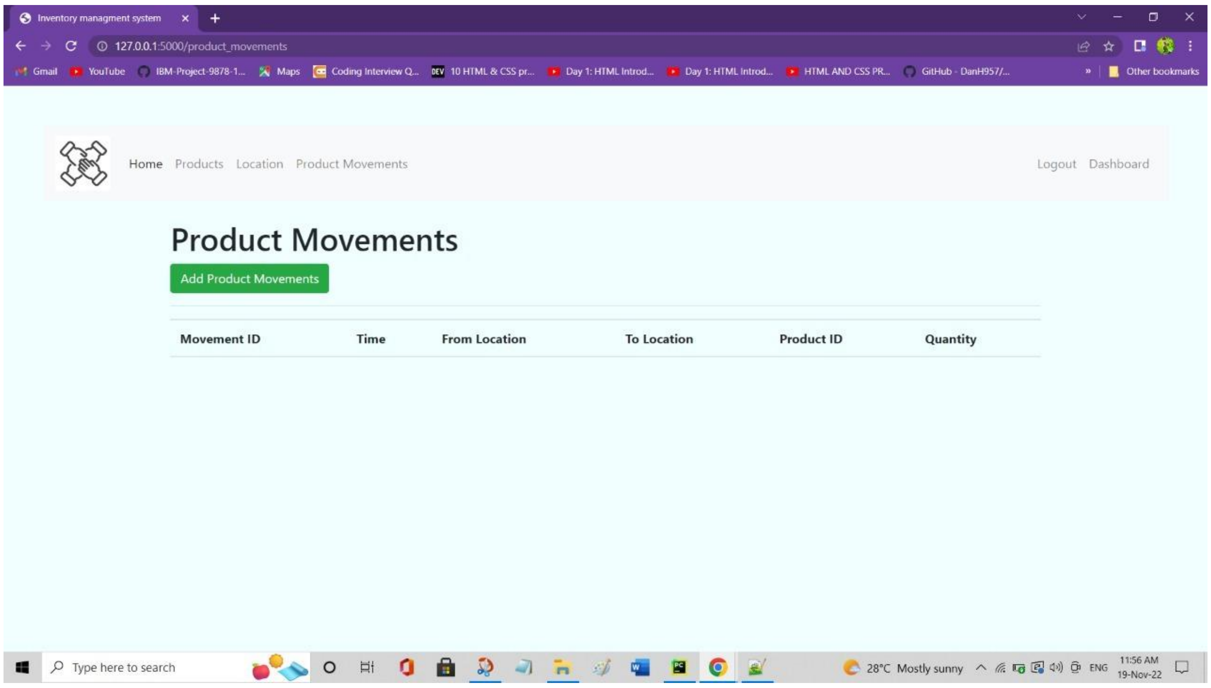
Add Location

Location ID		
Bangalore	Edit	Delete
Chennai	Edit	Delete
Hosur	Edit	Delete

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

{% block body %}
    <h1>Locations</h1>
    <a class="btn btn-success" href="/add_location">Add Location</a>
    <hr>
    <table class="table table-striped">
        <thead>
```

```
            <tr>
                <th>Location ID</th>
                <th></th>
                <th></th>
            </tr>
        </thead>
        <tbody>
                {% for location in locations %}
               <tr>
                <td>{{location.LOCATION_ID}}</td>
                <td><a href="edit_location/{{location.LOCATION_ID}}"
class="btn btn-primary pull-right">Edit</a></td>
                <td>
                    <form action="{{url_for('delete_location',
id=location.LOCATION_ID)}}" method="POST">
                       <input type="hidden" name="method" value="DELETE">
                       <input type="submit" value="Delete" class="btn btn-danger">
                    </form>
                </td>
               </tr>
               {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

PRODUCT MOVEMENT



Product Movements

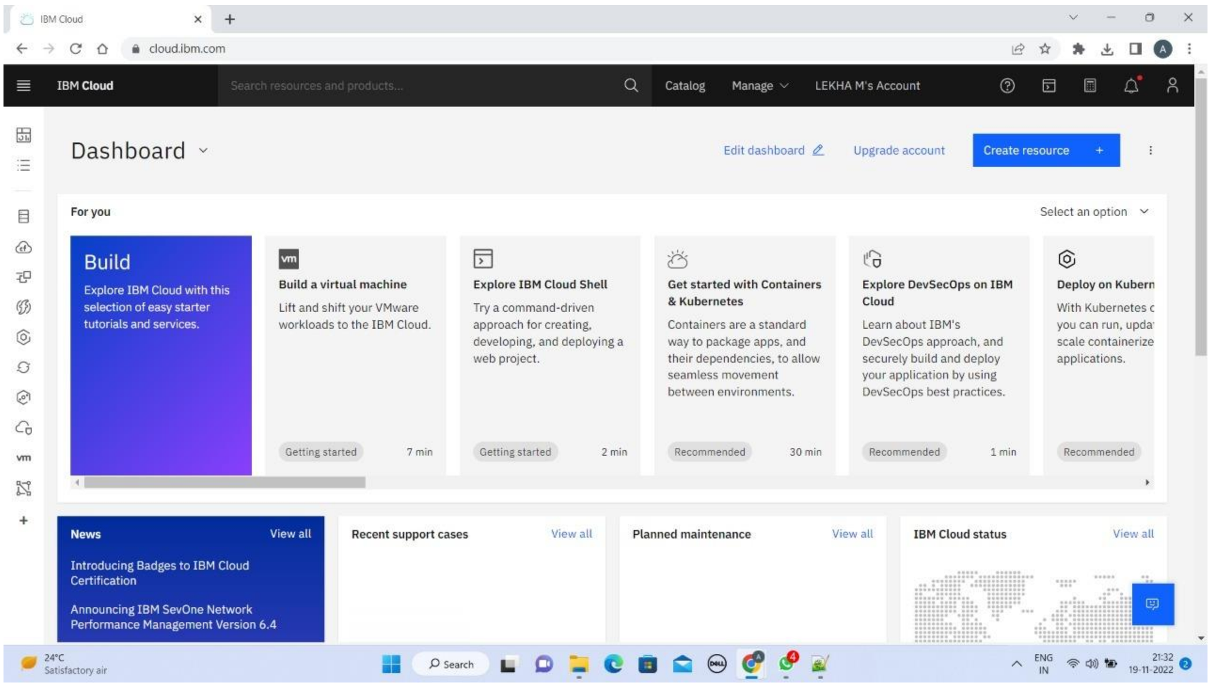
Movement ID	Time	From Location	To Location	Product ID	Quantity

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


{% block body %}
    <h1>Product Movements</h1>
    <a class="btn btn-success" href="/add_product_movements">Add Product
Movements</a>
    <hr>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Movement ID</th>
                <th>Time</th>
                <th>From Location</th>
                <th>To Location</th>
                <th>Product ID</th>
                <th>Quantity</th>
            </tr>
        </thead>
        <tbody>
                {% for movement in movements %}
               <tr>
                <td>{{movement.MOVEMENT_ID}}</td>
                <td>{{movement.TIME}}</td>
                <td>{{movement.FROM_LOCATION}}</td>
                <td>{{movement.TO_LOCATION}}</td>
                <td>{{movement.PRODUCT_ID}}</td>
                <td>{{movement.QTY}}</td>
                <!--<td><a
href="edit_product_movement/{{movement.MOVEMENT_ID}}" class="btn
btn-primary pull-right">Edit</a></td>-->
                <td>
                    <form action="{{url_for('delete_product_movements',
id=movement.MOVEMENT_ID)}}" method="POST">
                        <input type="hidden" name="method" value="DELETE">
                        <input type="submit" value="Delete" class="btn btn-danger">
                    </form>
                </td>
               </tr>
               {% endfor %}
        </tbody>
    </table>
{% endblock %}


{% extends 'layout.html' %}
{% block body %}
<h1>Edit Product Movements</h1>
{% from "includes/_formhelpers.html" import render_field %}
<form action="" method="POST">
   <div class="form-group">
      {{ render_field(form.from_location, class_="form-control") }}
   </div>
   <div class="form-group">
      {{ render_field(form.to_location, class_="form-control") }}
   </div>
   <div class="form-group">
      {{ render_field(form.product_id, class_="form-control") }}
   </div>
   <div class="form-group">
      {{ render_field(form.qty, class_="form-control") }}
   </div>
   <p><input type="submit" value="Update" class="btn btn-primary"></p>
</form>
{% endblock %}


{% extends 'layout.html' %}


{% block body %}
<h1>Add Product Movements</h1>
{% from "includes/_formhelpers.html" import render_field %}
<form action="" method="POST">
   <div class="form-group">
```

```
        {{ render_field(form.from_location, class_="form-control") }}
    </div>
    <div class="form-group">
        {{ render_field(form.to_location, class_="form-control") }}
    </div>
    <div class="form-group">
        {{ render_field(form.product_id, class_="form-control") }}
    </div>
    <div class="form-group">
        {{ render_field(form.qty, class_="form-control", type="number") }}
    </div>
    <p><input type="submit" value="Add" class="btn btn-primary"></p>
</form>
{% endblock %}
```

7.3 DATABASE SCHEMA

CLOUD ACCOUNT CREATION



Build
Explore IBM Cloud with this selection of easy starter tutorials and services.

Build a virtual machine
Lift and shift your VMware workloads to the IBM Cloud.

Explore IBM Cloud Shell
Try a command-driven approach for creating, developing, and deploying a web project.

Get started with Containers & Kubernetes
Containers are a standard way to package apps, and their dependencies, to allow seamless movement between environments.

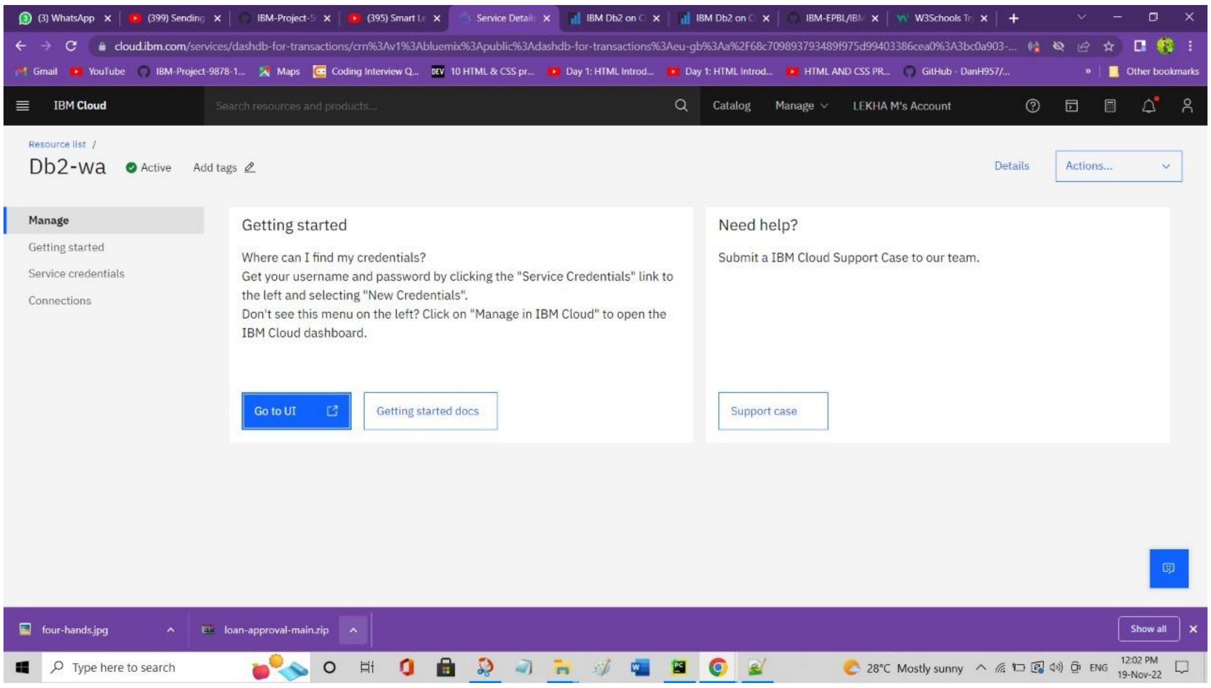
Explore DevSecOps on IBM Cloud
Learn about IBM's DevSecOps approach, and securely build and deploy your application by using DevSecOps best practices.

Deploy on Kubern
With Kubernetes c you can run, upda scale containerize applications.

Introducing Badges to IBM Cloud Certification

Announcing IBM SevOne Network Performance Management Version 6.4

DB2 CREATION



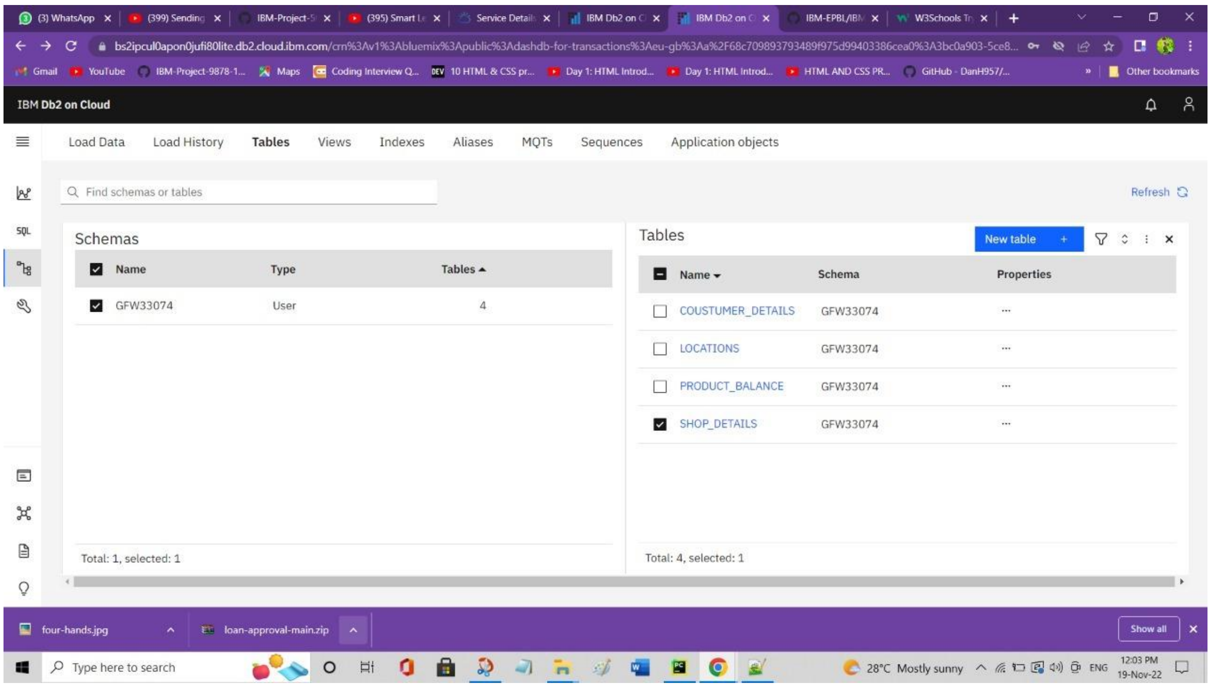
Getting started

Where can I find my credentials?
Get your username and password by clicking the "Service Credentials" link to the left and selecting "New Credentials".
Don't see this menu on the left? Click on "Manage in IBM Cloud" to open the IBM Cloud dashboard.

Need help?

Submit a IBM Cloud Support Case to our team.

DB2 CONNECTION



Schemas

Name	Type	Tables
GFW33074	User	4

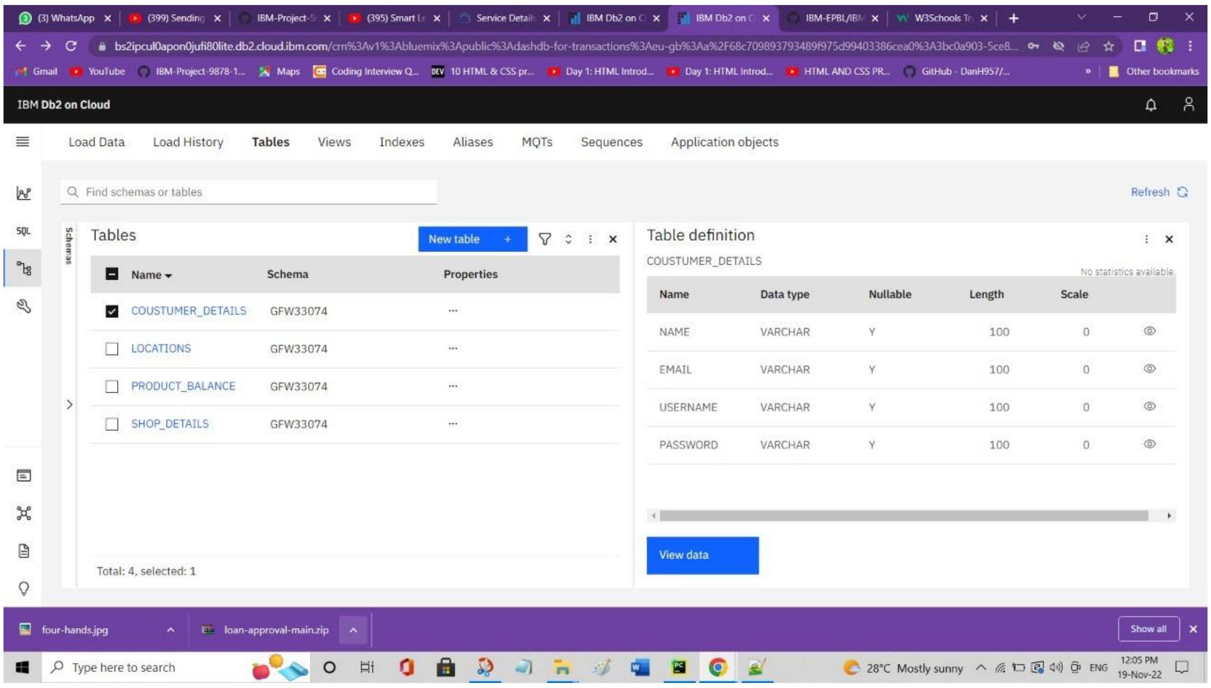
Total: 1, selected: 1

Tables

Name	Schema	Properties
COUSTUMER_DETAILS	GFW33074	...
LOCATIONS	GFW33074	...
PRODUCT_BALANCE	GFW33074	...
SHOP_DETAILS	GFW33074	...

Total: 4, selected: 1

CUDTOMER DETAILS



Tables

Name	Schema	Properties
☑ COUSTUMER_DETAILS	GFW33074	...
☐ LOCATIONS	GFW33074	...
☐ PRODUCT_BALANCE	GFW33074	...
☐ SHOP_DETAILS	GFW33074	...

Total: 4, selected: 1

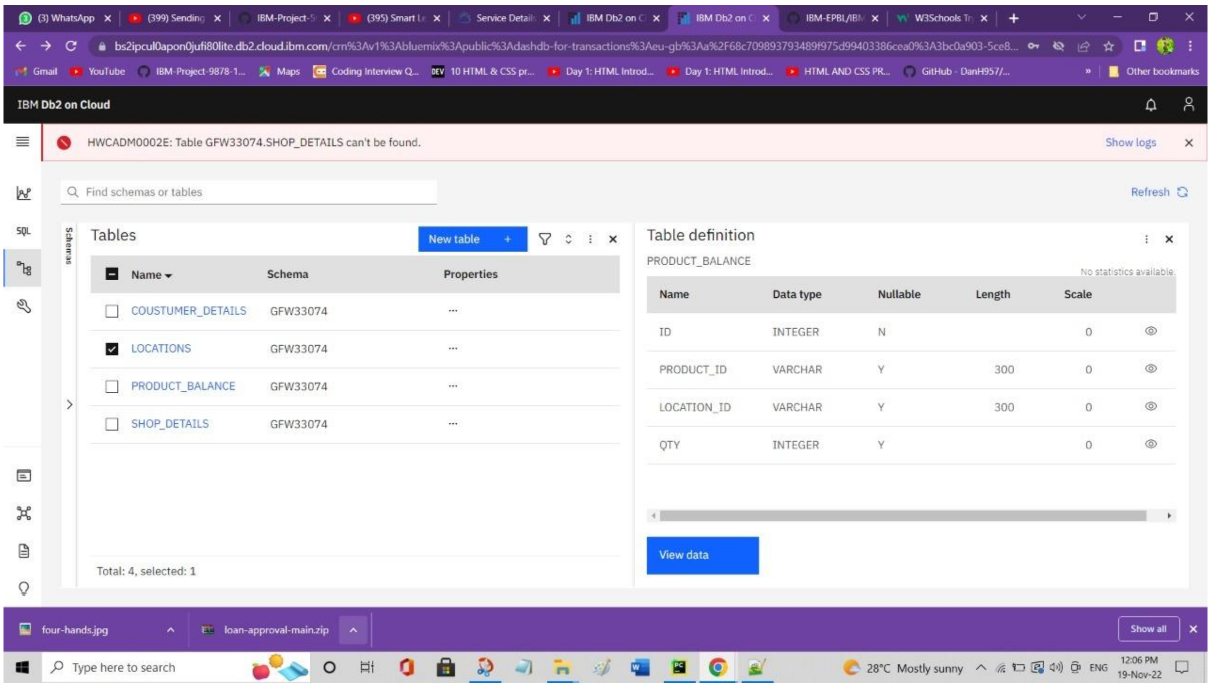
Table definition

COUSTUMER_DETAILS

No statistics available

Name	Data type	Nullable	Length	Scale
NAME	VARCHAR	Y	100	0
EMAIL	VARCHAR	Y	100	0
USERNAME	VARCHAR	Y	100	0
PASSWORD	VARCHAR	Y	100	0

LOCATION



HWCADM0002E: Table GFW33074.SHOP_DETAILS can't be found.

Tables

Name	Schema	Properties
☐ COUSTUMER_DETAILS	GFW33074	...
☑ LOCATIONS	GFW33074	...
☐ PRODUCT_BALANCE	GFW33074	...
☐ SHOP_DETAILS	GFW33074	...

Total: 4, selected: 1

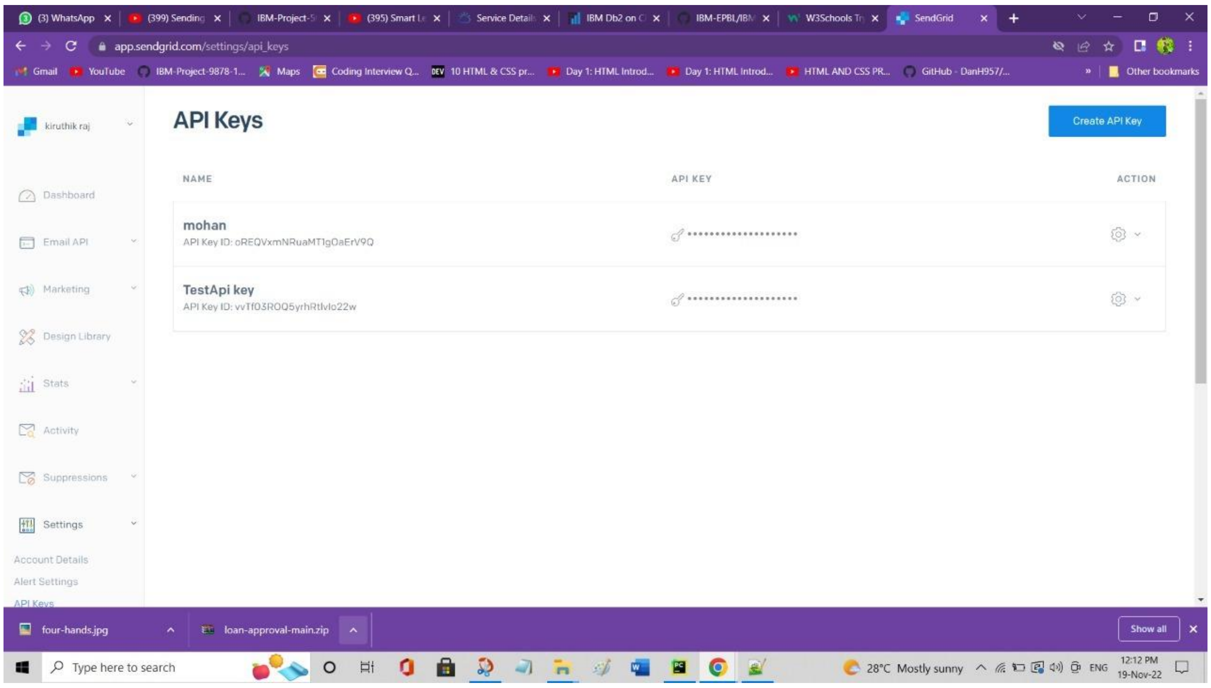
Table definition

PRODUCT_BALANCE

No statistics available

Name	Data type	Nullable	Length	Scale
ID	INTEGER	N		0
PRODUCT_ID	VARCHAR	Y	300	0
LOCATION_ID	VARCHAR	Y	300	0
QTY	INTEGER	Y		0

SENDGRID INTEGRATION



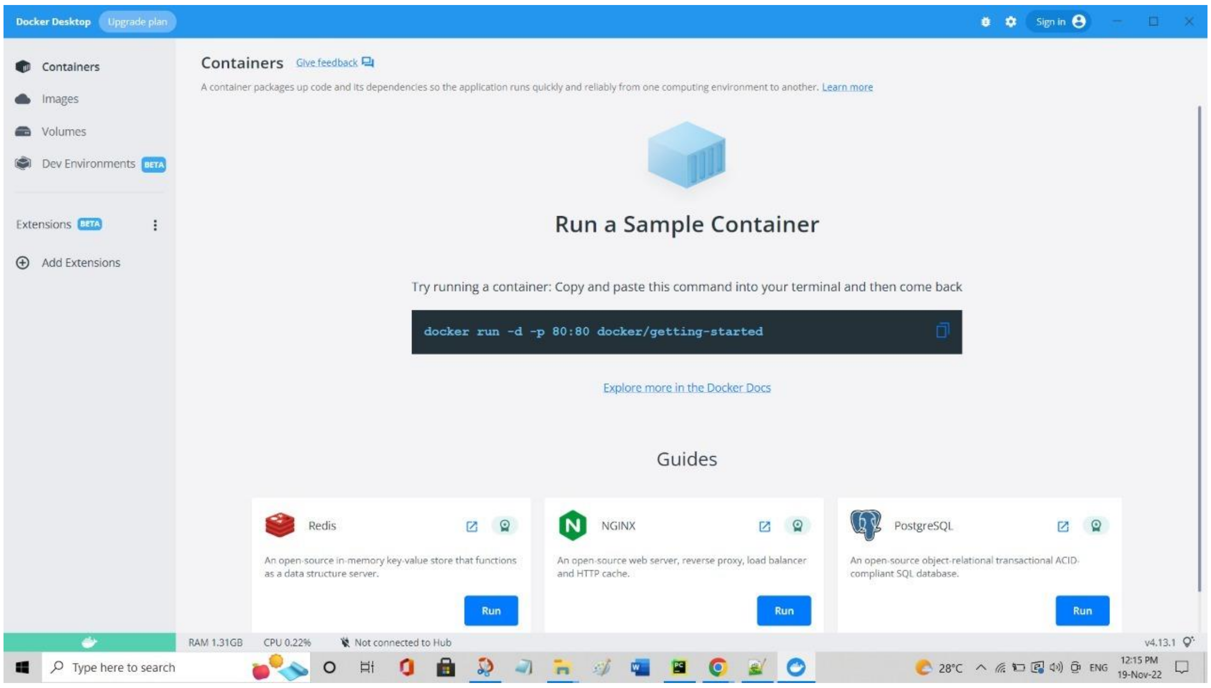
app.sendgrid.com/settings/api_keys

API Keys

NAME	API KEY	ACTION
mohan API Key ID: oREQVxmNRuaMT1gOaErV9Q	••••••••••••••••••••	
TestApi key API Key ID: vvTf03ROQ5yrhRtlvlo22w	••••••••••••••••••••	

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

def alert(main_msg):
   mail_from = 'adhiyamaan@clg.com'
   mail_to ="lekham025@gmail.com"
   msg = MIMEMultipart()
   msg['From'] = mail_from
   msg['To'] = mail_to
   msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
   mail_body = main_msg
   msg.attach(MIMEText(mail_body))

   try:
      server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)
      server.ehlo()
      # SENDGRID_APIKEY=oREQVxmNRuaMT1gOaErV9Q
      server.login('apikey', 'oREQVxmNRuaMT1gOaErV9Q')
      server.sendmail(mail_from, mail_to, msg.as_string())
      server.close()
      print("Mail sent successfully!")
   except:
      print("Some Issue, Mail not Sent :(")
```

CONTAIER INTEGRATION



Docker Desktop Upgrade plan | Sign in

Containers
Images
Volumes
Dev Environments BETA
Extensions BETA
Add Extensions

Containers Give feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Run a Sample Container

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

Explore more in the Docker Docs

Guides

Redis — An open-source in-memory key-value store that functions as a data structure server. Run

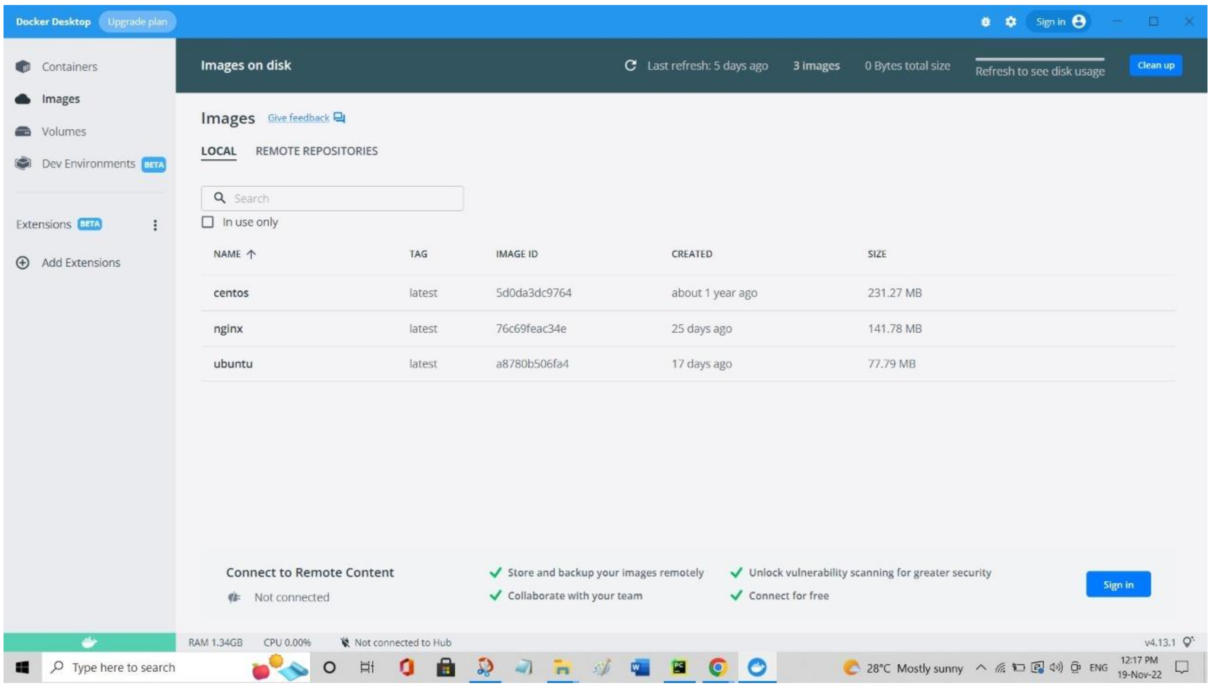
NGINX — An open-source web server, reverse proxy, load balancer and HTTP cache. Run

PostgreSQL — An open-source object-relational transactional ACID-compliant SQL database. Run

RAM 1.31GB CPU 0.22% Not connected to Hub v4.13.1

Type here to search 28°C ENG 12:15 PM 19-Nov-22

CONTAINER IMAGE DISK



Docker Desktop Upgrade plan | Sign in

Containers
Images
Volumes
Dev Environments BETA
Extensions BETA
Add Extensions

Images on disk | Last refresh: 5 days ago | 3 images | 0 Bytes total size | Refresh to see disk usage | Clean up

Images Give feedback

LOCAL REMOTE REPOSITORIES

Search

In use only

NAME	TAG	IMAGE ID	CREATED	SIZE
centos	latest	5d0da3dc9764	about 1 year ago	231.27 MB
nginx	latest	76c69feac34e	25 days ago	141.78 MB
ubuntu	latest	a8780b506fa4	17 days ago	77.79 MB

Connect to Remote Content — Not connected
✓ Store and backup your images remotely
✓ Collaborate with your team
✓ Unlock vulnerability scanning for greater security
✓ Connect for free
Sign in

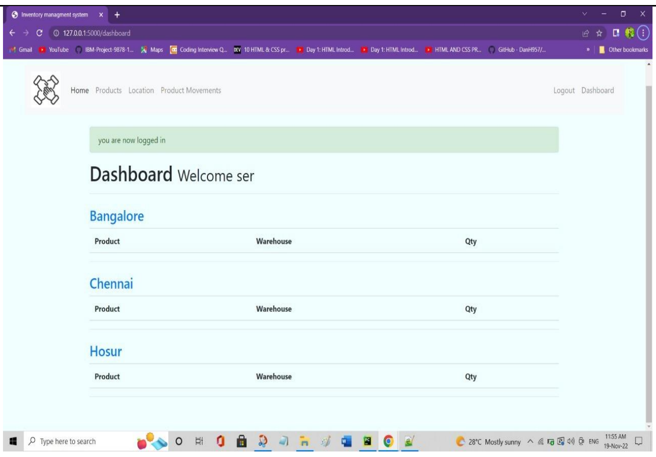
RAM 1.34GB CPU 0.00% Not connected to Hub v4.13.1

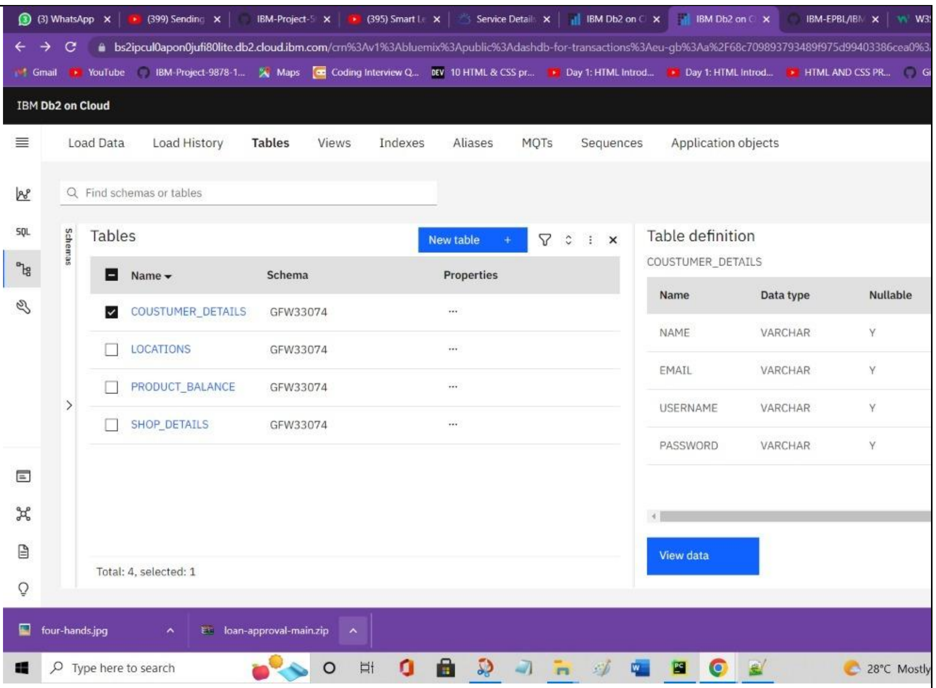
Type here to search 28°C Mostly sunny ENG 12:17 PM 19-Nov-22

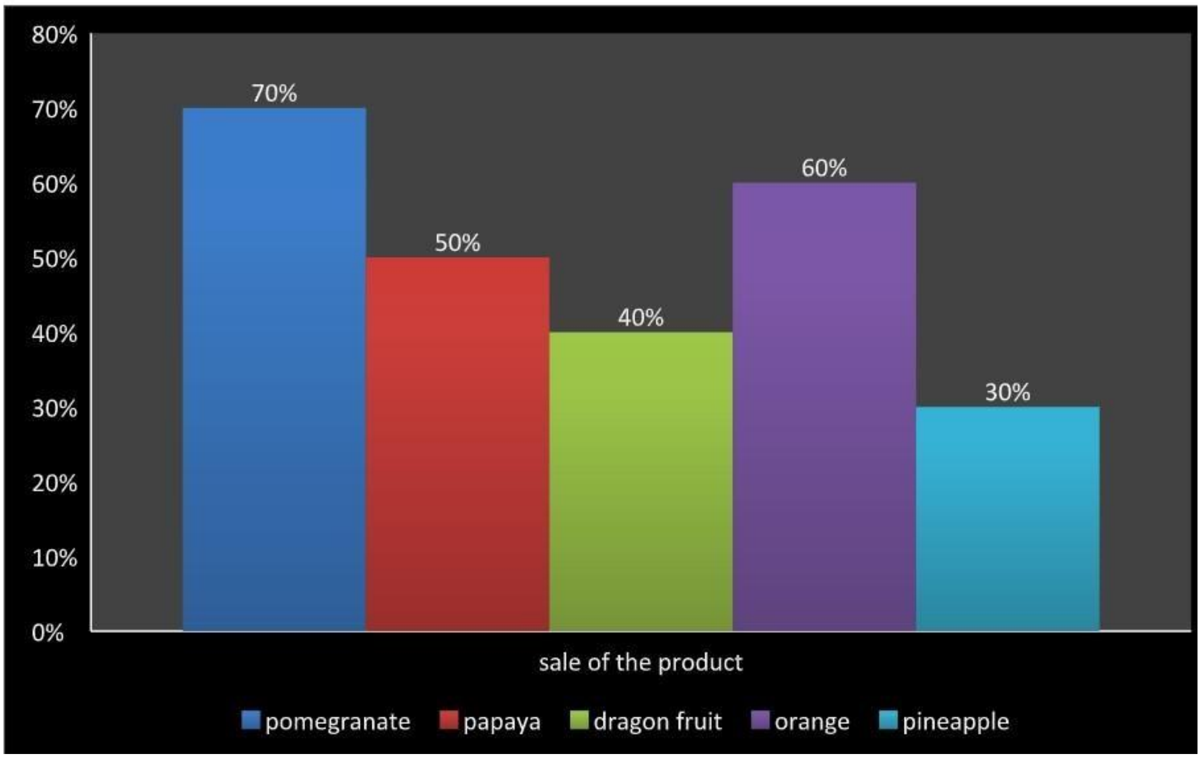
8.TESTING

In general, testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met.

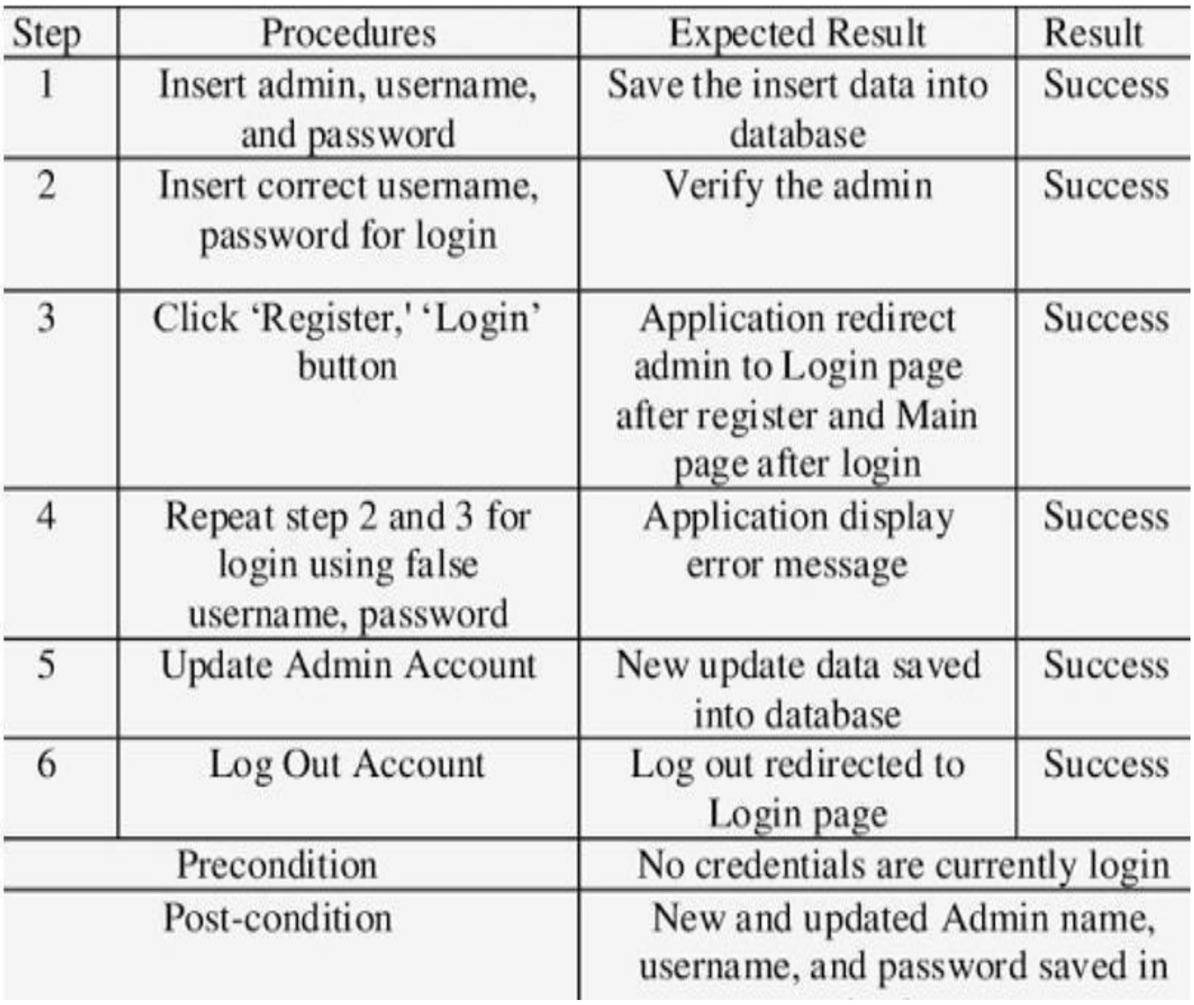
8.1. TEST CASE

S.No	**Parameter**	**Screenshot / Values**
1.	Dashboard design	 Inventory managment system 127.0.0.1:5000/dashboard Home Products Location Product Movements Logout Dashboard you are now logged in Dashboard Welcome ser Bangalore Product Warehouse Qty Chennai Product Warehouse Qty Hosur Product Warehouse Qty
2.	Data Responsiveness	• Responds quickly to unpredictable damand. • Have a lower utilization rate. • Utilise an excess buffer capacity. Invest in lead time reduction.
3.	Amount Data to Rendered (DB2 Metrics)	**7**

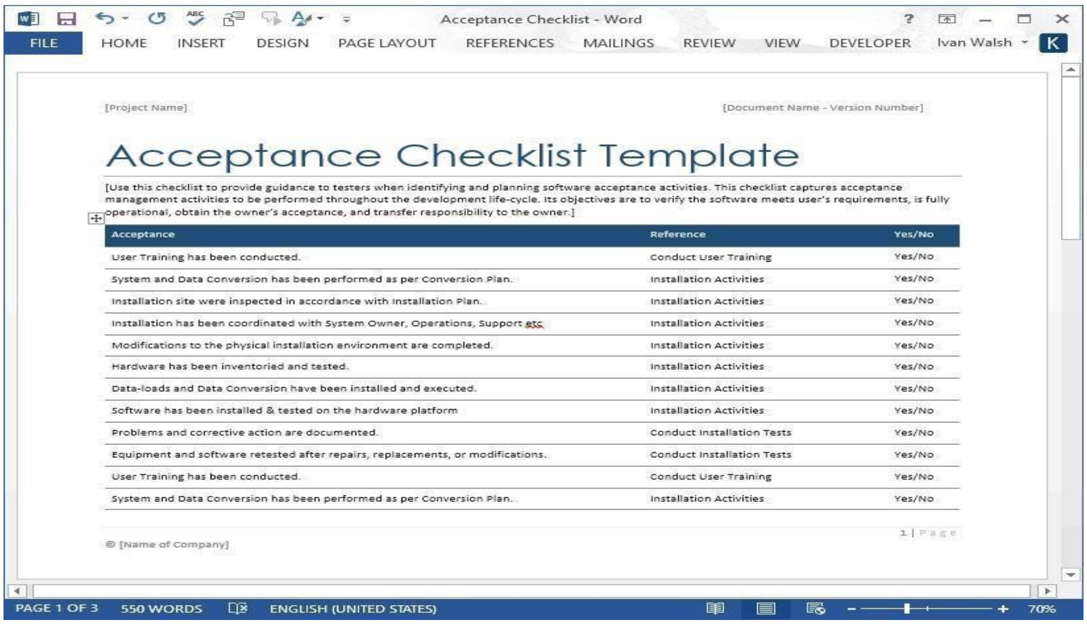
4.	Utilization of Data Filters	 IBM Db2 on Cloud Load Data · Load History · Tables · Views · Indexes · Aliases · MQTs · Sequences · Application objects Tables Name / Schema / Properties COUSTUMER_DETAILS · GFW33074 LOCATIONS · GFW33074 PRODUCT_BALANCE · GFW33074 SHOP_DETAILS · GFW33074 Total: 4, selected: 1 Table definition COUSTUMER_DETAILS Name / Data type / Nullable NAME · VARCHAR · Y EMAIL · VARCHAR · Y USERNAME · VARCHAR · Y PASSWORD · VARCHAR · Y
5.	Effective User Story	• Dashboard is used to accessing the other pages. • Add product page is to add the product they need. • Minimum quantity page is to view the present quantity and minimum quantity of the product they need. Search product page is used to search the product quickly.



8.2. USER ACCEPTANCE TESTING



Step	Procedures	Expected Result	Result
1	Insert admin, username, and password	Save the insert data into database	Success
2	Insert correct username, password for login	Verify the admin	Success
3	Click 'Register,' 'Login' button	Application redirect admin to Login page after register and Main page after login	Success
4	Repeat step 2 and 3 for login using false username, password	Application display error message	Success
5	Update Admin Account	New update data saved into database	Success
6	Log Out Account	Log out redirected to Login page	Success
Precondition		No credentials are currently login	
Post-condition		New and updated Admin name, username, and password saved in	



[Project Name] [Document Name - Version Number]

Acceptance Checklist Template

[Use this checklist to provide guidance to testers when identifying and planning software acceptance activities. This checklist captures acceptance management activities to be performed throughout the development life-cycle. Its objectives are to verify the software meets user's requirements, is fully operational, obtain the owner's acceptance, and transfer responsibility to the owner.]

Acceptance	Reference	Yes/No
User Training has been conducted.	Conduct User Training	Yes/No
System and Data Conversion has been performed as per Conversion Plan.	Installation Activities	Yes/No
Installation site were inspected in accordance with Installation Plan.	Installation Activities	Yes/No
Installation has been coordinated with System Owner, Operations, Support etc	Installation Activities	Yes/No
Modifications to the physical installation environment are completed.	Installation Activities	Yes/No
Hardware has been inventoried and tested.	Installation Activities	Yes/No
Data-loads and Data Conversion have been installed and executed.	Installation Activities	Yes/No
Software has been installed & tested on the hardware platform	Installation Activities	Yes/No
Problems and corrective action are documented.	Conduct Installation Tests	Yes/No
Equipment and software retested after repairs, replacements, or modifications.	Conduct Installation Tests	Yes/No
User Training has been conducted.	Conduct User Training	Yes/No
System and Data Conversion has been performed as per Conversion Plan.	Installation Activities	Yes/No

1 | Page

© [Name of Company]

9.RESULT

9.1. PERFORMANCE TESTING

Inventory Performance is a measure of how effectively and efficiently inventory is used and replenished. The goal of inventory performance metrics is to compare actual on-hand dollars versus forecasted cost of goods sold.

- Weeks on Hand. ...
- Inventory Turnover Rate. ...
- Days on Hand. ...
- Stock to Sales Ratio. ...
- Sell-through Rate. ...
- Backorder Rate. ...
- Accuracy of Forecast Demand. ...
- Rate of Return.

10.ADVANTAGE AND DISADVANTAGE

Advantage:

- To maintain the right amount of stocks
- To a more organized warehouse
- It saves time and money
- Improves efficiency and productivity
- A well-structured inventory management system leads to improved customer retention:
- It leads It helps Avoid lawsuits and regulatory fines
- Schedule maintenance
- Reduction in holding costs
- Flexibility

Disadvantage:

- Bureaucracy
- Impersonal touch
- Production problem
- Increased space is need to hold the inventory
- Complexity

11.CONCLUSION

In conclusion as you can see the importance of inventory management is very serious, it is one of the most important aspects of any business. The aspect of this part of the business is whether or not you can satisfy the demand of your customers if you aren't sure if you have all the materials available to make the final product Without having the proper inventory management, they would not be able to supply their customers with their ordered ambulance. And this product is what their entire business is based on, so it is of great importance When they are choosing from the different types of programs or automated systems to help with keeping records accurate, needs to keep in mind that the customer is not concerned with which materials are needed to complete the finished product, but the product is operating as promised based on the contract. In addition, the plans for the maintenance of having proper inventory levels need to be in place and also adjusted when the company grows and as the business dictates implements the new suggestions, they will be on the right track to having a well-established business.

12. FUTURE SCOPE

The scope of an inventory system can cover many needs, including valuing the inventory, measuring the change in inventory and planning for future inventory levels. The value of the inventory at the end of each period provides a basis for financial reporting on the balance sheet. Measuring the change in inventory allows the company to determine the cost of inventory sold during the period. This allows the company to plan for future inventory needs.

13.APPENDIX

SOURCE CODE:

```
# creating an app instance
app = Flask(__name__, template_folder='templates')

app.secret_key = 'a'

dsn_hostname = "764264db-9824-4b7c-82df-
40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud"
dsn_uid = "gfw33074"
dsn_pwd = "9s4qVEi8iBtYttud"
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "bludb"
dsn_port = "32536"
dsn_protocol = "TCPIP"
dsn_security = "SSL"
dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
    "SECURITY={7};").format(dsn_driver, dsn_database, dsn_hostname,
dsn_port, dsn_protocol, dsn_uid, dsn_pwd,
                            dsn_security)

conn = ibm_db.connect(dsn, "", "")

print(conn)
print("Connecting Successful...........")


# Index
@app.route('/')
def index():
    return render_template('home.html')


# Products
@app.route('/products')
def products():
    sql = "SELECT * FROM products"
    stmt = ibm_db.prepare(conn, sql)
    result = ibm_db.execute(stmt)

    products = []
    row = ibm_db.fetch_assoc(stmt)
    while (row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt)
    products = tuple(products)
    # print(products)

    if result > 0:
        return render_template('products.html', products=products)
```

```python
    else:
        msg = 'No products found'
        return render_template('products.html', msg=msg)


# Locations
@app.route('/locations')
def locations():
    sql = "SELECT * FROM locations"
    stmt = ibm_db.prepare(conn, sql)
    result = ibm_db.execute(stmt)

    locations = []
    row = ibm_db.fetch_assoc(stmt)
    while (row):
        locations.append(row)
        row = ibm_db.fetch_assoc(stmt)
    locations = tuple(locations)
    # print(locations)

    if result > 0:
        return render_template('locations.html', locations=locations)
    else:
        msg = 'No locations found'
        return render_template('locations.html', msg=msg)


# Product Movements
@app.route('/product_movements')
def product_movements():
    sql = "SELECT * FROM productmovements"
    stmt = ibm_db.prepare(conn, sql)
    result = ibm_db.execute(stmt)

    movements = []
    row = ibm_db.fetch_assoc(stmt)
    while (row):
        movements.append(row)
        row = ibm_db.fetch_assoc(stmt)
    movements = tuple(movements)
    # print(movements)

    if result > 0:
        return render_template('product_movements.html',
movements=movements)
    else:
        msg = 'No product movements found'
        return render_template('product_movements.html', msg=msg)


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

```
# user register
@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password = sha256_crypt.encrypt(str(form.password.data))

        sql1 = "INSERT INTO COUSTUMER_DETAILS (name, email, username,
password) VALUES(?,?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1, 1, name)
        ibm_db.bind_param(stmt1, 2, email)
        ibm_db.bind_param(stmt1, 3, username)
        ibm_db.bind_param(stmt1, 4, password)
        alert(email)
        ibm_db.execute(stmt1)
        # for flash messages taking parameter and the category of message
to be flashed
        flash("You are now registered and can log in", "success")

        # when registration is successful redirect to home
        return redirect(url_for('login'))
    return render_template('register.html', form=form)


# User login
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Get form fields
        username = request.form['username']
        password_candidate = request.form['password']

        sql1 = "Select * from COUSTUMER_DETAILS where username = ?"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1, 1, username)
        result = ibm_db.execute(stmt1)
        d = ibm_db.fetch_assoc(stmt1)
        if result > 0:
            # Get the stored hash
            data = d
            password = data['PASSWORD']

            # compare passwords
            if sha256_crypt.verify(password_candidate, password):
                # Passed
                session['logged_in'] = True
                session['username'] = username

                flash("you are now logged in", "success")
                return redirect(url_for('dashboard'))
            else:
                error = 'Invalid Login'
                return render_template('login.html', error=error)
            # Close connection
            cur.close()
        else:
            error = 'Username not found'
            return render_template('login.html', error=error)
    return render_template('login.html')


# check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login', 'danger')
            return redirect(url_for('login'))

    return wrap


# Logout
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return redirect(url_for('login'))


# Dashboard
@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql2 = "SELECT product_id, location_id, qty FROM product_balance"
    sql3 = "SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result = ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)

    products = []
    row = ibm_db.fetch_assoc(stmt2)
    while (row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products = tuple(products)

    locations = []
    row2 = ibm_db.fetch_assoc(stmt3)
    while (row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations = tuple(locations)

    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

    if result > 0:
        return render_template('dashboard.html', products=products,
locations=locs)
    else:
        msg = 'No products found'
        return render_template('dashboard.html', msg=msg)


# Product Form Class
class ProductForm(Form):
    product_id = StringField('Product ID', [validators.Length(min=1,
max=200)])
    product_cost = StringField('Product Cost', [validators.Length(min=1,
max=200)])
    product_num = StringField('Product Num', [validators.Length(min=1,
max=200)])


# Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
    form = ProductForm(request.form)
    if request.method == 'POST' and form.validate():
        product_id = form.product_id.data
        product_cost = form.product_cost.data
        product_num = form.product_num.data

        sql1 = "INSERT INTO products(product_id, product_cost, product_num)
VALUES(?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1, 1, product_id)
        ibm_db.bind_param(stmt1, 2, product_cost)
        ibm_db.bind_param(stmt1, 3, product_num)

        ibm_db.execute(stmt1)

        flash("Product Added", "success")

        return redirect(url_for('products'))

    return render_template('add_product.html', form=form)


# Edit Product
@app.route('/edit_product/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_product(id):
    sql1 = "Select * from products where product_id = ?"
    stmt1 = ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1, 1, id)
    result = ibm_db.execute(stmt1)
    product = ibm_db.fetch_assoc(stmt1)

    print(product)
    # Get form
    form = ProductForm(request.form)

    # populate product form fields
    form.product_id.data = product['PRODUCT_ID']
    form.product_cost.data = str(product['PRODUCT_COST'])
    form.product_num.data = str(product['PRODUCT_NUM'])

    if request.method == 'POST' and form.validate():
        product_id = request.form['product_id']
        product_cost = request.form['product_cost']
        product_num = request.form['product_num']

        sql2 = "UPDATE products SET
product_id=?,product_cost=?,product_num=? WHERE product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2, 1, product_id)
        ibm_db.bind_param(stmt2, 2, product_cost)
        ibm_db.bind_param(stmt2, 3, product_num)
        ibm_db.bind_param(stmt2, 4, id)
        ibm_db.execute(stmt2)

        flash("Product Updated", "success")

        return redirect(url_for('products'))

    return render_template('edit_product.html', form=form)


# Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in
def delete_product(id):
    sql2 = "DELETE FROM products WHERE product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2, 1, id)
    ibm_db.execute(stmt2)

    flash("Product Deleted", "success")

    return redirect(url_for('products'))


# Location Form Class
class LocationForm(Form):
    location_id = StringField('Location ID', [validators.Length(min=1,
max=200)])


# Add Location
@app.route('/add_location', methods=['GET', 'POST'])
@is_logged_in
def add_location():
    form = LocationForm(request.form)
    if request.method == 'POST' and form.validate():
        location_id = form.location_id.data

        sql2 = "INSERT into locations VALUES(?)"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2, 1, location_id)
        ibm_db.execute(stmt2)

        flash("Location Added", "success")

        return redirect(url_for('locations'))

    return render_template('add_location.html', form=form)


# Edit Location
@app.route('/edit_location/<string:id>', methods=['GET', 'POST'])
```

```
@is_logged_in
def edit_location(id):
    sql2 = "SELECT * FROM locations where location_id = ?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2, 1, id)
    result = ibm_db.execute(stmt2)
    location = ibm_db.fetch_assoc(stmt2)
    # Get form
    form = LocationForm(request.form)
    print(location)

    # populate article form fields
    form.location_id.data = location['LOCATION_ID']

    if request.method == 'POST' and form.validate():
        location_id = request.form['location_id']

        sql2 = "UPDATE locations SET location_id=? WHERE location_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2, 1, location_id)
        ibm_db.bind_param(stmt2, 2, id)
        ibm_db.execute(stmt2)

        flash("Location Updated", "success")

        return redirect(url_for('locations'))

    return render_template('edit_location.html', form=form)


# Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in
def delete_location(id):
    sql2 = "DELETE FROM locations WHERE location_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2, 1, id)
    ibm_db.execute(stmt2)

    flash("Location Deleted", "success")

    return redirect(url_for('locations'))


# Product Movement Form Class
class ProductMovementForm(Form):
    from_location = SelectField('From Location', choices=[])
    to_location = SelectField('To Location', choices=[])
    product_id = SelectField('Product ID', choices=[])
    qty = IntegerField('Quantity')


class CustomError(Exception):
    pass


# Add Product Movement
@app.route('/add_product_movements', methods=['GET', 'POST'])
@is_logged_in
def add_product_movements():
    form = ProductMovementForm(request.form)
    sql2 = "SELECT product_id FROM products"
    sql3 = "SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result = ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)

    products = []
    row = ibm_db.fetch_assoc(stmt2)
    while (row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products = tuple(products)

    locations = []
    row2 = ibm_db.fetch_assoc(stmt3)
    while (row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations = tuple(locations)

    prods = []
    for p in products:
        prods.append(list(p.values())[0])

    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

    form.from_location.choices = [(l, l) for l in locs]
    form.from_location.choices.append(("Main Inventory", "Main Inventory"))
    form.to_location.choices = [(l, l) for l in locs]
    form.to_location.choices.append(("Main Inventory", "Main Inventory"))
    form.product_id.choices = [(p, p) for p in prods]

    if request.method == 'POST' and form.validate():
        from_location = form.from_location.data
        to_location = form.to_location.data
        product_id = form.product_id.data
        qty = form.qty.data

        if from_location == to_location:
            raise CustomError("Please Give different From and To
Locations!!")


        elif from_location == "Main Inventory":
            sql2 = "SELECT * from product_balance where location_id=? and
product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2, 1, to_location)
            ibm_db.bind_param(stmt2, 2, product_id)
            result = ibm_db.execute(stmt2)
            result = ibm_db.fetch_assoc(stmt2)
            print("------------------")
            print(result)
            print("------------------")
            app.logger.info(result)
            if result != False:
                if (len(result)) > 0:
                    Quantity = result["QTY"]
                    q = Quantity + qty

                    sql2 = "UPDATE product_balance set qty=? where
location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, q)
                    ibm_db.bind_param(stmt2, 2, to_location)
                    ibm_db.bind_param(stmt2, 3, product_id)
                    ibm_db.execute(stmt2)

                    sql2 = "INSERT into productmovements(from_location,
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, from_location)
                    ibm_db.bind_param(stmt2, 2, to_location)
                    ibm_db.bind_param(stmt2, 3, product_id)
                    ibm_db.bind_param(stmt2, 4, qty)
                    ibm_db.execute(stmt2)
            else:

                sql2 = "INSERT into product_balance(product_id,
location_id, qty) values(?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2, 1, product_id)
                ibm_db.bind_param(stmt2, 2, to_location)
                ibm_db.bind_param(stmt2, 3, qty)
                ibm_db.execute(stmt2)

                sql2 = "INSERT into productmovements(from_location,
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2, 1, from_location)
                ibm_db.bind_param(stmt2, 2, to_location)
                ibm_db.bind_param(stmt2, 3, product_id)
                ibm_db.bind_param(stmt2, 4, qty)
                ibm_db.execute(stmt2)

            sql = "select product_num from products where product_id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, product_id)
            current_num = ibm_db.execute(stmt)
            current_num = ibm_db.fetch_assoc(stmt)

            sql2 = "Update products set product_num=? where product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2, 1, current_num['PRODUCT_NUM'] - qty)
            ibm_db.bind_param(stmt2, 2, product_id)
            ibm_db.execute(stmt2)

            alert_num = current_num['PRODUCT_NUM'] - qty

            if (alert_num <= 0):
                alert(
                    "Please update the quantity of the product {}, Atleast
{} number of pieces must be added to finish the pending Product
Movements!".format(
                        product_id, -alert_num))

        elif to_location == "Main Inventory":
            sql2 = "SELECT * from product_balance where location_id=? and 
product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2, 1, from_location)
            ibm_db.bind_param(stmt2, 2, product_id)
            result = ibm_db.execute(stmt2)
            result = ibm_db.fetch_assoc(stmt2)

            app.logger.info(result)
            if result != False:
                if (len(result)) > 0:
                    Quantity = result["QTY"]
                    q = Quantity - qty

                    sql2 = "UPDATE product_balance set qty=? where 
location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, q)
                    ibm_db.bind_param(stmt2, 2, to_location)
                    ibm_db.bind_param(stmt2, 3, product_id)
                    ibm_db.execute(stmt2)

                    sql2 = "INSERT into productmovements(from_location, 
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, from_location)
                    ibm_db.bind_param(stmt2, 2, to_location)
                    ibm_db.bind_param(stmt2, 3, product_id)
                    ibm_db.bind_param(stmt2, 4, qty)
                    ibm_db.execute(stmt2)

                    flash("Product Movement Added", "success")

                    sql = "select product_num from products where 
product_id=?"
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt, 1, product_id)
                    current_num = ibm_db.execute(stmt)
                    current_num = ibm_db.fetch_assoc(stmt)

                    sql2 = "Update products set product_num=? where 
product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, current_num['PRODUCT_NUM']
+ qty)
                    ibm_db.bind_param(stmt2, 2, product_id)
                    ibm_db.execute(stmt2)

                    alert_num = q
                    if (alert_num <= 0):
                        alert("Please Add {} number of {} to {} 
warehouse!".format(-q, product_id, from_location))
            else:
                raise CustomError("There is no product named {} in 
{}.".format(product_id, from_location))



        else:  # will be executed if both from_location and to_location are 
specified
            f = 0
            sql = "SELECT * from product_balance where location_id=? and
product_id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, from_location)
            ibm_db.bind_param(stmt, 2, product_id)
            result = ibm_db.execute(stmt)
            result = ibm_db.fetch_assoc(stmt)

            if result != False:
                if (len(result)) > 0:
                    Quantity = result["QTY"]
                    q = Quantity - qty

                    sql2 = "UPDATE product_balance set qty=? where
location_id=? and product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, q)
                    ibm_db.bind_param(stmt2, 2, from_location)
                    ibm_db.bind_param(stmt2, 3, product_id)
                    ibm_db.execute(stmt2)
                    f = 1

                    alert_num = q
                    if (alert_num <= 0):
                        alert("Please Add {} number of {} to {}
warehouse!".format(-q, product_id, from_location))

            else:
                raise CustomError("There is no product named {} in
{}.".format(product_id, from_location))

            if (f == 1):
                sql = "SELECT * from product_balance where location_id=?
and product_id=?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, to_location)
                ibm_db.bind_param(stmt, 2, product_id)
                result = ibm_db.execute(stmt)
                result = ibm_db.fetch_assoc(stmt)

                if result != False:
                    if (len(result)) > 0:
                        Quantity = result["QTY"]
                        q = Quantity + qty

                        sql2 = "UPDATE product_balance set qty=? where
location_id=? and product_id=?"
                        stmt2 = ibm_db.prepare(conn, sql2)
                        ibm_db.bind_param(stmt2, 1, q)
                        ibm_db.bind_param(stmt2, 2, to_location)
                        ibm_db.bind_param(stmt2, 3, product_id)
                        ibm_db.execute(stmt2)

                else:

                    sql2 = "INSERT into product_balance(product_id,
location_id, qty) values(?, ?, ?)"
                    stmt2 = ibm_db.prepare(conn, sql2)
                    ibm_db.bind_param(stmt2, 1, product_id)
                    ibm_db.bind_param(stmt2, 2, to_location)
                    ibm_db.bind_param(stmt2, 3, qty)
                    ibm_db.execute(stmt2)
                sql2 = "INSERT into productmovements(from_location,
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2, 1, from_location)
                ibm_db.bind_param(stmt2, 2, to_location)
                ibm_db.bind_param(stmt2, 3, product_id)
                ibm_db.bind_param(stmt2, 4, qty)
                ibm_db.execute(stmt2)

                flash("Product Movement Added", "success")

        render_template('products.html', form=form)

        return redirect(url_for('product_movements'))

    return render_template('add_product_movements.html', form=form)


# Delete Product Movements
@app.route('/delete_product_movements/<string:id>', methods=['POST'])
@is_logged_in
def delete_product_movements(id):
    sql2 = "DELETE FROM productmovements WHERE movement_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2, 1, id)
    ibm_db.execute(stmt2)

    flash("Product Movement Deleted", "success")

    return redirect(url_for('product_movements'))


if __name__ == '__main__':
    app.secret_key = "secret123"
    # when the debug mode is on, we do not need to restart the server again
and again
    app.run(debug=True)
```

GitHub & Project Demo Link:

https://github.com/IBM-EPBL/IBM-Project-9894-1659082785