

# INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

## PROJECT REPORT

*Submitted by*

TEAM ID	PNT2022TMID00176
TEAM LEADER	SANTHOS KAMAL A B
TEAM MEMBER 1	SANTHOSH R
TEAM MEMBER 2	SANTHOSH R S
TEAM MEMBER 3	SREYANKUMAR S

# **Project Report Format**

## **1. INTRODUCTION**

- a. Project Overview
- b. Purpose

## **2. LITERATURE SURVEY**

- a. Existing problem
- b. References
- c. Problem Statement

## **3. IDEATION & PROPOSED SOLUTION**

- a. Empathy Map Canvas
- b. Ideation & Brainstorming
- c. Proposed Solution
- d. Problem Solution fit

## **4. REQUIREMENT ANALYSIS**

- a. Functional requirement
- b. Non-Functional requirements

## **5. PROJECT DESIGN**

- a. Data Flow Diagrams
- b. Solution & Technical Architecture
- c. User Stories

## **6. PROJECT PLANNING & SCHEDULING**

- a. Sprint Planning & Estimation
- b. Sprint Delivery Schedule
- c. Reports from JIRA

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

- .
  - a. Feature 1
  - b. Feature 2
  - c. Database Schema (if Applicable)

## **8. TESTING**

- a. Test Cases
- b. User Acceptance Testing

## **9. RESULTS**

- a. Performance Metrics

## **10. ADVANTAGES & DISADVANTAGES**

## **11. CONCLUSION**

## **12. FUTURE SCOPE**

## **13. APPENDIX**

Source Code

GitHub & Project Demo Link

## **CHAPTER-1**

### **INTRODUCTION**

#### **1.1 Project Overview:**

This is the basic advance level of inventory management here in inventory management we can help people to manage their own shops inventory from where ever they want. They only need a required system with the software installed if they want, they can just use our web application in order to use our inventory management system. Internet is mandatory in order to access and work in your inventory management account , instead of working shifted times they can monitor their inventory form their home or from their vocation where ever the place they want .Thiswill drastically reduce the travelling expenses of the user instead of travelling to their shop they can simply check that out from their house itself, And it also makes the work very easier it shows in the tabled manner so that the user don't need to check on the inventory system to know the current stock in the inventory .The stock needed will be mailed to the userso that they don't want to check the inventory regularly instead the warning is given to the user by the system.

#### **1.2 Purpose:**

By following this inventory management system, the work of the user is minimized to none and the comfort of the user is increased and he doesn't need to monitor the work done or the work happening in the inventory the whole work is done by the inventory management system. He only need to refill the stock in the inventory. So that the stock in the inventory doesn't exhaust and the user can peacefully carry on his business. The inventory management system not only manages the stock of the

inventory but also helps the user to stay updated in the stock and the amount of stock needed in the inventory so that the user can keep up on it. The refilling must be done manually by the user the inventory management system can only be able to intimate the user, The rest is in the user's hand they need to take care of the amount of stock to be refilled and contact the required dealer in order to buy the stock and update the stock in his own inventory in order to avoid the continuous warning of the inventory management system.

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 main use of the inventory management is to serve the people and to make the work easier. In the current scenario, if a customer does not find the desired merchandise at one retail shop, he has a second brand to rely on. A retailer can't afford to lose even a single customer.

## **CHAPTER-2**

### **LITERATURE SURVEY**

A literature survey is a comprehensive summary of previous research on a topic. The literature survey surveys scholarly articles, books, and other sources relevant to a particular area of research. The survey should enumerate, describe, summarize, objectively evaluate and clarify this previous search.

A literature survey is an overview of the previously published works on a topic. The term can refer to a full scholarly paper or a **section** of a scholarly work such as a book, or an article. Either way, a literature survey is supposed to provide the researcher/author and the audiences with a general image of the existing knowledge on the topic under question. A good literature survey can ensure that a proper research question has been asked and a proper theoretical framework and/or research methodology have been chosen. To be precise, a literature survey serves to situate the current study within the body of the relevant literature and to provide context for the reader. In such case, the survey usually precedes the methodology and results sections of the work.

#### **2.1 Existing Problem:**

The main problem in inventory management is to maintain the stock in the inventory the user finds it difficult to maintain the stock in the inventory all by himself so he needs an assistance in the work in order to maintain the amount of stock in the inventory so that the shop will run with the stock in the inventory. The main problem arises is the trust issues in the inventory so that the owner doesn't able to trust the employee working in the inventory so, He needs to supervise himself so that the mistakes won't happen in the inventory and the employee may have the power to move

without the knowledge of the owner is less, this may cause delay in decision making and may lead to decrease in the efficiency of the inventory management system. And the owner may or may not be reachable at the moment may lead to the surplus of goods in the inventory management so the main problem is the owner should need an assistance an exact assistance to guide him out and carry out the inventory. This makes the work a lot more easier and reduces the mental pressure to the owner or the user, so they can work in mental peace and the efficiency of the work is increased.

## **2.2 References**

SNO	TITLE	AUTHOR S	ABSTRACT	DRAWBACKS
1	Inventory management system	Anish Singh Maharjan  Mandip Humagain	This project is aimed at developing a desktop-based application named Inventory Management System for managing the inventory system of any organization. The Inventory Management System (IMS) refers to the system and processes to manage the stock of organization with the involvement of Technology system. This system can be used to store the details of the inventory, stock maintenance, update the inventory based on the sales details, generate sales and inventory report daily or weekly based.	This application is not suitable for those organization where there is large quantity of product and different level of warehouses
2	Inventory Management system	MS. Dhruvika Patel	The project has been developed to keep track of details regarding the equipment. The current project is a window based. To provide the basic services related to the supply of the equipment. The project will take care of all	Manual Errors at the time of entering the data can't be check, only the validation required w.r.t proposed system is checked

			supply order	
3	Inventory Management System	<p>Raj kumar</p> <p>Neelesh Kumar Singh</p>	<p>Inventory management system is an application which is helpful for business operate. Inventory management is a challenging problem area in supply chain management. Companies need to have inventories in warehouses in order to fulfil customer demand, meanwhile these inventories have holding costs and this is frozen fund that can be lost. Therefore, the task of inventory management is to find the quantity of inventories that will fulfil the demand, avoiding overstocks. This paper presents a case study for the assembling company on inventory management. It is proposed to use inventory management in order to decrease stock levels and to apply an agent system for automation of inventory management processes. Inventory management system (IMS) use for a departmental store.</p>	<p>. It is difficult to found records due to file management system.</p>



## **2.3 Problem Statement**

To solve the need that the shopkeepers doesn't have the systematic way to keep their records of the inventory data. In inventory system, demand is usually uncertain and the lead-time can also vary. To avoid shortages, managers often maintain a safety stock. In such situations, it is not clear what order quantities and reorder points will minimize expected total inventory cost.

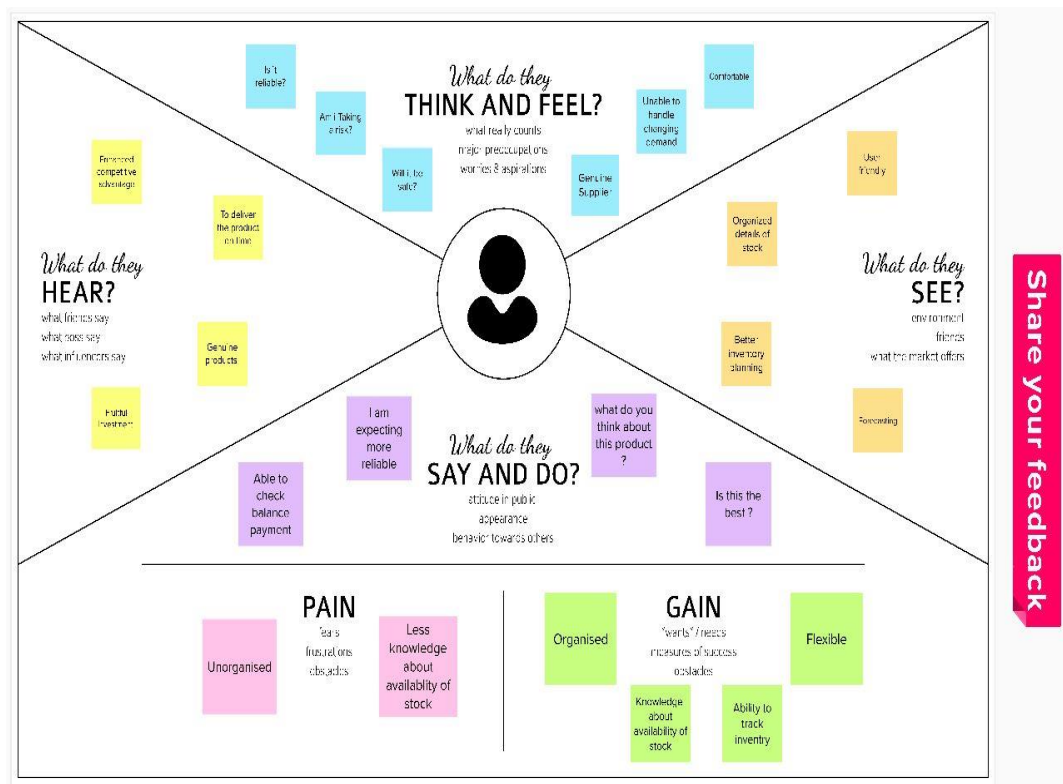
The above problem statement explains about the shopkeepers have the need to maintain their inventory in the easier and efficient way so they can use it according to their convenience. They may lead to loss of records or the error in the records stored in it. And also, they can handle it only if they physically present there and they needed to store the records for the future reference else the information may be altered or they may be lost, they need to be protected so that, they can use it for big data analysis and use it accordingly in order to customize the amount of stock to be refilled in the inventory according to the user and the records may help in the future inference. They do not need to store any physical records with them the inventory management itself stores the records of the user in the dedicated cloud so that the user can use the inference of the data stored in the cloud according to their needs in their desired ways so that the user can utilize every resource in the inventory management system itself he /she doesn't need to take any back up of the documents or the sales records of the data nor the stock records too. It intimates the user to refill the inventory so the user just needs to be active in order to perform that action so the inventory doesn't need to refill it manually another time.

## CHAPTER 3

### IDEATION & PROPOSED SOLUTION

#### 3.1 Empathy Map Canvas:

Empathy map is the graphical representation of the project. It can simply explain the process of the process so that the new comer may easily understand the undergoing process of the project and the personal opinion of the user.

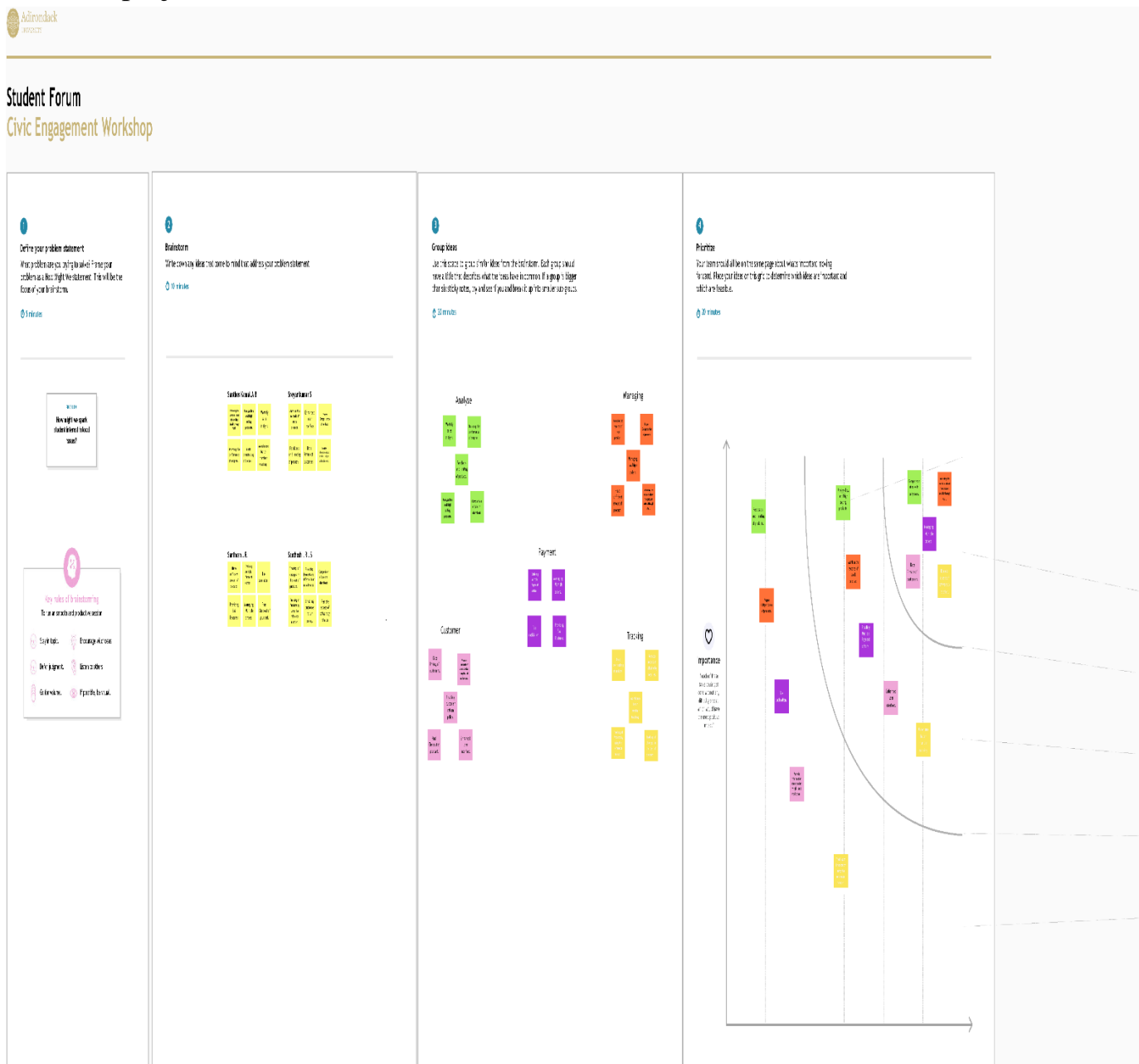


## Empathy Map

### Empathy map by keeping user on their shoes

### 3.2 Ideation & Brainstorming:

Every plan doesn't execute as automatically every thing needs a plan in order to execute and it should be verified as it held according to the plan so one needs to be supervise the work in order to maintain the flow of the project.



## **Proposed Solution**

Proposed solution is the rough idea of the project to give the multiple solution of the project and the best one is selected in order to obtain the best outcome of the project to produce the maximum efficiency.

<b>S. No.</b>	<b>Parameter</b>	<b>Description</b>
1	Problem Statement	The problem statement aims to make desktop application for retailers and to track all areas of Inventory Management System like purchase details , sales details , stock management and other policies
2	Idea / Proposed Solution	An application which retailers successfully log in to the application, that 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 the stock reduced to the limited amount found in the inventory. So that they can order new stock.
3	Novelty / Uniqueness	It connects people virtually and give information like purchase details, sales details, stock management. Act as all-in-one place for stock retail and purchases
4	Social Impact / Customer Satisfaction	It helps Retailers to manage the entire stock warehouse and a single platform. It also helps in managing the about demand and supply of their Inventories. Proper Inventory Management System reduces the risk of over stocking and prevents wastage of capital and stock of the Retailers..

5	Business Model	Inventory management system helps Retailers to identify which and how much stock to order at what time. Cuts off the unwanted cost for the Retailers.
6	Scalability of the Solution	These solutions streamline order management as they enable business to automatically reorder stock that is running low before it runs out, ensuring no sales opportunities are missed. On the other hand, this also means that overstocking can be avoided if certain products aren't selling as expected.

### 3.3 Problem solution fit:



## **CHAPTER-4**

### **REQUIREMENT ANALYSIS**

An easy-to-use interface that doesn't require advanced training, support or documentation. Automation for eliminating manual processes of business functions related to inventory management. A reliable, secure database that provides accurate, real-time data.

#### **4.1 Functional Requirements:**

1. **User Registration** - The Users can make the Registration through form and Registration through Email.
2. **User Confirmation** – The Users Get Confirmation via Email and Confirmation via OTP.
3. **Login** – Login to the application by entering the Email and Password.
4. **Dashboard** – By opening the Dashboard the list of products display and User can view the product availability.
5. **Add items to cart** – User as they wish to buy the product, they can select the product and add it to the cart.
6. **Stock Update** – If the desired product is unavailable, they can update the products into the list for buying products.

#### **4.2 Non-Functional Requirements:**

7. **Usability** – While usability determines how effective implementing an inventory tracking system is in your business. If it takes hours for your staff to learn the ins and outs of the software, then it's probably not worth buying.
8. **Security** - The process of ensuring the safety and optimum management control of stored goods. It is of central importance for optimum warehouse

management because the performance of a company stands or falls with the safety and efficiency of a warehouse.

9. **Reliability** - Relying on manual inventory counts to know what you have will only guarantee high inefficiencies and a loss of customers.
10. **Performance** - Creating systems to log products, receive them into inventory, track changes when sales occur, manage the flow of goods from purchasing to final sale and check stock counts.
11. **Availability** - Whether a specific item is available for customer orders. Additional information provided by retailers may include the quantity available.
12. **Scalability** - They should use an automated inventory management system for inventory tracking. This will make your business much more scalable so that you can continue building consistent growth and take advantage of increased sales.

## **CHAPTER-5**

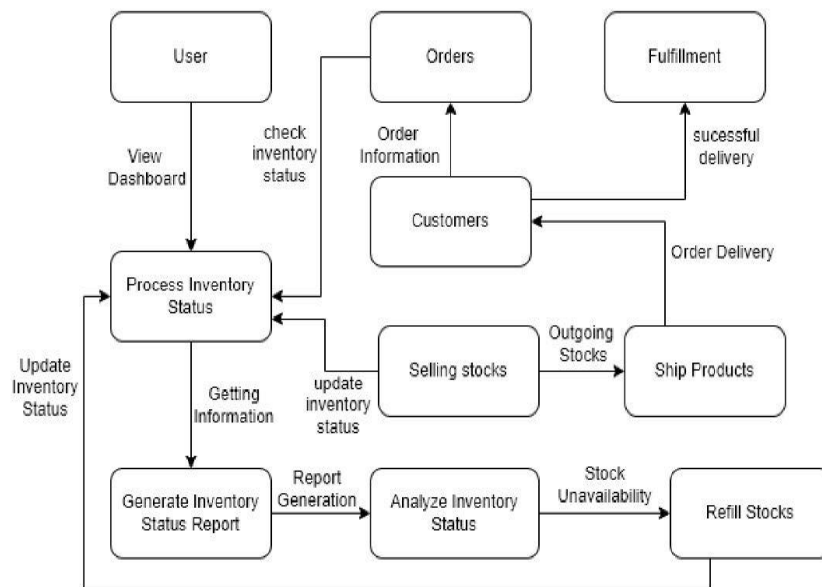
### **PROJECT DESIGN**

#### **5.1 Data flow diagram:**

The term data flow diagram shows the path of the traveling in the project done or explaining the total working of the project it also explains the path of the travel or the working process of the project and the process that held inside the project.

#### **Data flow diagram for inventory management**

The above shown diagram shows the data flow of the inventory



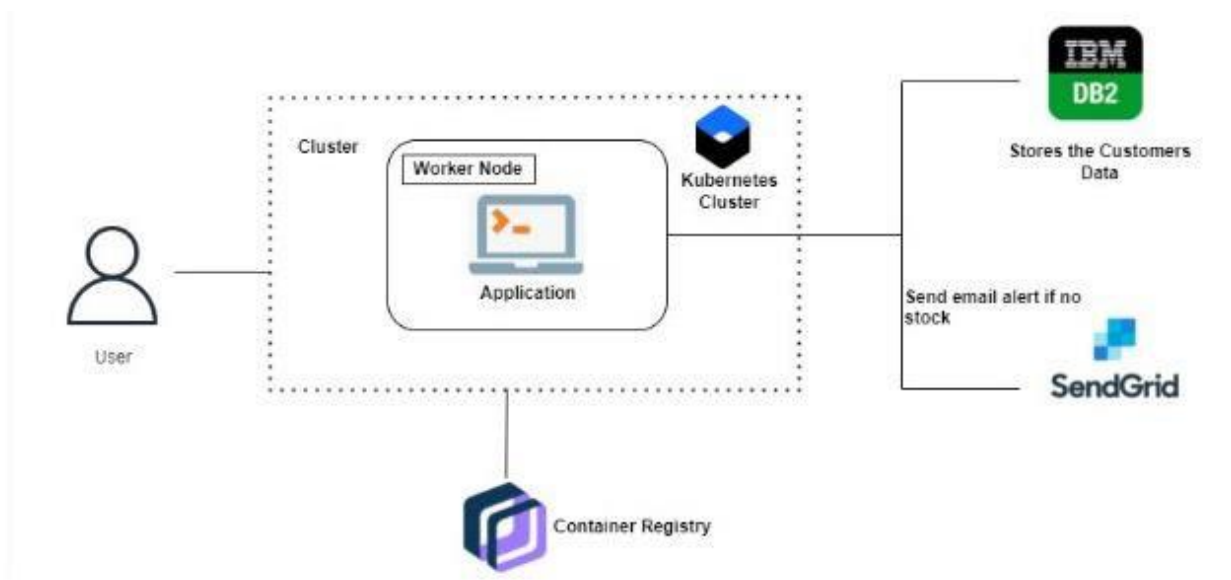


management system the user must register himself in order to work in the inventory management system. After registering himself he must login to the dashboard and check the product availability and decide whether to buy the product or not in case of buying they need to check on the product and they can add the desired product to their cart and they may proceed for buying. If they didn't find their desired product they are allowed to mention the product that they needed in order to make the product available for the next time.

## **5.2 Solution & Technical architecture:**

The term solution and the technical architecture defined as the solution of the problem mentioned and the way that the solution that has to be implemented on the way in order to achieve the goal and to solve the problem.

### **Process of inventory management**



The above shown diagram is the technical architecture of the inventory management system. It explains about the inventory management system so that the user can run a shop using a send grid so according to the amount of stock given by the user the inventory management system warns the user via mail only if the product comes to the down range. After refilling the stock in the inventory it has to be updated in the cloud database db2. It has to estimate the available products after refilling it will count on the exhaustion of the product. It calculates the estimated products sold by the shop owners these calculations may be done on the google sheets or any other tabled format use to perform calculations. According with that it also do the customer survey or the product that customer wants that they didn't get now was noted they are added to the inventory with the acknowledgement of the user or the owner.

### **5.3 User stories:**

The below mentioned are the user stories of the inventory management and the user experience of the individual so that each opinion may be taken under survey so that we can update the inventory management according to the desire of the customer so that they can even work efficiently and comfortably.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web 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	Sprint1
		USN-2	As a user, I can register for the application through E-mail	I can access my account / dashboard	Medium	Sprint1

	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	I can get confirmation for my email and password and create authenticated account.	Medium	Sprint1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can log onto the application with verified email and password	High	Sprint1
	Dashboard	USN-5	As a user, I can view the products which are available	Once I log on to the application, I can view products to buy.	High	Sprint2

## **CHAPTER-6**

### **PROJECT PLANNING & SCHEDULING**

#### **6.1 Sprint Planning & Estimation**

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

**Table of sprint planning**

#### **6.2 Sprint Delivery Schedule**

This topic explains about the delivery info of the sprint and the time taken to complete the sprint separately.

##### **6.2.1 Product Backlog, Sprint Schedule, and Estimation:**

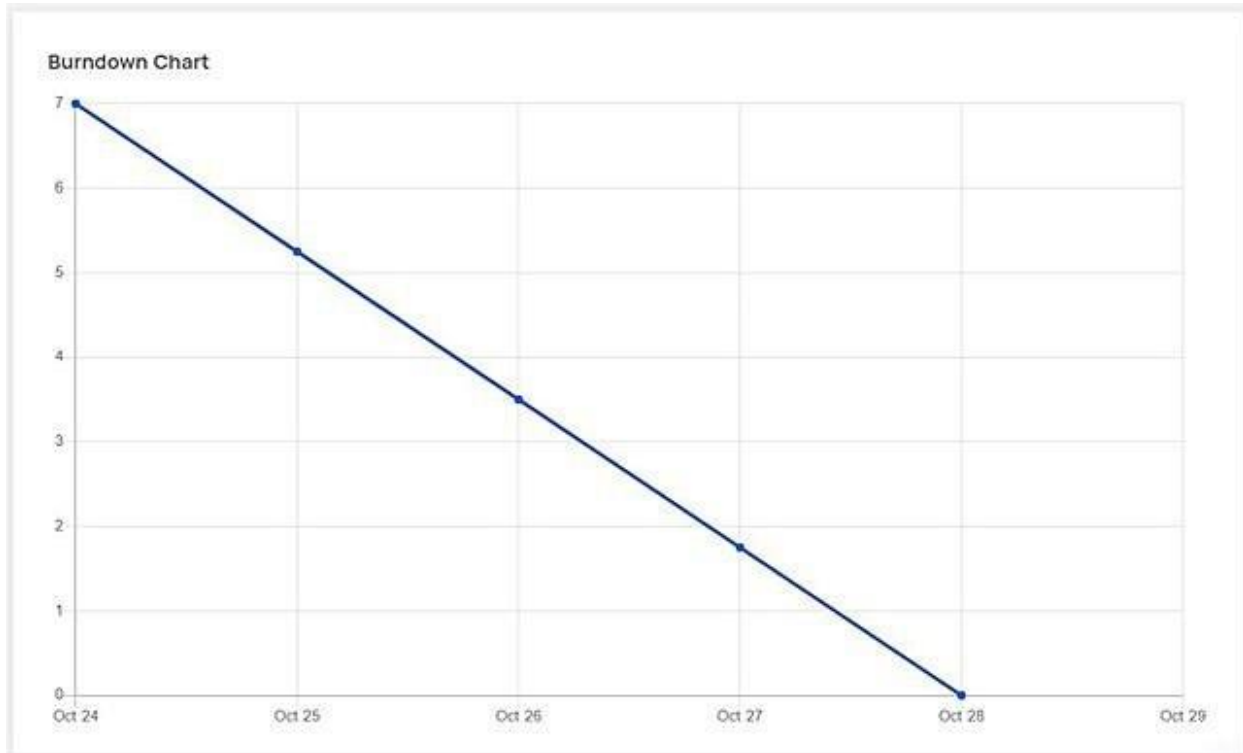
The above planned sprint must be delivered on time so the more important is to deliver the sprint correctly. So the product must satisfy the customer so a survey must be held and collect the needs of the user and priori the needs of the user and also note the requirements of the system. The above mentioned are tabulated below

<b>Sprint</b>	<b>Functional Requirement (Epic)</b>	<b>User Story Number</b>	<b>User Story / Task</b>	<b>Story Points</b>	<b>Priority</b>	<b>Team Members</b>
Sprint1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	5	High	Santhos Kamal. AB
Sprint1		USN-2	As a user, I can register for the application through E-mail	5	High	Santhosh.R
Sprint1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	5	Medium	Santhosh.RS
Sprint1	Login	USN-4	As a user, I can log into the application by entering email	5	High	SreyanKumar.S

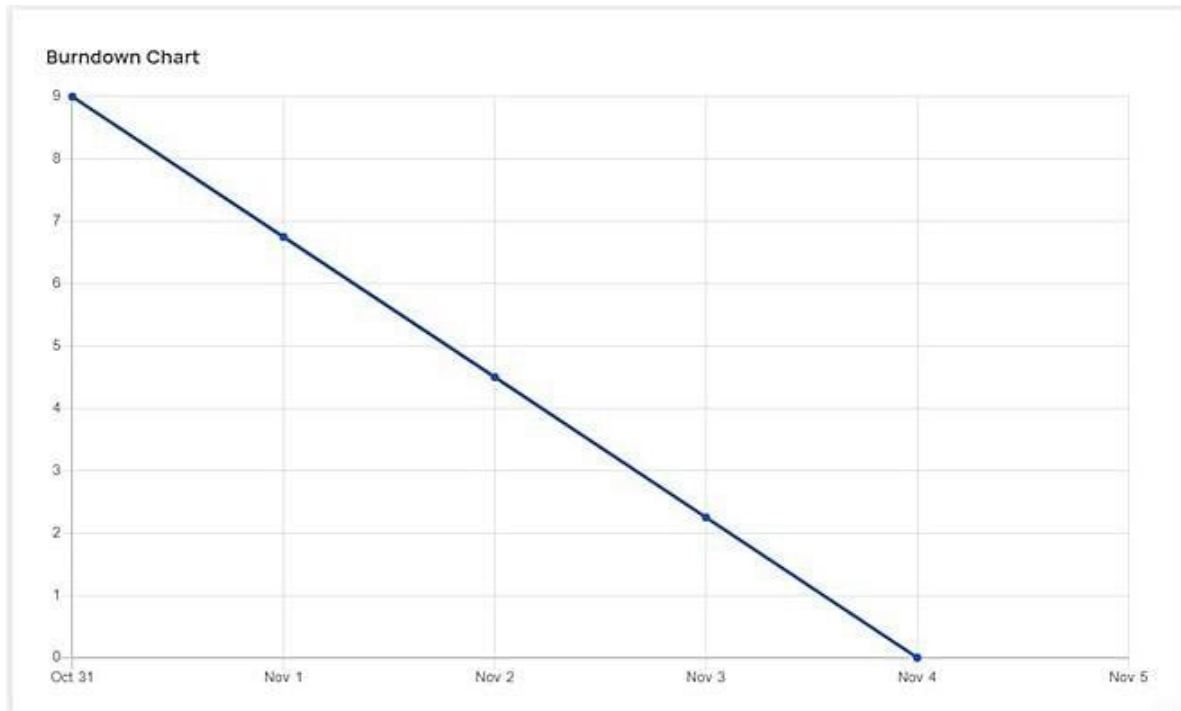
Sprint2	Dashboard	USN-5	As a user, I can view the products which are available	10	High	Santhos Kamal. AB Santhosh.RS
Sprint2	Stocks update	USN-6	As a user, I can add the products which are not available in the inventory and the restock the products to make them available	10	Medium	Santhosh. R Sreyan Kumar. S
Sprint3	Sales prediction	USN-7	As a user, I can be able to get access to sales report to predict the which product runs out of stocks fast . So it helps me to predict the restock of product	20	High	Santhos Kamal. AB Santhosh. R Santhosh.RS Sreyan Kumar. S
Sprint4	Customer Care	USN-8	As a user, I am able to get in touch with customer care to enquire the doubts and problems I have faced	10	Medium	Santhos Kamal. AB Santhosh. R
Sprint-4	Feedback	USN-9	As a user, I am able to send feedback to improve the system	10	Medium	Santhosh.RS Sreyan Kumar. S

### **6.2.2 Project Tracker, Velocity & Burndown Chart**

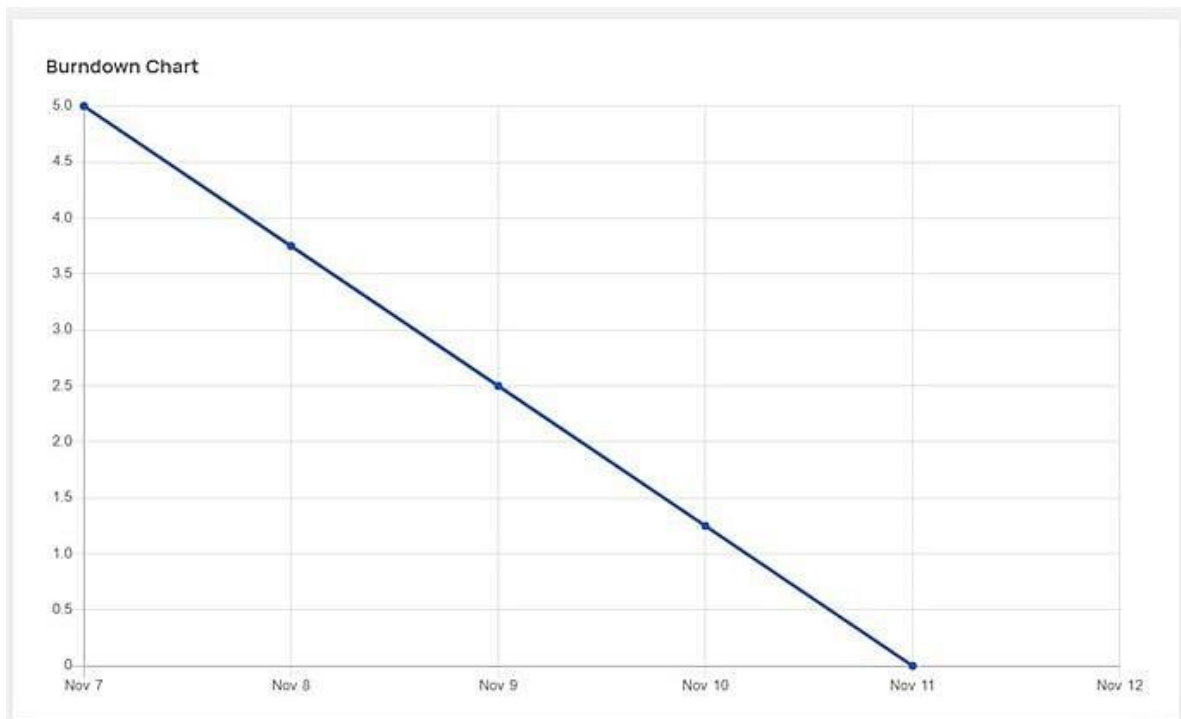
The below shown respective diagrams are the burndown charts of the sprint planning and the execution graph of the sprint.



**Sprint 1(registration, confirmation, login)**

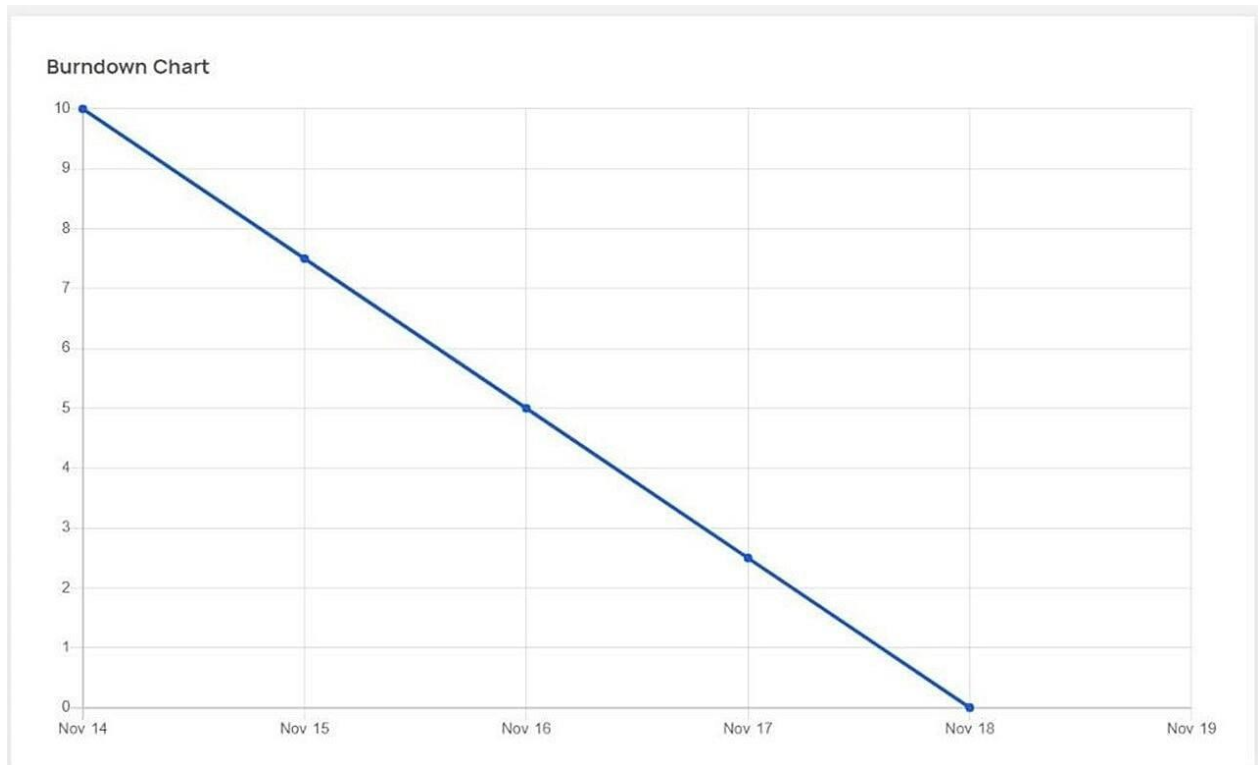


**Sprint 2(dashboard, add items to cart)**



**Sprint 3(stock update)**



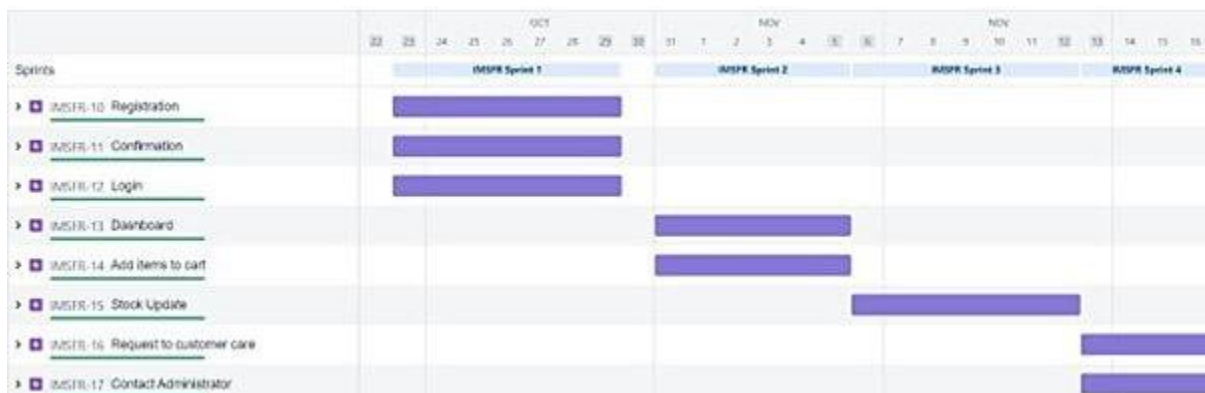


### **Sprint 4(request to customer care, contact administrator)**

### **6.3 Reports from JIRA:**

The planning of the sprint individually and the work done and the time taken to do that work is mentioned in the below picture.

The JIRA reports are mentioned below,

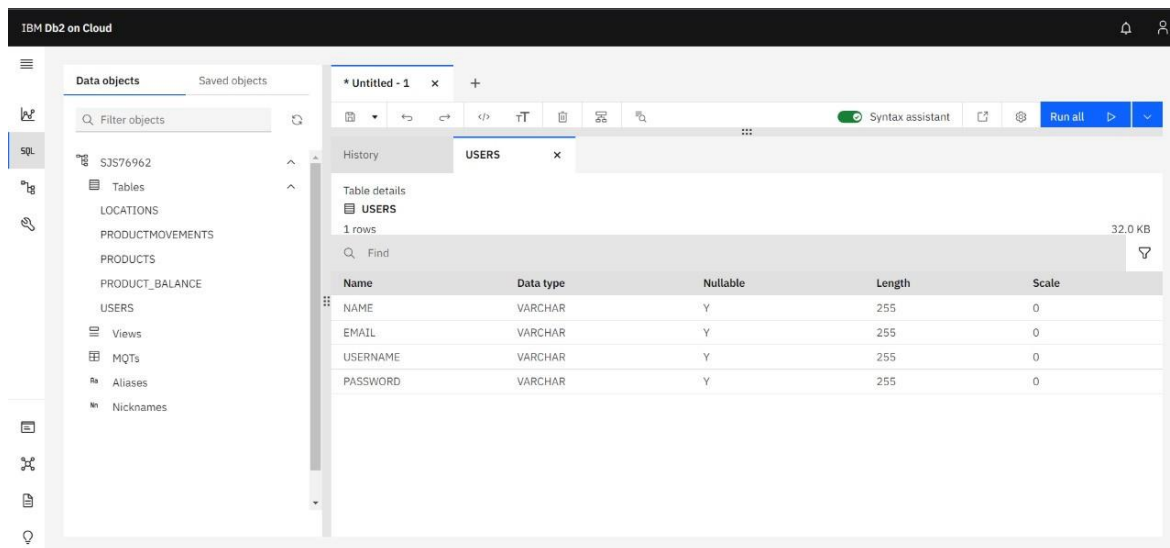


## **CHAPTER-7**

### **CODING & SOLUTIONING**

#### **7.1 FEATURES 1:**

In inventory management the mandatory need is the database so the user needs to store the info of the shop. The inventory management stores the data of the inventory in a tabled format in IBM DB2 provided by the IBM itself, which stores the information .



#### **IBM db2 storage**

And for object storage we have used IBM Cloud Object Storage which stores images and cascading style sheets(css) files for the system.

IBM Cloud

Cloud Object Storage

Storage instances

Cloud Object Storage-c0

Buckets

Integrations

Endpoints

Usage details

Service credentials

Connections

Plan

Search resources and products...

Catalog

Manage

SANTHOSH R S's Acco...

cloudassg3

Transfers

Details

Actions...

Objects

Configuration

Permissions

Warning: All objects in this bucket have public view access.

If you're seeing more usage than expected, versions count towards your usage or you may have incomplete uploads [Learn more](#)

Prefix filter

Upload

<input type="checkbox"/>	Object name	Archived ⓘ	Size	Last modified	
<input type="checkbox"/>	tech1.jpeg		8.2 KB	2022-10-29 10:54 AM	⋮
<input type="checkbox"/>	tech2.webp		150.5 KB	2022-10-29 10:53 AM	⋮
<input type="checkbox"/>	tech3.jpeg		5.7 KB	2022-10-29 10:54 AM	⋮
<input type="checkbox"/>	tech4.jpeg		6.2 KB	2022-10-29 10:54 AM	⋮
<input type="checkbox"/>	tech5.jpeg		5.2 KB	2022-10-29 10:54 AM	⋮

Drag and drop files (objects) here or click to upload

## IBM object storage

## **CHAPTER-8 TESTING**

### **8.1 TEST CASES:**

#### **Purpose of Document**

The purpose of this document is to briefly explain the test cover age and open issues of the Corporate Employee Attrition Analytics project at the time of the release to User Acceptance Testing (UAT).

#### **Defect Analysis**

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

Resolution	Severity1	Severity2	Severity3	Severity4	Subtotal
By Design	12	4	2	7	25
Duplicate	0	0	0	2	2
External	2	3	0	1	6
Fixed	11	5	4	15	35
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	0	0	1	1
Totals	35	12	8	27	72

## **Test Case Analysis**

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

Section	Total Cases	Not Tested	Fail	Pass
Home page	4	0	0	4
Register Page	5	0	1	5
Login Page	5	0	0	5
Admin Login Page	4	0	0	4
Index Page	3	0	0	3
Products list page	5	0	0	3
Update Product Page	4	0	0	4
Contact Page	3	0	0	3
Complaint Page	3	0	0	3
Admin Index page	2	0	0	2
Admin Product Page	3	0	0	3
Logout from User Page	2	0	0	2
Logout from Admin page	2	0	0	2

## **8.2 USER ACCEPTANCE TESTING:**

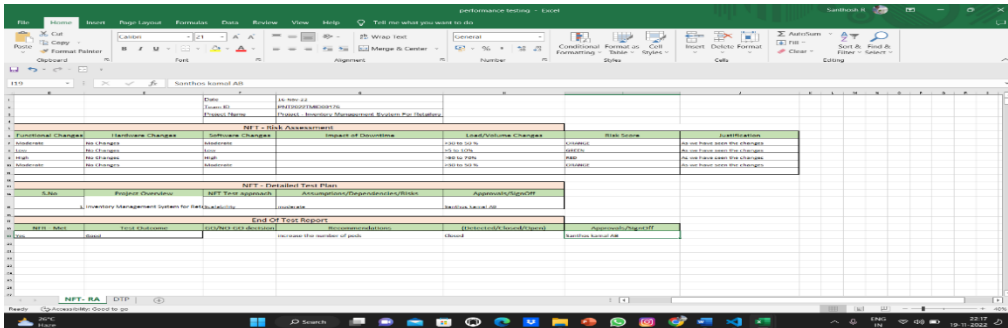
This is table that explaining the user's opinion about the functions of the components of the inventory management system and the detailed report is uploaded in the git hub repositories.



## CHAPTER 9

### RESULTS

#### 9.1 PERFORMANCE METRICS:



performance testing - Excel						
Functional Changes		Hardware Changes		Software Changes		Impact of Database
NPT - Risk Assessment						
Functional Changes	Hardware Changes	Software Changes	Impact of Database	Load/Volume Changes	Risk Score	Justification
increase	no changes	increase	increase	20% to 30%	increase	no test data for changes
decrease	no changes	decrease	decrease	20% to 30%	decrease	no test data for changes
no change	no changes	no change	no change	20% to 30%	no change	no test data for changes
NPT - Detailed Test Plan						
Test Case		Test Data		Test Results		Test Status
Test Case		Test Data		Test Results		Test Status
End Of Test Report						
Test Case		Test Data		Test Results		Test Status
Test Case		Test Data		Test Results		Test Status

#### Performance metrics

The above is the performance metrics evaluation of our application calculated by the Gatling.

The is the report generated by the Gatling – Professional Load Testing Tool.

## **CHAPTER-10**

### **ADVANTAGES & DISADVANTAGES**

#### **10.1 Advantages**

1. It helps to maintain the right amount of stocks
2. It leads to a more organized warehouse
3. It saves time and money
4. Improves efficiency and productivity
5. A well-structured inventory management system leads to improved customer retention
6. Schedule maintenance
7. Reduction in holding costs
8. Flexibility - the ability to easily adjust production levels, rawmaterial purchases, and transport capacity

#### **10.2 Disadvantages**

1. Bureaucracy - The term bureaucracy refers to a complex organization that has multi layered systems and processes.
2. Impersonal touch
3. Increased space is need to hold the inventory
4. High implementation costs
5. Complexity - Longer / less reliable lead times from global suppliers.



## **CHAPTER – 11**

### **CONCLUSION**

Inventory management has to do with keeping accurate records of goods that are ready for shipment. This often means having enough stock of goods to the inventory totals as well as subtracting the most recent shipments of finished goods to buyers. When the company has a return policy in place, there is usually a sub-category contained in the finished goods inventory to account for any returned goods that are reclassified or second grade quality. Accurately maintaining figures on the finished goods inventory makes it possible to quickly convey information to sales personnel as to what is available and ready for shipment at any given time by buyer. Inventory management is important for keeping costs down, while meeting regulation. Supply and demand is a delicate balance, and inventory management hopes to ensure that the balance is undisturbed. Highly trained Inventory management and high-quality software will help make Inventory management a success. The ROI of Inventory management will be seen in the forms of increased revenue and profits, positive employee atmosphere, and on overall increase of customer satisfaction.

## **CHAPTER – 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, store and van stock management is being revolutionised with the use of inventory and supply-chain management apps on mobile devices. However, not all app-based inventory management systems are the same. Leading- edge app-based Inventory Management systems: make use of the best technologies to manage the process, Inventory management is vital for retailers because the practice helps them increase profits. They are more likely to have enough inventory to capture every possible sale while avoiding overstock and minimizing expenses. From a strategic point of view, retail inventory management increases efficiency.

1. Invest in an inventory management system.
2. Set up stock alerts.
3. Select suppliers strategically.
4. Implement SKU management practices.
5. Optimize your order size.
6. Consider drop shipping.

## **CHAPTER – 13**

### **APPENDIX**

The system analyst interviewed various officials like manager of the store, dealing hand, and users to discuss present way of working and possible improvement in the system. During the discussions it was found that first user has to place a request on pre-printed form to store manager through the concerned head of the section. Depending on the availability, item will be issued to user otherwise NOC is issued. Quantity of item issued is subtracted from the balance and when item is received from the supplier quantity is added in the balance. A special ledger is maintained to records issue and receipt entries of items. When the particular item reaches below reorder level procurement procedure starts to replenish the item. ABC analysis of inventory items is performed on regular basis which is time consuming process. The management also finds fast moving and slow items from time to time so that sufficient stock of fast moving items can be maintained and slow moving items can be reduced from the inventory. To facilitate inventory management, ABC analysis classifies the inventory items into following three classes based on the consumption value:

- A. Class A: These items constitute the most important class of inventories so far as the proportion in the total value of inventory.
- B. Class B: These items constitute an intermediate position, which constitute approximately 20% of the total items, accounts for approximately 20% of the total material consumption value.
- C. Class C: It consists remaining 70% items, accounting only 10% of the monetary value of total material usage. Quite relaxed inventory procedures are used.

### **Expected outputs from the system**

- A. Current balance of item in stock.
- B. List of items below reorder level.
- C. Item wise cost of inventory.
- D. List of spoiled items.
- E. List of items received and issued within a specific period.
- F. Ledger preparations as required by management
- G. ABC analysis of inventory items.
- H. Placement of orders for replenishment of items

### **Input requirements**

- A. Details of Inventory items
- B. Details of items received
- C. Details of items issued
- D. Details of issue form
- E. Details of items ordered for procurement
- F. Details of potential vendors

### **Methods and procedure**

Computing Current Balance:  $Current\ balance = Current\ Balance + Receipt - Issue$

Computing Reorder Level:  $Reorder\ Level = Daily\ Demand * Procurement\ Time$

$Daily\ Demand = Yearly\ Demand / No.\ of\ Working\ Days$

Procurement time is generally constant

## Source code:

### login.html:

```
<h1>Login</h1>
<form method="POST" action="">

    <section class="vh-100">
        <div class="container-fluid h-custom">
            <div class="row d-flex justify-content-center align-items-center h-100">
                <div class="col-md-9 col-lg-6 col-xl-5">
                    
                </div>
                <div class="col-md-8 col-lg-6 col-xl-4 offset-xl-1">
                    <p class="text-center h1 fw-bold mb-5 mx-1 mx-md-4 mt-4">Login</p>
                    <form>

                        <!-- username input -->
                        <br>
                        <div class="form-outline mb-4">
                            <input type="text" id="form3Example3" name="username" class="form-control form-control-lg"
                                placeholder="Enter a username" value="{{request.form.username}}">
                            <!-- <label class="form-label" for="form3Example3">Email address</label> -->
                        </div>

                        <!-- Password input -->
                        <div class="form-outline mb-3">
                            <input type="password" id="form3Example4" name="password" class="form-control form-control-lg"
                                placeholder="Enter password" value="{{request.form.password}}"/>
                            <!-- <label class="form-label" for="form3Example4">Password</label> -->
                        </div>

                        <div class="d-flex justify-content-between align-items-center">
                            <!-- Checkbox -->
                            <div class="form-check mb-0">
                                <input class="form-check-input me-2" type="checkbox" value=""
                                    id="form2Example3" />
                                <label class="form-check-label" for="form2Example3">
                                    Remember me
                                </label>
                            </div>
                        </div>

                    </form>
                </div>
            </div>
        </div>
    </section>
</form>
```

```

        <a href="#" class="text-body">Forgot password?</a>
    </div>

    <div class="text-center text-lg-start mt-4 pt-2">
        <button type="submit" class="btn btn-primary btn-lg"
            style="padding-left: 2.5rem; padding-right: 2.5rem;"
value="Submit">Login</button>
        <p class="small fw-bold mt-2 pt-1 mb-0">Don't have an account? <a
href="{{ url_for('register') }}"
            class="link-danger">Register</a></p>
    </div>

    </form>
</div>
</div>
</div>

</section>
</form>

```

## Register.html:

```

<table>

{% block body %}

<h1>Register</h1>
{% from "_formhelpers.html" import render_field %}
<form method="POST" action="">
    <table>
        <td>
            
        </td>
        <td>
            <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>
</td>
</table>
</form>
</table>

```

### app.py:

```

from flask import Flask, render_template, flash, redirect, url_for, session, request,
logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators,
SelectField, IntegerField
import ibm_db
from passlib.hash import sha256_crypt
from functools import wraps

from sendgrid import *

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

app.secret_key='a'

# conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=9938aec0-8105-433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32459;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UID=gkx39361;PWD=lceiXiiRK9Vwzqgq",'','')
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=2d46b6b4-cbf6-40eb-bbce-
6251e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32328;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UID=sjs76962;PWD=QgoTQVcIHv7xh1qA",'','')

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

# @app.route('/layout')
# def layout():
#     return render_template('layout.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)
        # print(product_num)
        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 = [(1,1) for 1 in locs]
    form.from_location.choices.append(("Main Inventory", "Main Inventory"))
    form.to_location.choices = [(1,1) for 1 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)
    print(current_num)

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

    alert_num=int(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,int(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:

```



```

        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)
    print(sql2)
    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)

```

## **GIT HUB:**

The repositories of the git hub link is given below ,you can refer this below link to know more info:

<https://github.com/IBM-EPBL/IBM-Project-10806-1659207640>

## **project demo link:**

[https://drive.google.com/file/d/10NG6\\_jbx4fLJL9QDIxdV18lsPNmz9y9R/view?usp=share\\_link](https://drive.google.com/file/d/10NG6_jbx4fLJL9QDIxdV18lsPNmz9y9R/view?usp=share_link)