**PROJECT REPORT**

**INVENTORY MANAGEMENT** **SYSTEM FOR RETAILERS**

**submitted by**

# PNT2022TMID24328

AJAY KUMAR S - 113319104004

MADHAN T - 113319104046

KIRUTHIK RAJAN K – 113319104042

SURENDHAR PK – 113319104088

AKASH S-113319104006

# TABLE OF CONTENTS

1. INTRODUCTION 1
   1. PROJECT OVERVIEW 1
   2. PURPOSE 1
2. LITERATURE SURVEY 2
   1. EXISTING PROBLEM 2
   2. REFERENCES 2
   3. PROBLEM STATEMENTDEFINITION 5
3. IDEATION AND PROPOSED SOLUTION 6
   1. EMPATHY MAP CANVAS 6
   2. IDEATION & BRAINSTORMING 7
   3. PROPOSED SOLUTION 8
   4. PROBLEM SOLUTIONFIT 9
4. REQUIREMENT ANALYSIS 10
   1. FUNCTIONAL REQUIREMENTS 10
   2. NON FUNCTIONAL REQUIREMENTS 11
5. PROJECT DESIGN 12
   1. DATA FLOW DIAGRAM 12
   2. SOLUTION &TECHNICAL ARCHITECTURE 13
   3. USER STORIES 15
6. PROJECT PLANNINGAND SCHEDULING 16
   1. SPRINT PLANNING AND ESTIMATION 16
   2. SPRINT DELIVERYSCHEDULE 16
7. CODING &SOLUTIONING 18
8. TESTING 20
   1. TEST CASES 20
   2. USER ACCEPTANCE TESTING 22

8.2.1. DEFECT ANALYSIS 22

## 8.2.2. TEST CASE ANALYSIS 22

1. RESULTS 23

9.1. PERFORMANCE METRICS 23

1. ADVANTAGES & DISADVANTAGES 29

ADVANTAGES 29

## DISADVANTAGES 29

1. CONCLUSION 30
2. FUTURE SCOPE 31

## APPENDIX 32

SOURCE CODE 32

GITHUB 41

PROJECT DEMO 41

# CHAPTER 1 INTRODUCTION

## 1.1. PROJECT OVERVIEW

More specifically, a cloud application is software that runs its processing logic and data storage between 2 different systems: client-side and server-side. Some processing takes place on an end user's local hardware, such as a desktop or mobile device, and some takes place on a remote server.

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.

## 1.2. PURPOSE

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.

## CHAPTER 2

### LITERATURE SURVEY

#### 2.1. EXISTING PROBLEM

The problem statement intends to create a desktop application for retailers that will track all aspects of IMS such as purchase details, sales details, and stock management. The application provides the retailer with detailed information about the products in their inventory, and the ability to manage them more effectively.

The inventory management system operates by connecting the Point - of – sale system's database and the application's database. When a specific product is purchased, the product's availability in the database decreases accordingly. When the quality of a product falls below a certain threshold value, the application automatically contacts the supplier for a new order. The user must specify the threshold value. The application also informs the user of the products' expiration date. The user also receives an indication of the products ordered, which they can track, and the application can make automatic payments for each supply purchase.

#### 2.2. REFERENCES

#### 1) PAPER : Inventory management for retail companies: A literature review and current trends

**AUTHOR :** Cinthya Vanessa Munoz Macas,

Jorge Andres Espinoza Aguirre,

Rodrigo Arcentales-Carrion,

Mario Pena

#### YEAR : 2021

**DESCRIPTION :**

To analyze and present an extensive literature concerning inventory management, containing multiple definitions and fundamental concepts for the retail sector. The primary outcomes of this study are the leading inventory management systems and models, the Key Performance Indicators (KPIs) for their correct management, and the benefits and challenges for choosing or adopting an efficient inventory control and management system.

**FUTURE WORK AND ANALYSIS :**

To reduce the cost and maintenance and make available for all the companies.The need to analyze their KPIs becomes highly significant, as well as their different systems, methodologies, and tools used within inventory management and optimization.

**2)PAPER :** Retail inventory management with stock-out based dynamic demand substitution

**AUTHOR :** Baris Tan

Selcuk Karabati

#### YEAR : 2013

**DESCRIPTION :** To study an inventory management problem in a retail setting with stock-out based substitutions and multiple items in a product category and propose an approximate solution to determine the order-up-to levels to maximize the expected profit subject to service level constraints. The method uses demand parameters including the substitution probabilities estimated from the point-of-sales data. The method provides a practical tool for retailers to manage their inventory.

**FUTURE WORK AND ANALYSIS :**Through a computational study, by explicitly accounting for substitutions, the performance of the inventory system can be improved. The amount of improvement depends on the minimum direct service level requirement as well as the correlation between the market share and the profit margin of the products. By combining the method we presented in an earlier study to estimate the demand and customer choice parameters, the method we presented in this study can be used to manage inventory in a better way in retailing.

**3)PAPER :** Robust inventory management with stock-out substitution

**AUTHOR :** Zhaolin Li

Grace Fu

#### YEAR : 2017

**DESCRIPTION :** Stock-out substitution is a well-documented phenomenon that occurs when customers seek a different product as a substitute for their first-choice item if it runs out of stock. We consider a single-period inventory model with limited information regarding the external demands (i.e., mean, variance, and covariance) and focus on identifying the inventory levels that maximize the worst-case expected profit. We formulate a two-stage optimization model: the second stage characterizes the worst-case joint demand distribution by treating the inventory levels as input parameters, and the first stage identifies the optimal inventory levels based on the results of the second stage.

**FUTURE WORK AND ANALYSIS :** Our approach makes use of the limited information on product demands and is suitable for the circumstance in which exact demand distributions can not be accurately estimated. After formulating the optimization model as a two-stage model, we find that the closed-form solution of the second stage is intractable except for two special cases. We develop a heuristic solution based on these two special cases. An extensive numerical study indicates that the performance of the heuristic solution is nearly optimal over a wide range of parameters. Investigating the effects of robust decision rule changes could offer interesting future research

**4)PAPER :** Managing demand uncertainty:Probabilistic selling versus inventory substitution

#### AUTHOR : Yi Zhang

Guowei Hua

Shoyang Wang

Juliang Zhang

Vicenc Fernandez

#### YEAR : 2018

**DESCRIPTION :** To combat demand uncertainty, both strategies of inventory substitution and probabilistic selling can be used. Although the two strategies differ in operation, we believe that they share a common feature in combating demand uncertainty by encouraging some customers to give up some specific demand for the product to enable demand substitution. It is interesting to explore which strategy is more advantageous to the retailer.

**FUTURE WORK AND ANALYSIS :** inventory substitution is the better choice for the retailer when the product similarity is higher.The price of the probabilistic product is an exogenous variable. Future research may extend our work by combining the pricing and inventory decisions. It is also worth considering PS in a supply chain setting. For example, it is interesting to explore the conditions under which a retailer's probabilistic selling will benefit the supplier, the retailer, and both.

**5)PAPER :** A joint model for cash and inventory management for a retailer under delay in payments

**AUTHOR :** Lama Moussawi-Haidar

Mohamad Y. Jaber

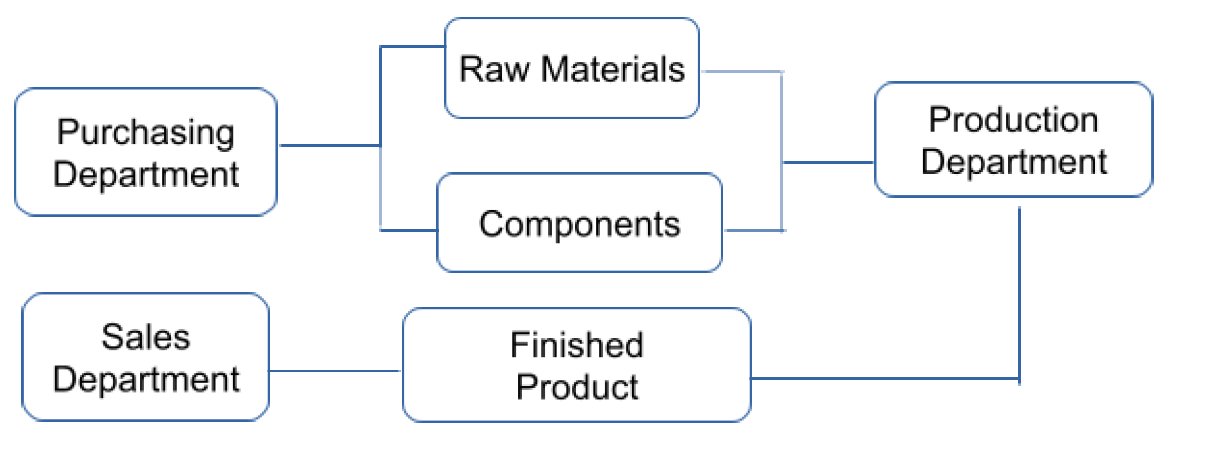
#### YEAR : 2013

**DESCRIPTION :** The problem of finding the optimal operational (how much to order and when to pay the supplier) and financial decisions (maximum cash level and loan amount) by integrating the cash management and inventory lot sizing problems.As retail companies continue to navigate through the economy downturn, it becomes critical to find innovative cost reduction methods. Cash management is a cost-intensive process for retailers, who are currently focusing on effective cash management, such as deciding on the maximum cash level to keep in their business accounts and how much to borrow to finance inventories and pay suppliers.

**FUTURE WORK AND ANALYSIS :**Results indicate that as the percentage margin increases, the order quantity and maximum cash level increase for a given credit period, and that they both increase with the credit period. Increasing the holding and storage cost, the order quantity and cash level decrease given the retailer’s return on cash. Further research should be done in the credit period to show that the cash management model reduces the retailer’s cost.

#### 2.3. PROBLEM STATEMENT DEFINITION

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. The problem statement intends to create a desktop application for retailers that will track all aspects of IMS such as purchase details, sales details, and stock management. The application provides the retailer with detailed information about the products in their inventory, and the ability to manage them more effectively.

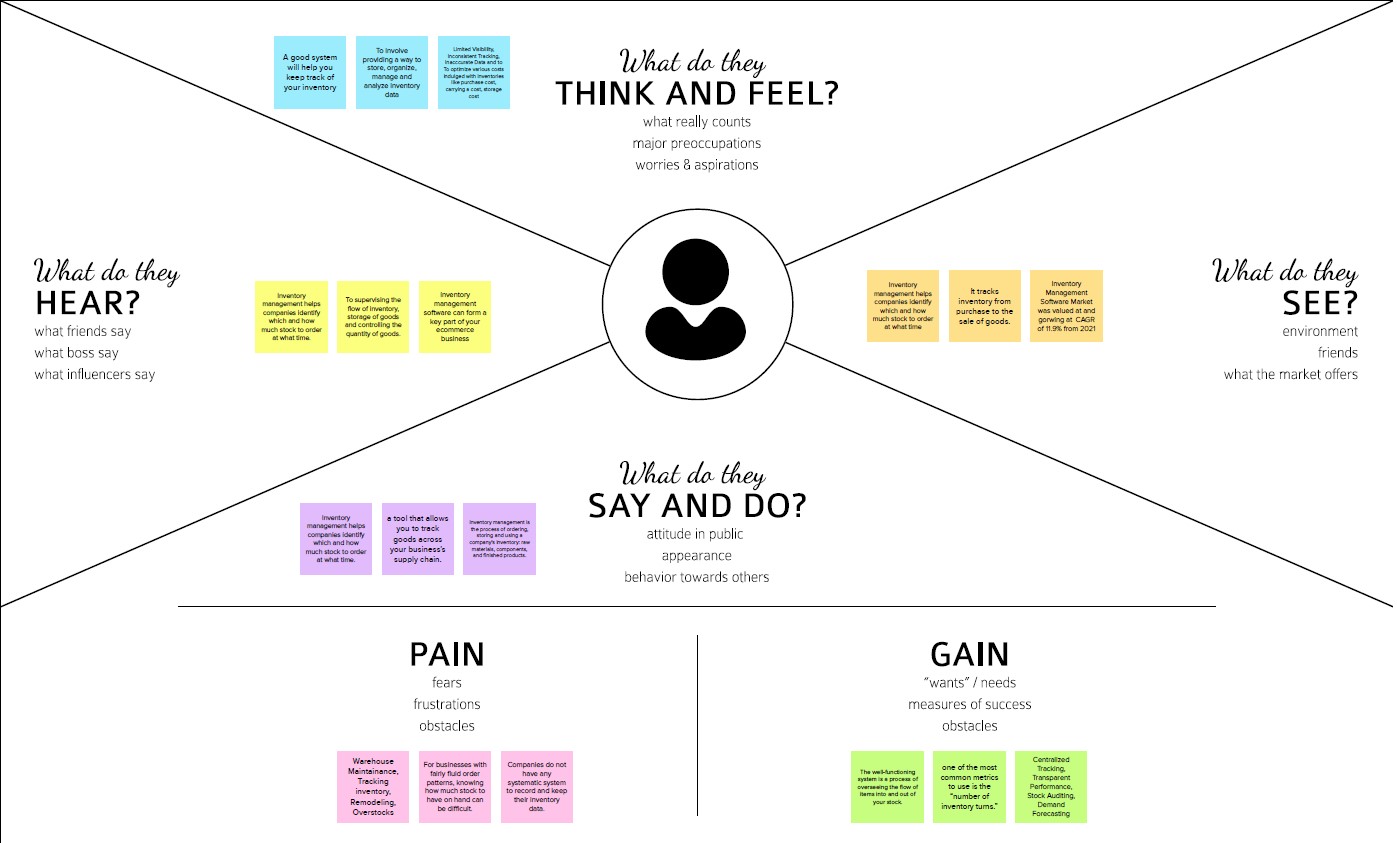


The inventory management system operates by connecting the Point - of – sale system's database and the application's database. When a specific product ispurchased, the product's availability in the database decreases accordingly. When the quality of a product falls below a certain threshold value, the application automatically contacts the supplier for a new order. The user must specify the threshold value. The application also informs the user of the products' expiration date. The user also receives an indication of the products ordered, which they can track, and the application can make automatic payments for each supply purchase.

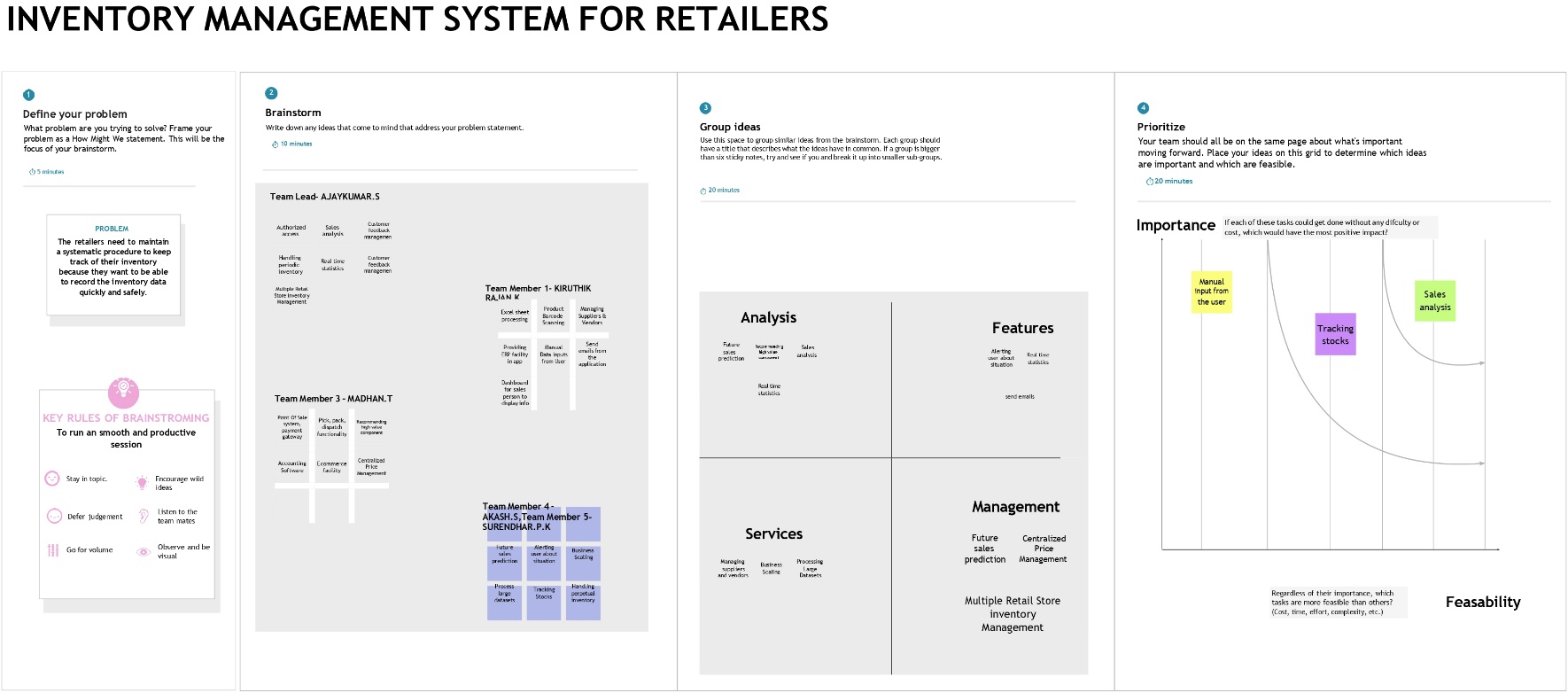
## CHAPTER 3

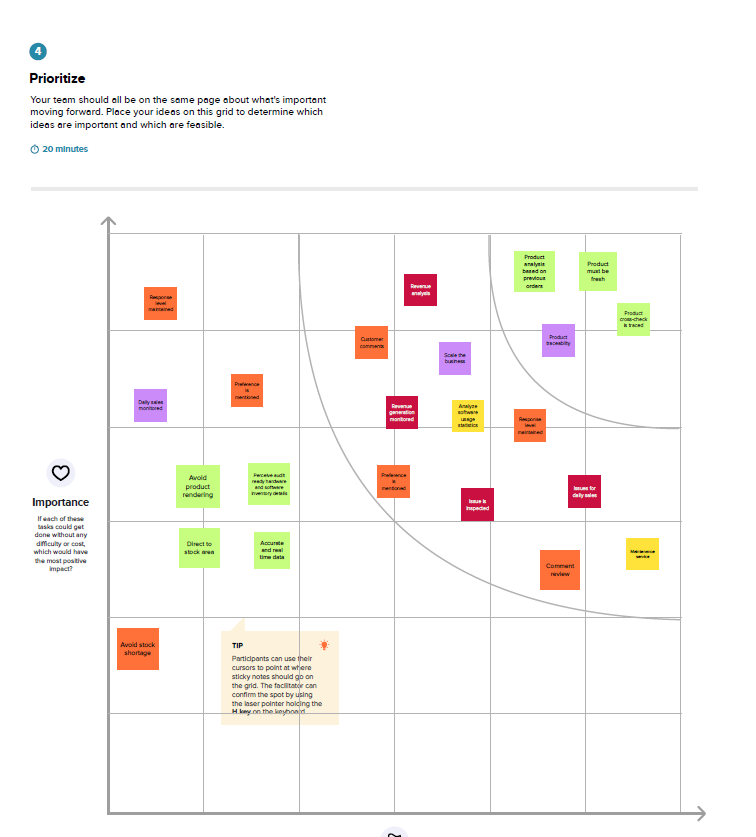
### IDEATION AND PROPOSEDSOLUTION

#### 3.1. EMPATHY MAP CANVAS



#### 3.2. IDEATION & BRAINSTORMING





##### 3.3 PROPOSED SOLUTION

|  |  |  |
| --- | --- | --- |
| **S.**  **NO** | **PARAMETER** | **DESCRIPTION** |
| 1 | Problem Statement | To create an application that manages the inventory of an retailer |
| 2 | Idea / Solution Description | The application takes the stock count and maintain the stock availability and intimate requirements |
| 3 | Novelty / Uniqueness | Instead of checking the stock availability manually this  application sends an email about the stock |
| 4 | Social Impact / Customer Satisfaction | This application reduces the manual tasks that need to be performed. This improves productivity in the workplace. |
| 5 | Business Model | This application can be used in every shops, effective for retailers    This application can also be integrated with billing system  in shops |
| 6 | Scalability of the Solution | The application can easily be scaled to accept multiple inputs  and process them parallelly to further increaseefficiency |

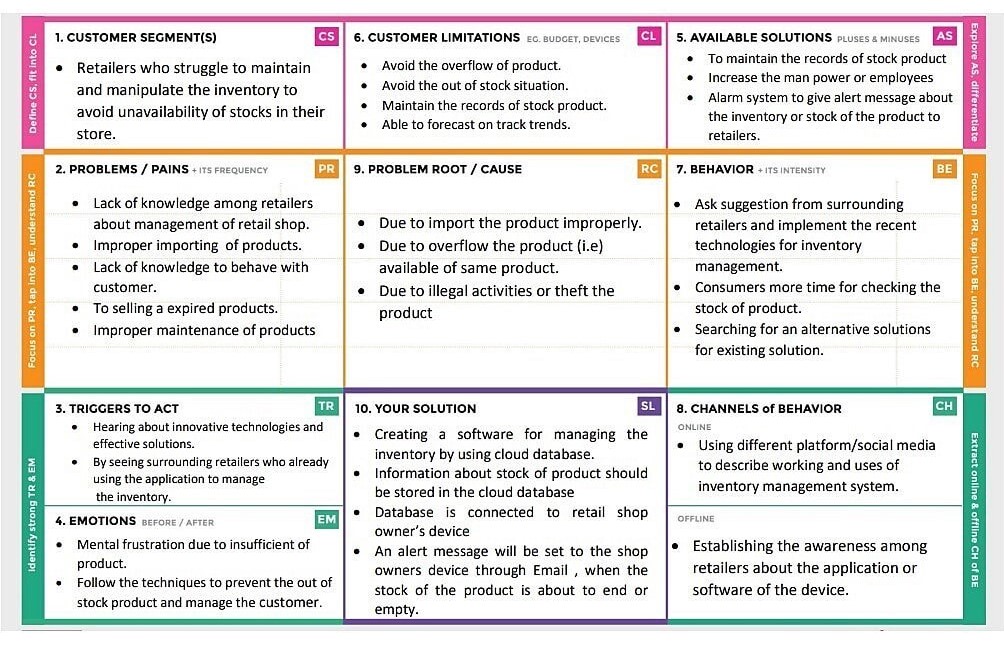
##### 3.4 PROBLEM SOLUTION FIT

Problem-solution fit is a term used to describe the point validating that the base problem resulting in a business idea really exists and the proposed solution actually solves that problem.

**Purpose**:

1. **Validate that the problem exists:** When you validate your problem hypothesis using real-world data and feedback. That is, you gather information from real users to determine whether or not they care about the pain point you’re trying to solve.
2. **Validate that your solution solves the problem:** When you validate that the target audience

appreciates the value your solution delivers to them.



## CHAPTER 4

### REQUIREMENT ANALYSIS

#### 4.1. FUNCTIONAL REQUIREMENTS

Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish. The 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 |
| FR-2 | User Confirmation | Confirmation via Email  Confirmation via OTP |
| FR-3 | User Login | Login with username  Login with password |
| FR-4 | Product record | Product name  Stock count  Product category  Vendor details |
| FR-5 | Email Notification | Email through SendGrid  Reduced stock quantity  Email to both retailer and seller |
| FR-6 | Audit Monitoring | Monitor incoming and outgoing stock |

##### 4.2 NON FUNCTIONAL REQUIREMENTS

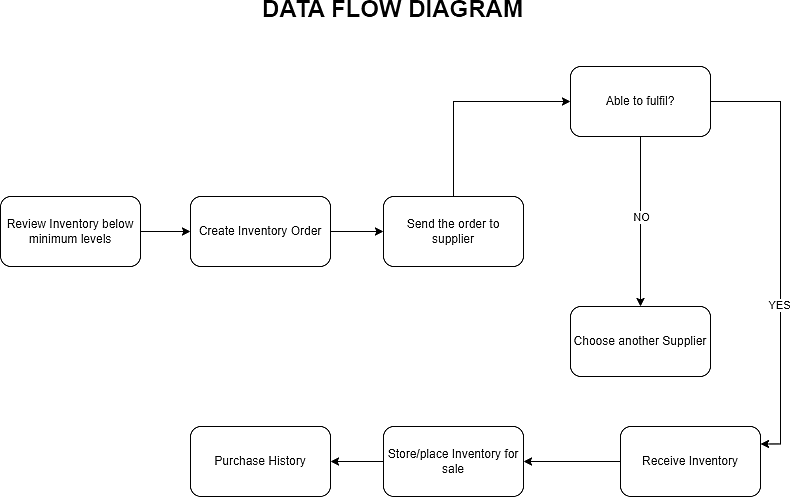
Non-functional requirement (NFR) is a [requirement](https://en.wikipedia.org/wiki/Requirement) that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. The following are the non-functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **NFR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | Highly portable, User-friendly and highly responsive UI for easy access |
| NFR-2 | **Security** | Access Control, User privileges,  Password management features |
| NFR-3 | **Reliability** | Secure server for reliable and fault tolerant connection |
| NFR-4 | **Performance** | Reliable performance with high-end servers |
| NFR-  5 | **Availability** | Service hosting server downtime should be negligible during upgradation |

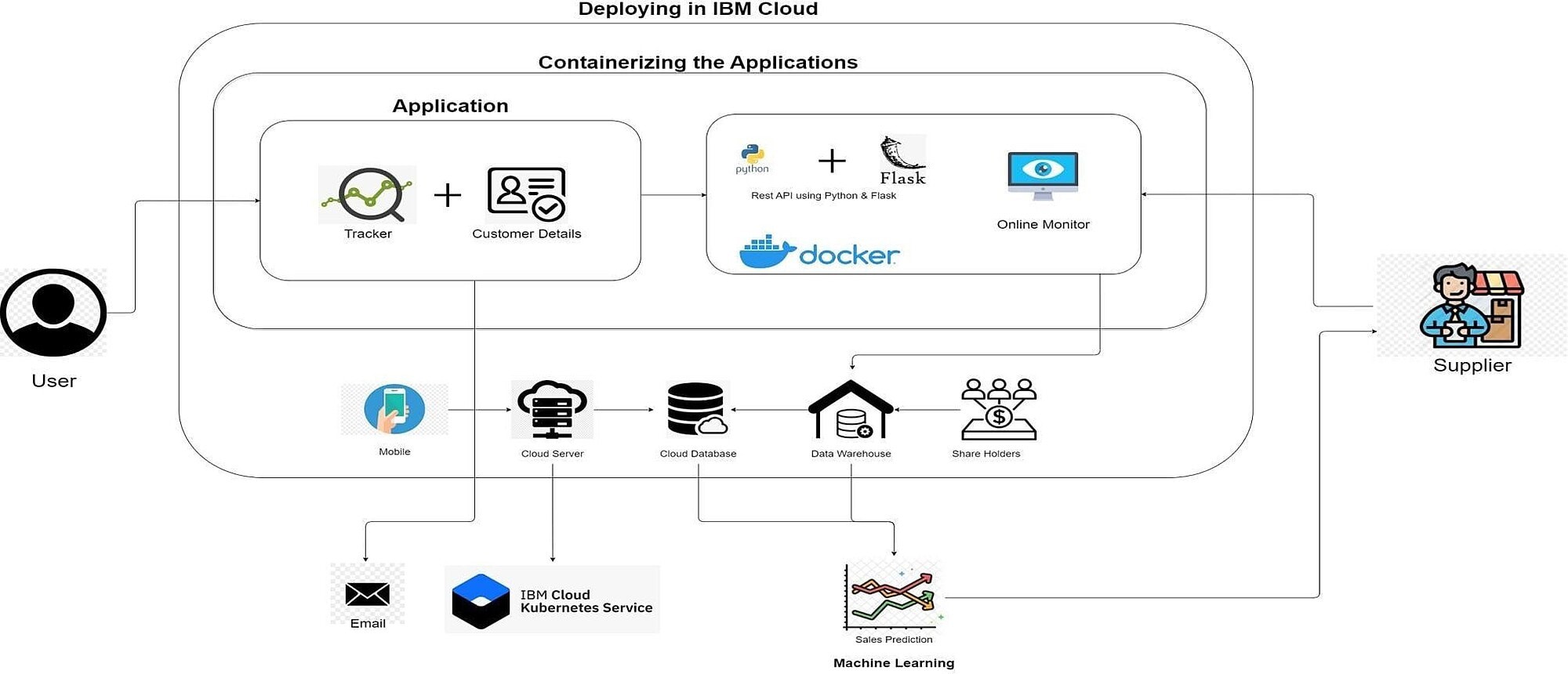
## CHAPTER 5

### PROJECT DESIGN

#### 5.1. DATA FLOW DIAGRAM



#### 5.2. SOLUTION & TECHNICAL ARCHITECTURE



**Table-1 : Components & Technologies**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.**  **No** | **Component** | **Description** | **Technology** |
| 1. | User Interface | How user interacts with application e.g. Web UI, Mobile App, Chatbot,  etc. | HTML, CSS, JavaScript, IBM  Cloud Object Storage, PythonFlask, Kubernetes, Docker, IBM  DB2, IBM Container Registry. |
| 2. | Application Logic | The logic for a process in the application | Python-Flask. |
| 3. | Database | Data Type Configuration etc. | MySQL, etc. |
| 4. | ChatBox | Chatbox for users to access help from a virtual assistant on the application. | IBM Watson Assistant |
| 5. | Cloud Database | Database Service on Cloud | IBM DB2 |
| 6. | File Storage | File storage requirements | IBM Cloud Object Storage |
| 7. | App Container | Contain the whole application in a single container. | Docker Container, IBM Container  Registry |
| 8. | Infrastructure  (Server / Cloud) | Application Deployment on Local  System / Cloud  Local Server Configuration: port 5000  Cloud Server Configuration : | Local, Cloud Foundry, Kubernetes. |
| 9. | Send Mail | To send emails when low stock is present in the inventory to retailers. | IBM SendGrid |

.

#### Table-2: Application and characteristics

|  |  |  |  |
| --- | --- | --- | --- |
| **S.**  **No** | **Characteristics** | **Description** | **Technology** |
| 1. | Open-Source  Frameworks | We use HTML, CSS, Bootstrap and Flask as the open source for our application | HTML, CSS, JavaScript,  Bootstrap, Python-Flask. |
| 2. | Security  Implementations | User log in and authentication are done to provide secure access to their account. | IBM Cloud Security, Cookies.. |
| 3. | Scalable  Architecture | The system can be scalable easily by using these technologies as to optimize, improve and add new features, allocate sufficient bandwidth to allow more users at a time, etc. | Docker, Kubernetes Cluster |
| 4. | Availability | System availability is high as we make sure the unwanted database access is minimized through SQL and code optimization. | IBM Db2, IBM Container  Registry |
| 5. | Performance | Deployment is easy and fast by containerizing the application. Providing fast access time and responsiveness by deploying the application in cloud. | Flask, Docker, IBM Db2. |

#### 5.3. USER STORIES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Functional**  **Requireme nt (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-3 |
|  |  | USN-4 | As a user, I can register for the application through  Gmail | I can register for the application through Gmail | Medium | Sprint-2 |
|  | Login | USN-5 | As a user, I can log into the application by entering email & password | I can log in by entering  Gmail & password | High | Sprint-1 |
|  | Dashboard | USN-6 | As a user, I can track data of sales of products and  inventory levels | I can track data of sales of products and inventory  levels. | High | Sprint-1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Customer (Web user) | Registration | USN-7 | 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-8 | 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-9 | As a user, I can register for the application through  Facebook | I can register & access the dashboard  with  Facebook  Login | Low | Sprint-3 |
|  |  | USN-  10 | As a user, I can register for the application through Gmail | I can register for the application through Gmail | Medium | Sprint-2 |
|  | Dashboard | USN-  12 | As a user, I can track data of sales of products and inventory  levels | I can track data of sales of products and inventory levels. | High | Sprint-1 |
| Administr ator | Manage  the Stocks | USN-  14 | As a administrator, I manage the stocks by adding, shipping and storing the stocks in the storage units | I  manage the stocks by adding, shipping and storing the stocks in the storage units. | High | Sprint-1 |
|  | Control all the users | USN-  15 | As a administrator, I can control all the users by performing basic  CRUD  operations. | I can control all the users by performing basic  CRUD  operations | High | Sprint-1 |
|  | Access the database | USN-  16 | As a administrator, I can control and access the database | I can control and access the database. | High | Sprint-1 |

## CHAPTER 6

**PROJECT PLANNING AND SCHEDULING**

### 6.1. SPRINT PLANNING AND ESTIMATION

|  |  |  |
| --- | --- | --- |
| **TITLE** | **DESCRIPTION** | **DATE** |
| **Literature Survey &**  **Information Gathering** | Literature survey on selected project and  gathering information by referring the project’s  related technical papers, research publications, etc. | 18 SEPTEMBER 2022 |
| **Prepare Empathy Map** | Prepare empathy map canvas to capture the user’s pains & gains and prepare the list of problem statements. | 22 SEPTEMBER 2022 |
| **Ideation** | To list by the organizing brainstorm sessions and  prioritize the top three  ideas based on the feasibility and importance. | 23 SEPTEMBER 2022 |

|  |  |  |
| --- | --- | --- |
| **Proposed Solution** | To prepare the proposed solution documents, which  includes the novelty, feasibility of ideas, business model,  social impact, scalability of the  solution, etc. | 23 SEPTEMBER 2022 |
| **Problem Solution Fit** | Preparing the problem solution fit document. | 27 SEPTEMBER 2022 |
| **Solution Architecture** | To prepare the solution architecture document | 27 SEPTEMBER 2022 |
| **Customer Journey** | Prepare the customers journey map help the customers understand the  user  interaction and experiences  with the application from the beginning to the end. | 19 OCTOBER 2022 |
| **Functional Requirement** | Prepare the functional requirement document. | 28 OCTOBER 2022 |
| **Data Flow Diagrams** | Draw the data flow diagrams and submit for the review. | 29 OCTOBER 2022 |
| **Technology Architecture** | Prepare technical architecture diagram. | 28 OCTOBER 2022 |
|  |  | 28 OCTOBER 2022 |
| **Prepare Milestone & Activity List** | Prepare the milestones and activity of the project. |  |
| **Project Development –**    **Delivery of Sprint-1, 2, 3 &**  **4** | Develop and submit the developed code by testing it and having no errors. | 19 NOVEMBER 2022 |

**6.2. SPRINT DELIVERY SCHEDULE**

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

Use the below template to create product backlog and sprint schedule

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional**  **Requirement**  **(Epic)** | **User Story**  **Number** | **User Story / Task** | **Story**  **Points** | **Priority** | **Team**  **Members** |
| Sprint-1 | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my Password or by entering phone number and confirming by otp | 2 | High | Ajay Kumar S  Akash S  Madhan T |
| Sprint-1 |  | USN-2 | As a user, I can register for the  application through E-mail or phone number | 1 | Medium | Madhan T  Kiruthik Rajan  Surendhar |
| Sprint-1 | Confirmation | USN-3 | As a user, I will receive confirmation email or otp once I have registered for the application | 1 | Medium | Madhan T  Akash S  Surendhar S |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional** **Requireme**  **nt (Epic)** | **User Story**  **Number** | **User Story / Task** | **Story**  **Points** | **Priori ty** | **Team**  **Members** |
| Sprint-1 | Login | USN-4 | As a user, I can log into the application by entering email & password or using phone number and otp. | 2 | High | Akash S  Ajay Kumar S |
| Sprint-2 | Dashboard | USN-5 | As a user, I can view the products which are available | 4 | High | Akash S  Madhan T  Surendhar PK  Akash S |
| Sprint-3 | Stock  Update | USN-7 | Once the product reaches the threshold level as a user  ,I will be getting the notification to reorder the stock. | 5 | High | Ajay Kumar  Kiruthik Rajan  Akash |
| Sprint-4 | Expiry update | USN-8 | As a user, I will be notified about the expiry date of the  products | 5 | High | Madhan  Surendhar  Ajay Kumar |

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

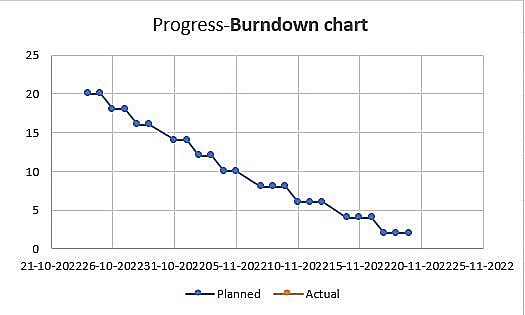
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Total Story**  **Points** | **Durati on** | **Sprint Start**  **Date** | **Sprint End Date**  **(Planned)** | **Story Points**  **Completed**  **(as on**  **Planned End**  **Date)** | **Sprint Release**  **Date (Actual)** |
| Sprint-1 | 6 | 6 Days | 24 Oct 2022 | 29 Oct 2022 | 20 | 29 Oct 2022 |
| Sprint-2 | 4 | 6 Days | 31 Oct 2022 | 05 Nov 2022 |  | 05 Nov 2022 |
| Sprint-3 | 5 | 6 Days | 07 Nov 2022 | 12 Nov 2022 |  | 12 Nov 2022 |
| Sprint-4 | 5 | 6 Days | 14 Nov 2022 | 19 Nov 2022 |  | 19 Nov 2022 |

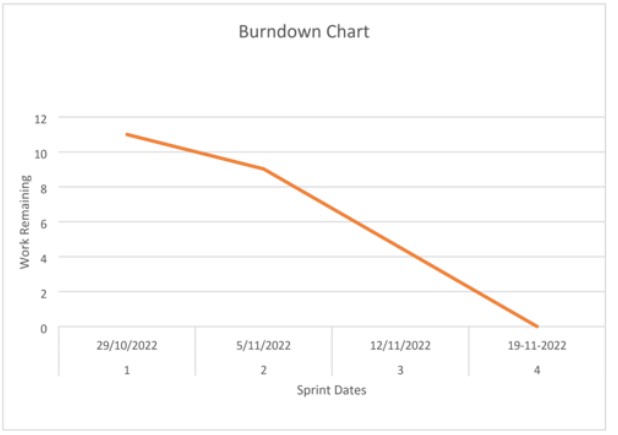
**Velocity:**

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



**AV=20/6 =3.33**





## CHAPTER 7

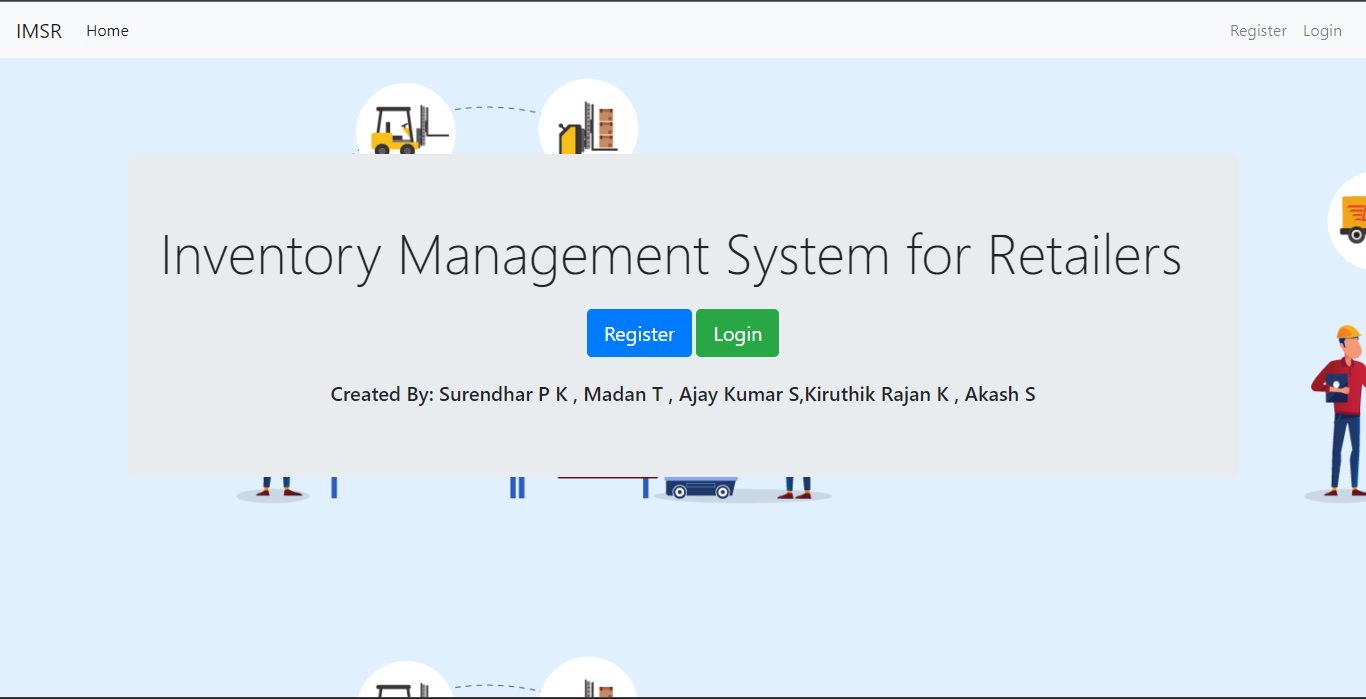
### CODING & SOLUTIONING

**7.1 Feature 1**

The main features of our project are to track product and details and users

information. The project details includes product name, product id, quantity and price.

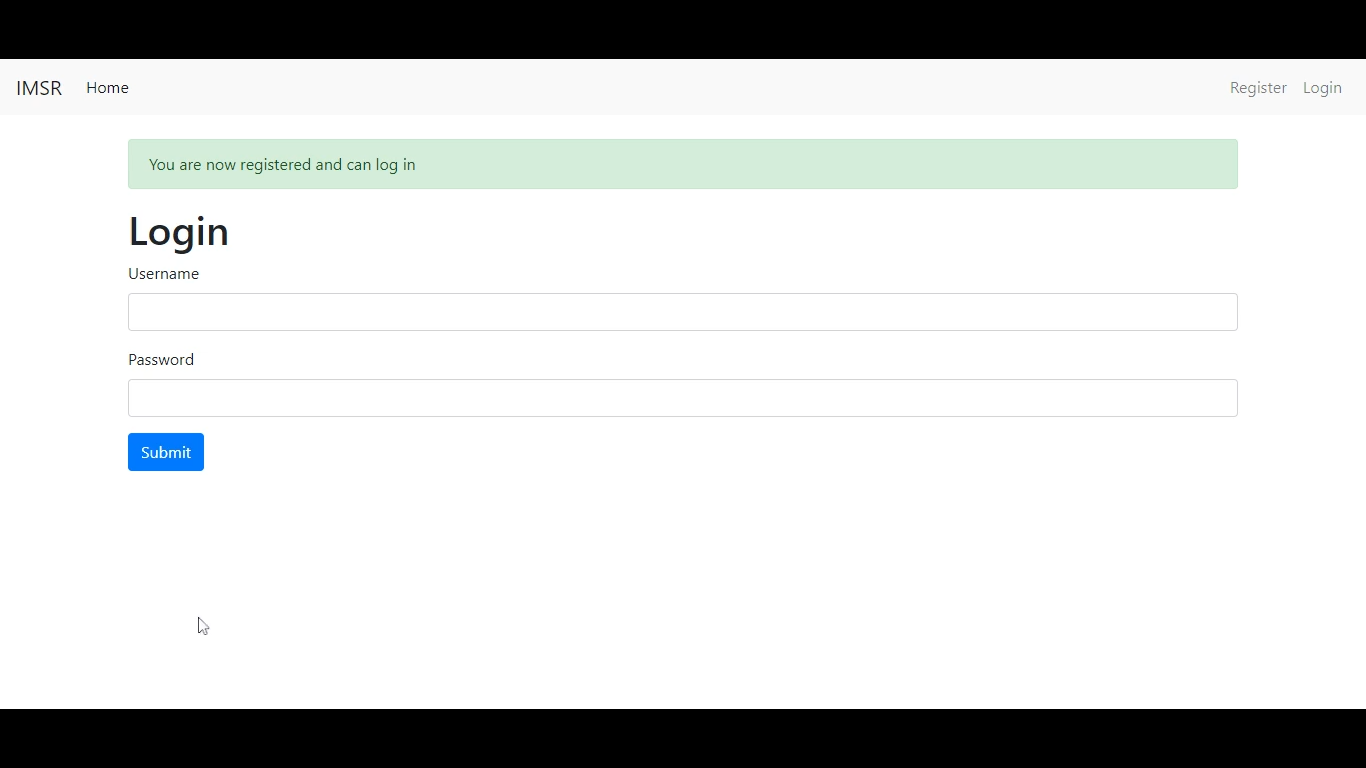
**Home Page**



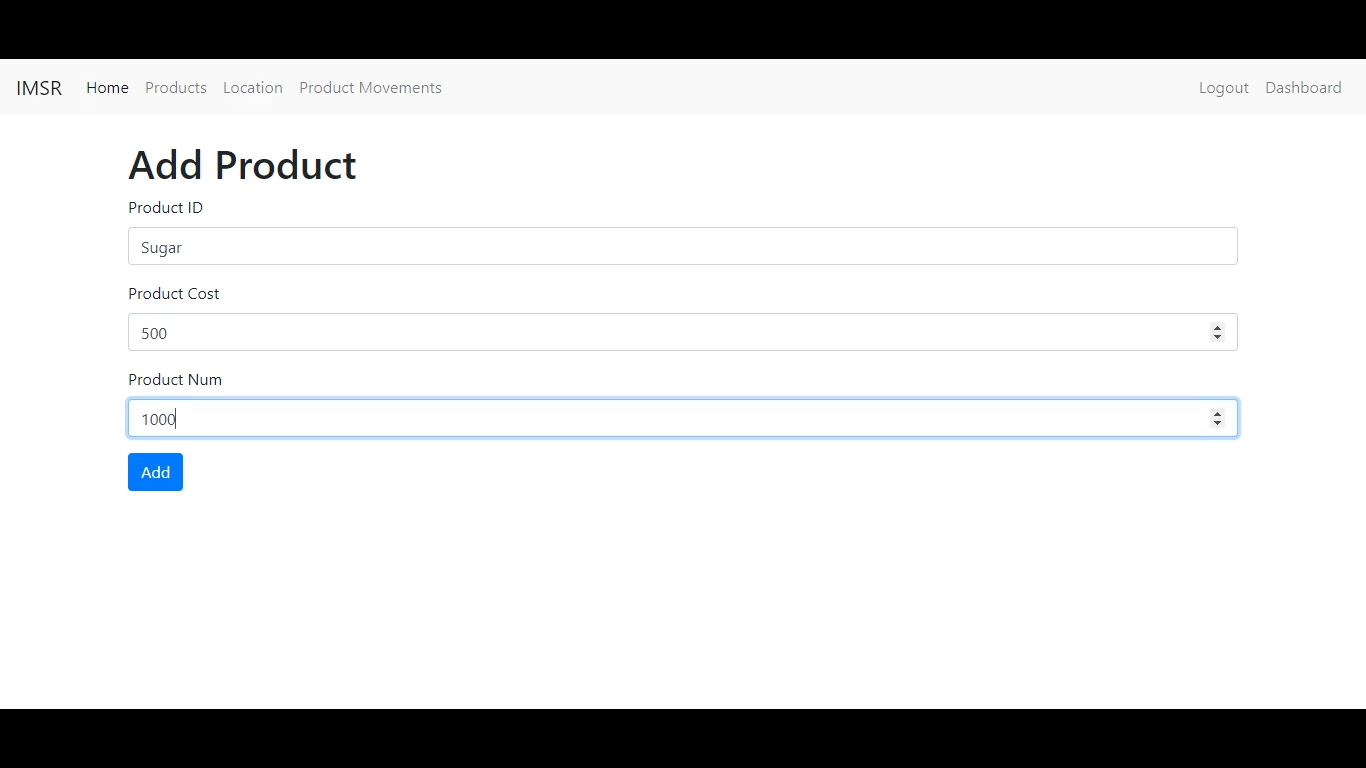
**Register page**



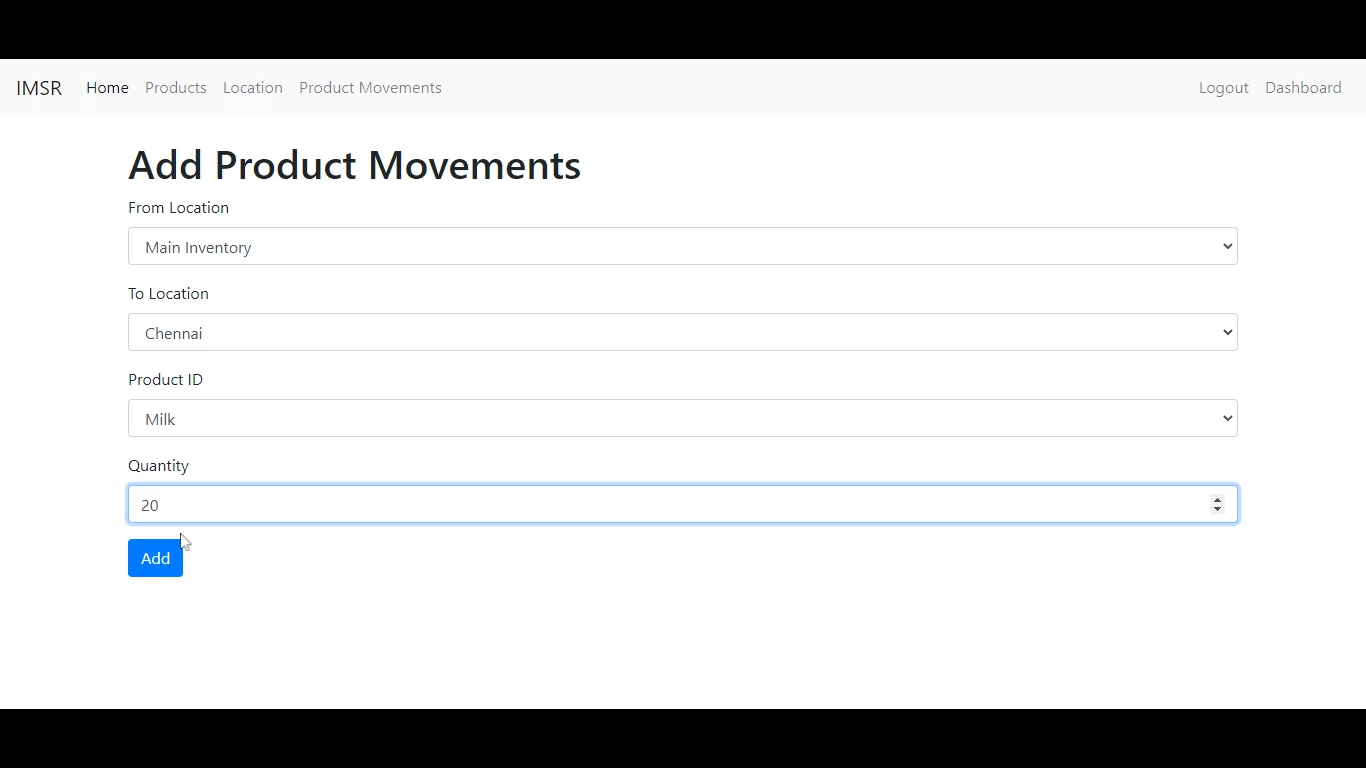
**Login page**



**Product adding page**



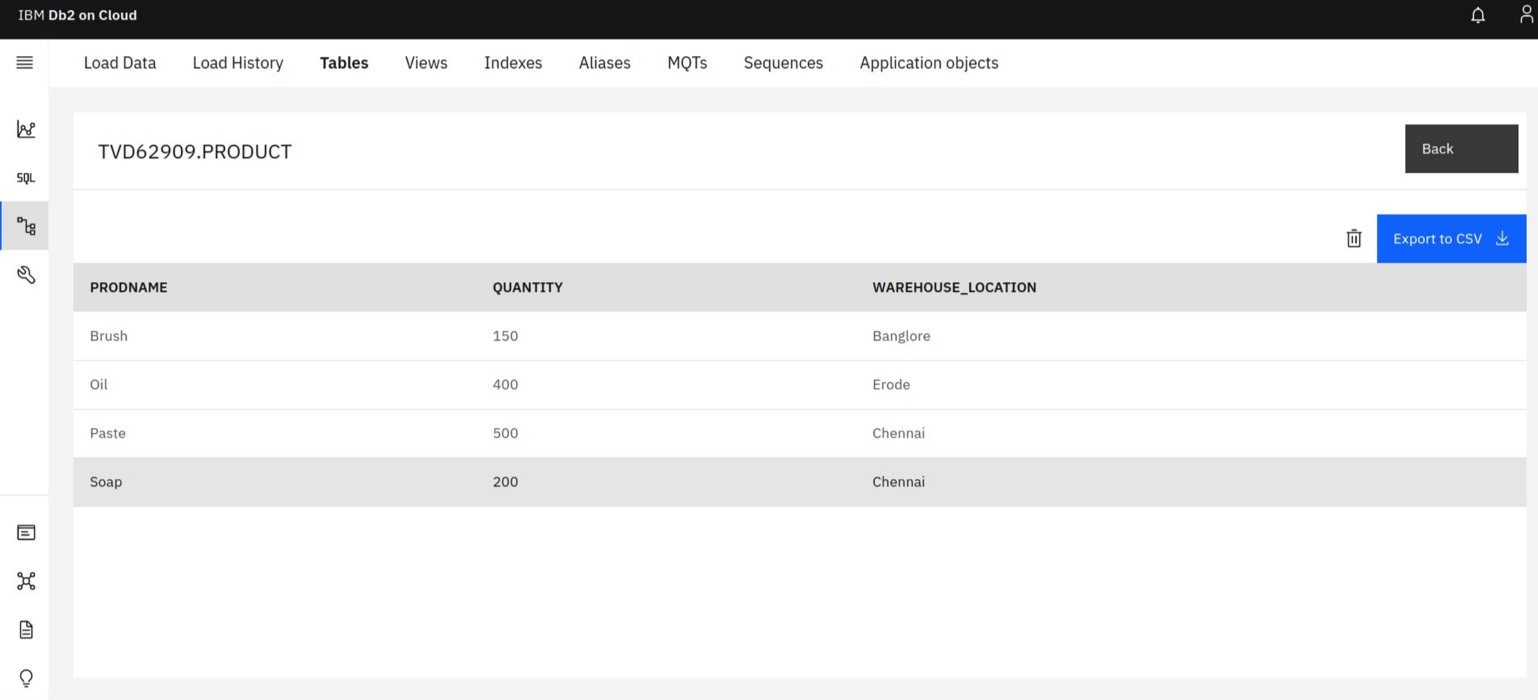
**Update page**



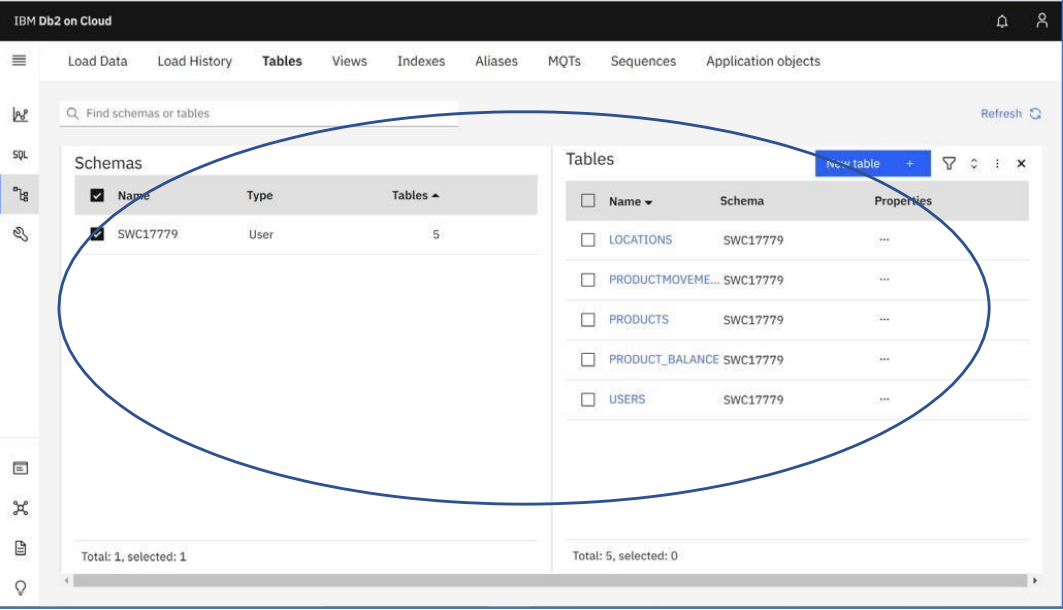
#### 7.2 Feature 2

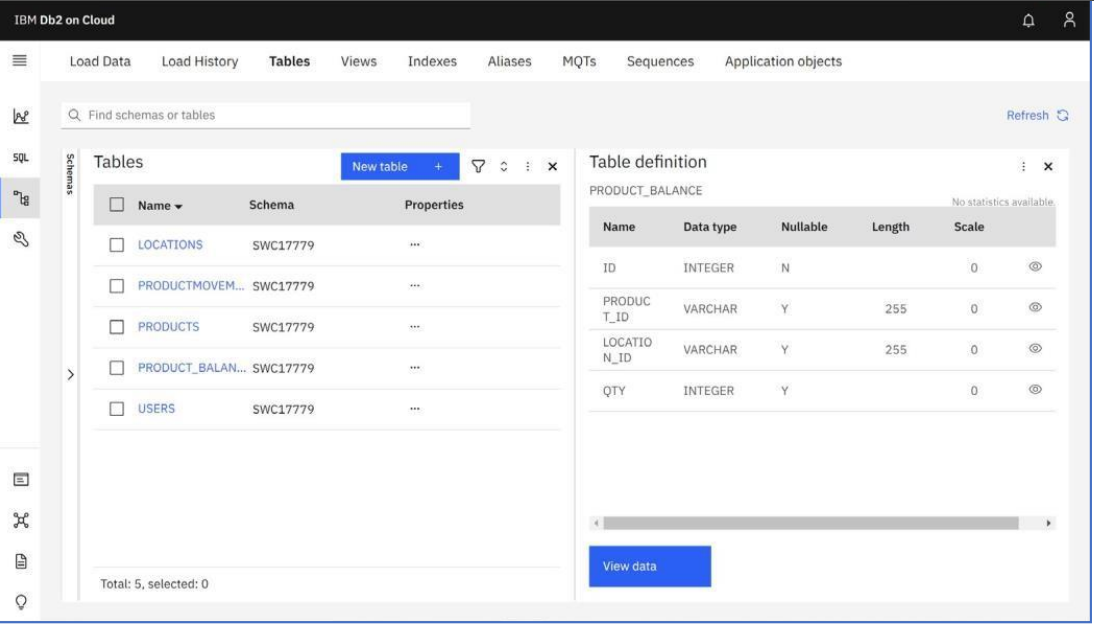
For the user information consist of user name, user id, gmail, password**.** If the stoct gets below desired limit, the warning to increase stock will be sent to user through email service.

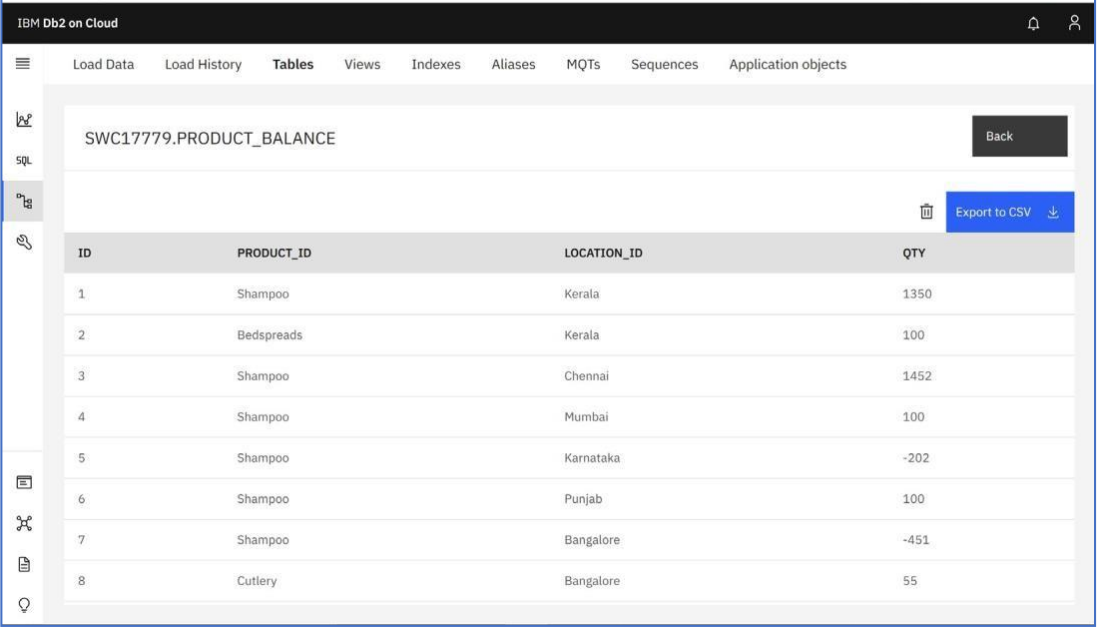
**User and product cloud details**



#### 7.3 Database Schema







## CHAPTER 8

### TESTING

#### 8. TESTING

##### 8.1 Test Cases Login

1. Verify user is able to see login page
2. Verify user is able to loginto application or not?
3. Verify login page elements

#### Register

1. Verify if user is able to enter all the details and register
2. Verify if user is redirected to login page once registered.

#### Add products

1. Verify user is able to add products in ad product page
2. Verify whether added products are correctly added in the table

#### View products

1. Verify whether products can be viewed in view page
2. Verify whether products can be retrieved correctly from database

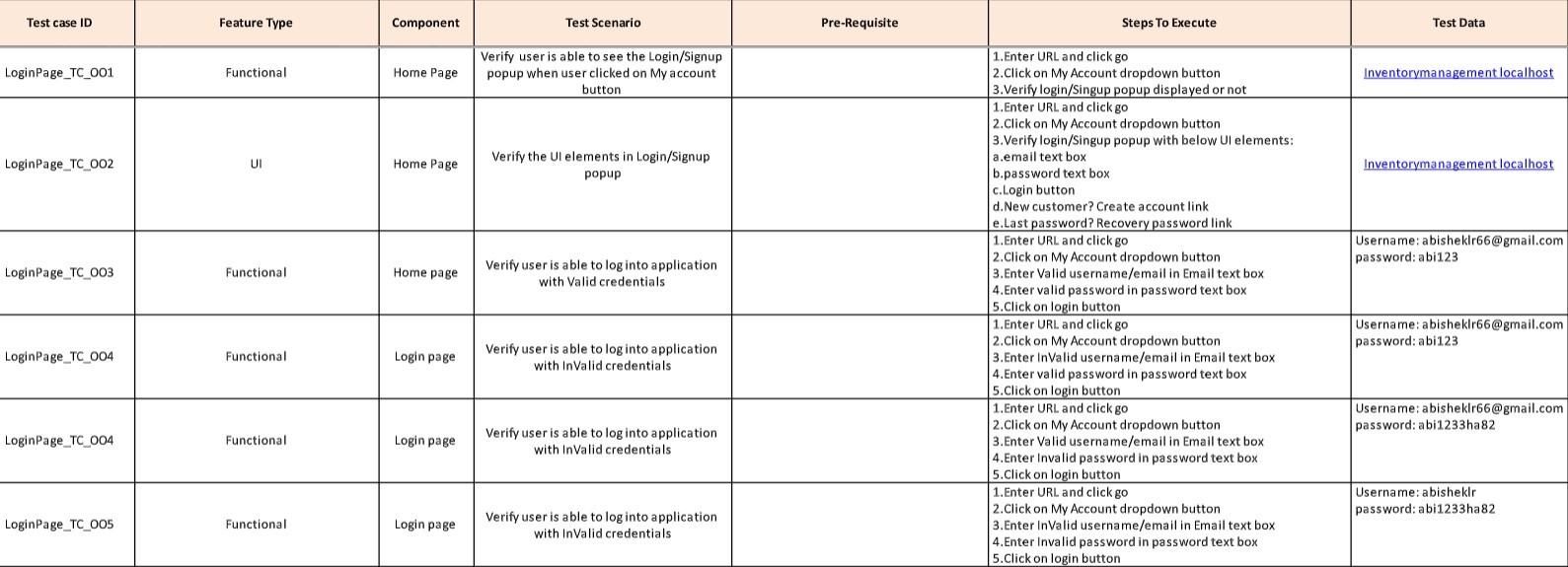
#### Update Products

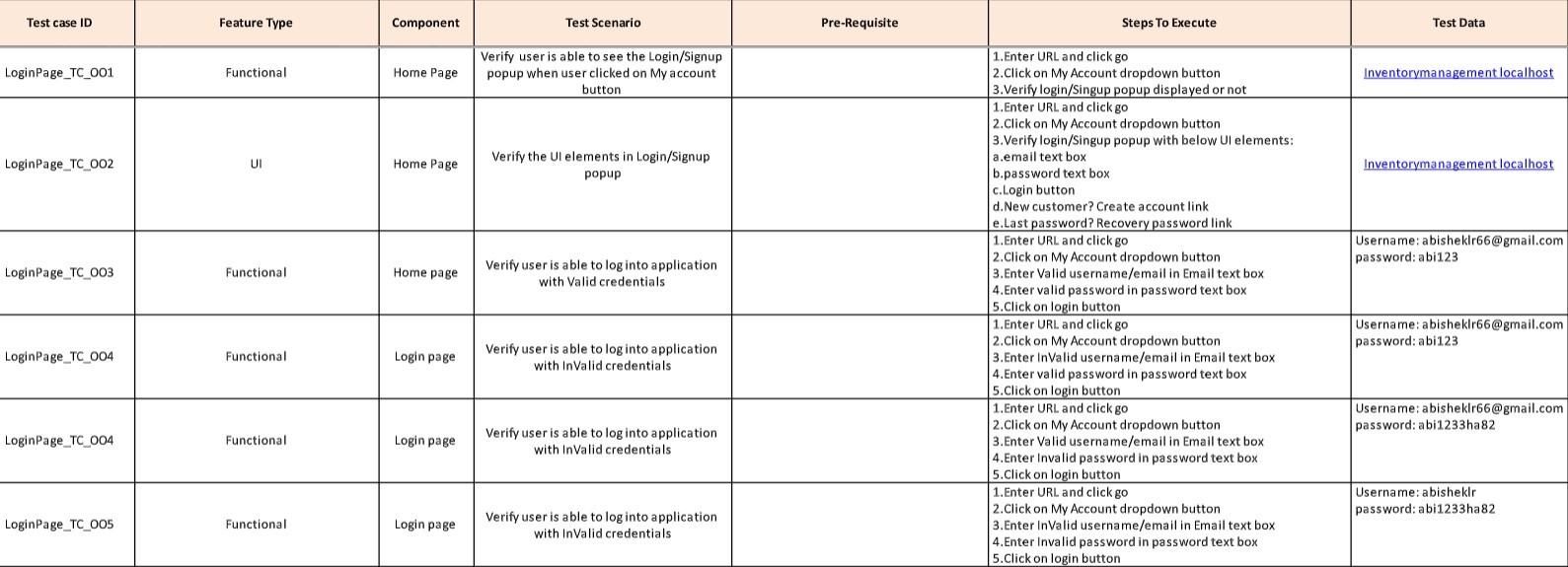
1. Verify user is able to update products in update page
2. Verify whether updated product details are correctly updated in the table
3. Verify if the product quantity is less than 5
4. Verify if an alert email has been sent to retailer if the product quantity is less

#### Delete Products

1. Verify user is able to delete product movements
2. Verify whether deleted product gets deleted from the table

##### 8.2 User Acceptance Testing





## CHAPTER 9

### RESULTS

FInally we obtained a web application for inventary system for retailers it gives the major outcome of this application is stock managing features and intimation of retailers about the stock availability. All the requirements for inventary system is obtained as mucha s possible.

## CHAPTER 10

**ADVANTAGES & DISADVANTAGES**

**10.1 ADVANTAGES:**

1. **It helps to maintain the right amount of stocks:**

Contrary to popular assumption, inventory management tries to maintain a balance where your inventory is operating at optimal efficiency and you do not need to have too many or too few inventories on hand at any given time. This helps you keep the proper quantity of stock on hand.

### 1. It leads to a more organized warehouse

It results in a more organised warehouse since you can easily organise your warehouse with the help of an effective inventory management system. It will be quite challenging to maintain your inventory if your warehouse is disorganised

1. **It saves money and time:**

A successful inventory management system can result in time and money

savings for the company. You can avoid the inconveniences of having to do an inventory recount in order to verify the accuracy of your records by keeping track of the merchandise you already have on hand.

1. **Increases productivity and efficiency:**

Inventory management tools like bar code scanners and inventory management software can significantly boost a company's productivity and efficiency.

**10.2 DISADVANTAGES:**

1. **Lack of a human touch:**

This is another drawback of inventory management. The availability of products across the globe is facilitated by large supply chain management systems, and the majority of them offer customer service support in the event of a problem, but the increase in infrastructure can frequently mean a loss of the personal touch that makes a company stand out from the competition.

1. **Increased room is required to hold the inventory:**

Unless the products you trade in are extremely small in size, a warehouse will be required to store your inventory. In addition, you will require workers, forklifts to transport the stock, and shelves and racks to store your products.

1. **Complexity:**

Some inventory management techniques and strategies might be challenging for personnel to comprehend and relatively sophisticated. Employee training may be required as a result so they can understand how the system operates.

1. **High implementation costs:**

Because the business must install specialised systems and software in order to use them, some inventory management systems can be expensive to implement.

**11. CONCLUSION**

Thus, the projected using IBM CLOUD is tested,verified and executed successfully.

### 12. FUTURE SCOPE

In feature, we planned using implemented this project in large scale which will be helpful and used by all the people.

### 13. APPENDIX

**13.1 Source Code**

### HOME.HTML

{% extends 'layout.html' %}

{% block body%}

<style>

body {

background-image: url('https://softwareauggest-blogimages.s3.ca-central-1.amazonaws.com/blog/wp-content/uploads/2016/02/14191055/9-Top-Retail-Inventory-Management-Software-for-SMEs-in-India-1068x578.png');

}

</style><br><br>

<div class="jumbotron mt-4">

<h7 class="display-4">Inventory Management 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>

<center><h5>Created By: Surendhar P K, Madhan T, Ajay Kumar S, Kiruthik Rajan K, Akash S</h5><center>

</div>

{% endblock %}

### Register.html

{% extends 'layout.html' %}

{% block body %}

<h1>Register</h1>

{% from "includes/\_formhelpers.html" import render\_field %}

<form method="POST" action="">

<div class="form-group">

{{render\_field(form.name, class\_="form-control")}}

</div>

<div class="form-group">

{{render\_field(form.email, class\_="form-control")}}

</div>

<div class="form-group">

{{render\_field(form.username, class\_="form-control")}}

</div>

<div class="form-group">

{{render\_field(form.password, class\_="form-control")}}

</div>

<div class="form-group">

{{render\_field(form.confirm, class\_="form-control")}}

</div>

<p><button type="submit" class="btn btn-primary" value="Submit">Submit</button></p>

</form>

{% endblock %}

### Login.HTML

{% extends 'layout.html' %}

{% block body %}

<h1>Login</h1>

<form method="POST" action="">

<div class="form-group">

<label>Username</label>

<input type="text" name="username" class="form-control" value={{request.form.username}}>

</div>

<div class="form-group">

<label>Password</label>

<input type="password" name="password" class="form-control" value={{request.form.password}}>

</div>

<p><button type="submit" class="btn btn-primary" value="Submit">Submit</button></p>

</form>

{% endblock %}

**Add Products:**

{% extends 'layout.html' %}

{% block body %}

<h1>Add Product</h1>

{% from "includes/\_formhelpers.html" import render\_field %}

<form action="" method="POST">

<div class="form-group">

{{ render\_field(form.product\_id, class\_="form-control") }}

</div>

<div class="form-group">

{{ render\_field(form.product\_cost, class\_="form-control", type="number") }}

</div>

<div class="form-group">

{{ render\_field(form.product\_num, class\_="form-control", type="number") }}

</div>

<p><input type="submit" value="Add" class="btn btn-primary"></p>

</form>

{% endblock %}

**Add Location:**

{% extends 'layout.html' %}

{% block body %}

<h1>Add Location</h1>

{% from "includes/\_formhelpers.html" import render\_field %}

<form action="" method="POST">

<div class="form-group">

{{ render\_field(form.location\_id, class\_="form-control") }}

</div>

<p><input type="submit" value="Add" class="btn btn-primary"></p>

</form>

{% endblock %}

**Dashboard:**

{% 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 Movement:**

{% 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 %}

**Driver Script:**

from flask import Flask, render\_template, flash, redirect, url\_for, session, request, logging

from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField, IntegerField

import ibm\_db

from passlib.hash import sha256\_crypt

from functools import wraps

from sendgrid import \*

#creating an app instance

app = Flask(\_\_name\_\_)

app.secret\_key='a'

conn=ibm\_db.connect("DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:32536;PORT=32536;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=hdb12409;PWD=hGL4pSIT9G8hudiq;",'','')

#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)

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 users(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)

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 users 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)

**Source Code**

GitHub Link **: https://github.com/IBM-EPBL/IBM-Project-53945-1661581709**

### Project demo link

https://drive.google.com/file/d/120oXb39p9m9ZwpLDvTxNVRAiwNM97o5m/view?usp=sharing