

INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

IBM PROJECT REPORT

submitted by

BENSON JOSEPH R

AAKASH A

ARUN KUMAR G

EASHAN SAI U.C

TABLE OF CONTENTS

CHAPTER NO	TITLE	PG NO
1	INTRODUCTION	4
	1.1 Project Overview	5
	1.2 Purpose	5
2	LITERATURE SURVEY	6
	2.1 Existing problem	7
	2.2 References	8
	2.3 Problem Statement Definition	9
3	IDEATION & PROPOSED SOLUTION	13
	3.1 Empathy Map Canvas	14
	3.2 Ideation & Brainstorming	15
	3.3 Proposed Solution	21
	3.4 Problem Solution fit	22
4	REQUIREMENT ANALYSIS	23
	4.1 Functional requirements	24
	4.2 Non-Functional requirements	25
5	PROJECT DESIGN	26
	5.1 Data Flow Diagrams	27
	5.2 Solution & Technical Architecture	28
	5.3 User Stories	29
6	PROJECT PLANNING & SCHEDULING	30
	6.1 Sprint Planning & Estimation	31
	6.2 Sprint Delivery Schedule	33
	6.3 Reports from JIRA	34

7	CODING & SOLUTIONING	36
	7.1 Feature 1	37
	7.2 Feature 2	39
8	TESTING	42
	8.1 Test Cases	43
	8.2 User Acceptance Testing	46
9	RESULTS	47
	9.1 Performance Metrics	48
10	ADVANTAGES & DISADVANTAGES	51
11	CONCLUSION	54
12	FUTURE SCOPE	56
13	APPENDIX	57
	Source Code	58
	GitHub & Project Demo Link	77

CHAPTER 1

INTRODUCTION

INTRODUCTION

1.1 PROJECT OVERVIEW

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

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

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

1.2 PURPOSE

The purpose of the Inventory system is to serve customers efficiently. This project aims to help the staff/employees to make their work faster by using the system where they can monitor the remaining products in the records where they can see in the database. The system will provide a good service to the company like automatically send an email alert to the retailers that brings bigger profit.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Visibility problems

When your inventory becomes hard to find, you have inventory visibility problems. Lack of visibility is one of the most common inventory management problems. Locating the correct item in the right place as quickly as possible is essential to inventory. If the hard to find inventory is part of the supply chain for manufacturing, it can impact the operations of the entire manufacturing process. If the inventory stock is being accessed for shipping and cannot be located, it leads to incomplete or wrong shipments and severely impacts customer satisfaction. Either way inventory visibility problems have a severe impact on the performance of the business and is one of the symptoms of poor inventory management.

Lack of real-time reporting

Inventory reports are essential to making decisions. An inventory department cannot summarise and report based on real-time inventory data when using a manual system. Reports on historical trends are also challenging to prepare quickly. When management cannot visualise inventory stock or trends, making informed decisions on purchase and inventory becomes tough. This directly affects the bottom line of the company.

Inefficiency

Managing inventory manually is a cumbersome and tedious process. Even the routine tasks become slower than they should be. As companies scale up, the process becomes more inefficient and slow. Manual inventory management becomes even more challenging to implement across multiple warehouse locations. Inefficient inventory management slows down operations. Inventory management problems that cause slow shipping of products leads to a fall in customer satisfaction. Even when software solutions are used, improperly designed or obsolete

systems do nothing more than merely replicate the manual process inventory management. Inefficiency and redundancy are some of the symptoms of poor inventory management.

Overstocking

Money that is spent on inventory gets locked in if the items are not used. Overstocking can impact the profitability of a business. This is because more stock is bought than being sold. Management of inventory to stock the correct quantity of items is essential to a company's financial well-being. Overstocking also results in the buildup of obsolete stock. This is the material that has been bought or stocked in excess and is no longer in demand. In a manual system, this stock may be abandoned or forgotten. When it is in demand again, the company may buy more of it instead of using what is already stocked.

2.2 REFERENCES

1. J. D. Sterman y G. Dogan, “‘I’m not hoarding, I’m just stocking up before the hoarders get here.’: Behavioural causes of phantom ordering in supply chains”, *Journal of Operations Management*, vol. 39, pp. 6–22, 2015.
2. Y. Wang, S. W. Wallace, B. Shen, y T.-M. Choi, “Service supply chain management: A review of operational models”, *European Journal of Operational Research*, vol. 247, núm. 3, pp. 685–698, 2015.
3. S. Mahar y P. D. Wright, “The value of postponing online fulfilment decisions in multi-channel retail/e-tail organizations”, *Computers & operations research*, vol. 36, núm. 11, pp. 3061–3072, 2009.
4. A. Hübner, A. Holzapfel, y H. Kuhn, “Operations management in multi-channel retailing: an exploratory study”, *Operations Management Research*, vol. 8, núm. 3–4, pp. 84–100, 2015.
5. Hübner, H. Kuhn, J. Wollenburg, y A. Trautrim, “From bricks-and-mortar to bricks-and-clicks—logistics networks in omni-channel grocery retailing”, *Empirical Studies in Multi-Channel and OmniChannel Retail Operations and Logistics*, p. 102, 2018.

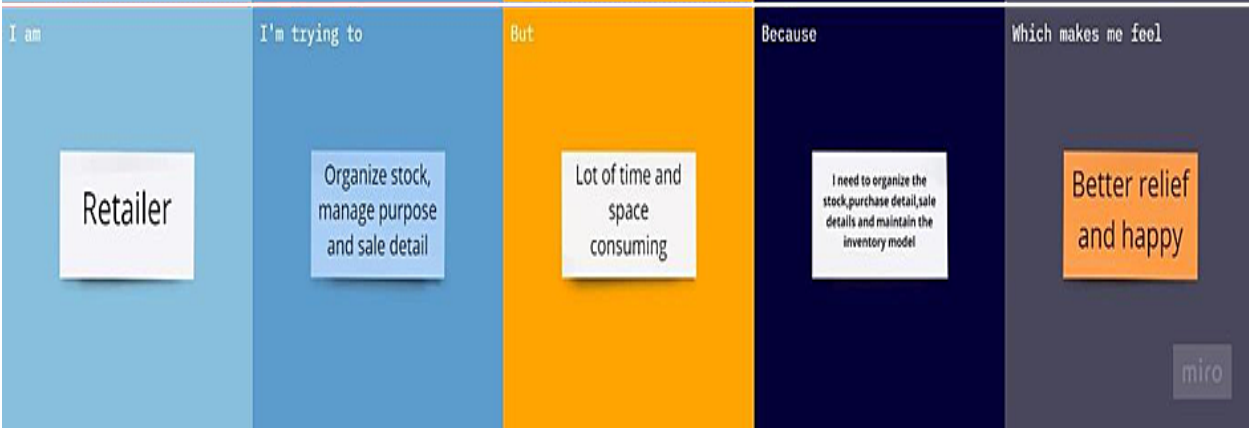
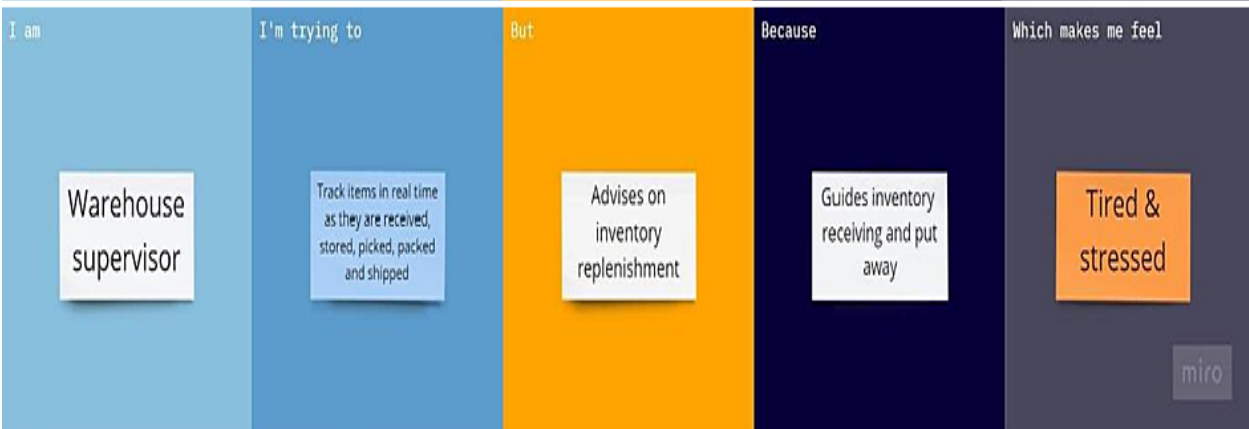
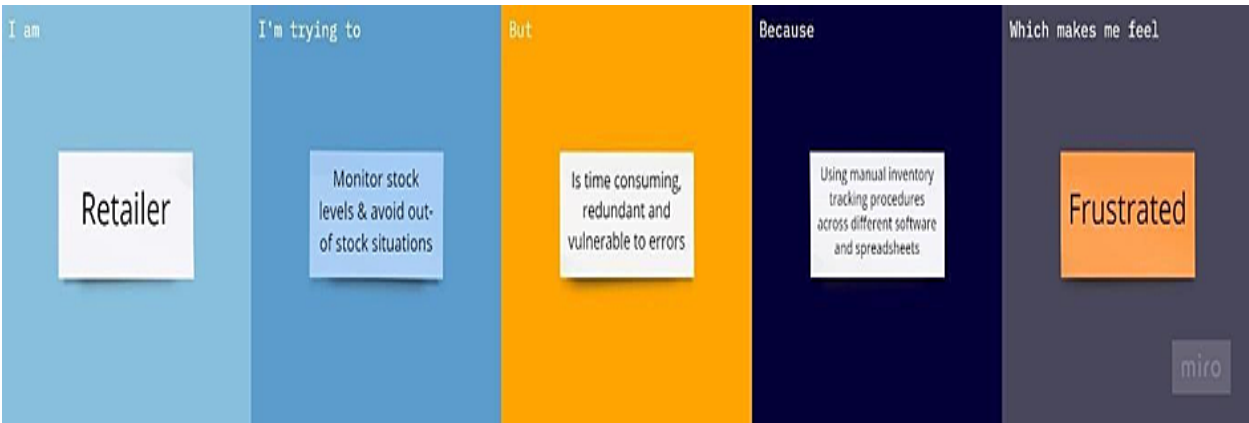
6. A. Fink, *Conducting research literature reviews: From the internet to paper*. Sage publications, 2019.
7. A. Cooke, D. Smith, y A. Booth, “Beyond PICO: the SPIDER tool for qualitative evidence synthesis”, *Qualitative health research*, vol. 22, núm. 10, pp. 1435– 1443, 2012.
8. W. Zhou y S. Piramuthu, “Effects of ticket-switching on inventory management: Actual vs. information system-based data”, *Decision Support Systems*, vol. 77, pp. 31–40, sep. 2015, doi: 10.1016/j.dss.2015.05.010.
9. W. Zhang y K. Rajaram, “Managing limited retail space for basic products: Space sharing vs. space dedication”, *European Journal of Operational Research*, vol. 263, núm. 3, pp. 768–781, dic. 2017, doi: 10.1016/j.ejor.2017.05.045.
10. A. Solti, M. Raffel, G. Romagnoli, y J. Mendling, “Misplaced product detection using sensor data without planograms”, *Decision Support Systems*, vol. 112, pp. 76–87, ago. 2018, doi: 10.1016/j.dss.2018.06.006.
11. M. Keramatpour, S. T. A. Niaki, y S. H. R. Pasandideh, “A bi-objective two-level news vendor problem with discount policies and budget constraint”, *Computers & Industrial Engineering*, vol. 120, pp. 192– 205, jun. 2018, doi: 10.1016/j.cie.2018.04.040.

2.3 PROBLEM STATEMENT DEFINITION

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

A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

I am	Describe customer with 3-4 key characteristics - <i>who are they?</i>	Describe the customer and their attributes here
I'm trying to	List their outcome or "job" the care about - <i>what are they trying to achieve?</i>	List the thing they are trying to achieve here
but	Describe what problems or barriers stand in the way – <i>what bothers them most?</i>	Describe the problems or barriers that get in the way here
because	Enter the "root cause" of why the problem or barrier exists – <i>what needs to be solved?</i>	Describe the reason the problems or barriers exist
which makes me feel	Describe the emotions from the customer's point of view – <i>how does it impact them emotionally?</i>	Describe the emotions the result from experiencing the problems or barriers



EXAMPLE

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Retailer	Organize stock, manage purpose and sale detail	Lot of time and space consuming	I need to organize the stock, purchase detail, sale details and maintain the inventory model	Better relief and happy
PS-2	Retailer	Monitor stock levels & avoid out of stock situations	Is time consuming, redundant and vulnerable to errors	Using manual inventory tracking procedures across different software and spreadsheets	Frustrated
PS-3	Warehouse supervisor	Track items in real time as they are received, stored, picked, packed and shipped	Advises on inventory replenishment	Guides inventory receiving and put away	Tired & stressed

CHAPTER 3

IDEATION & PROPOSED SOLUTION

IDEATION & PROPOSED SOLUTION

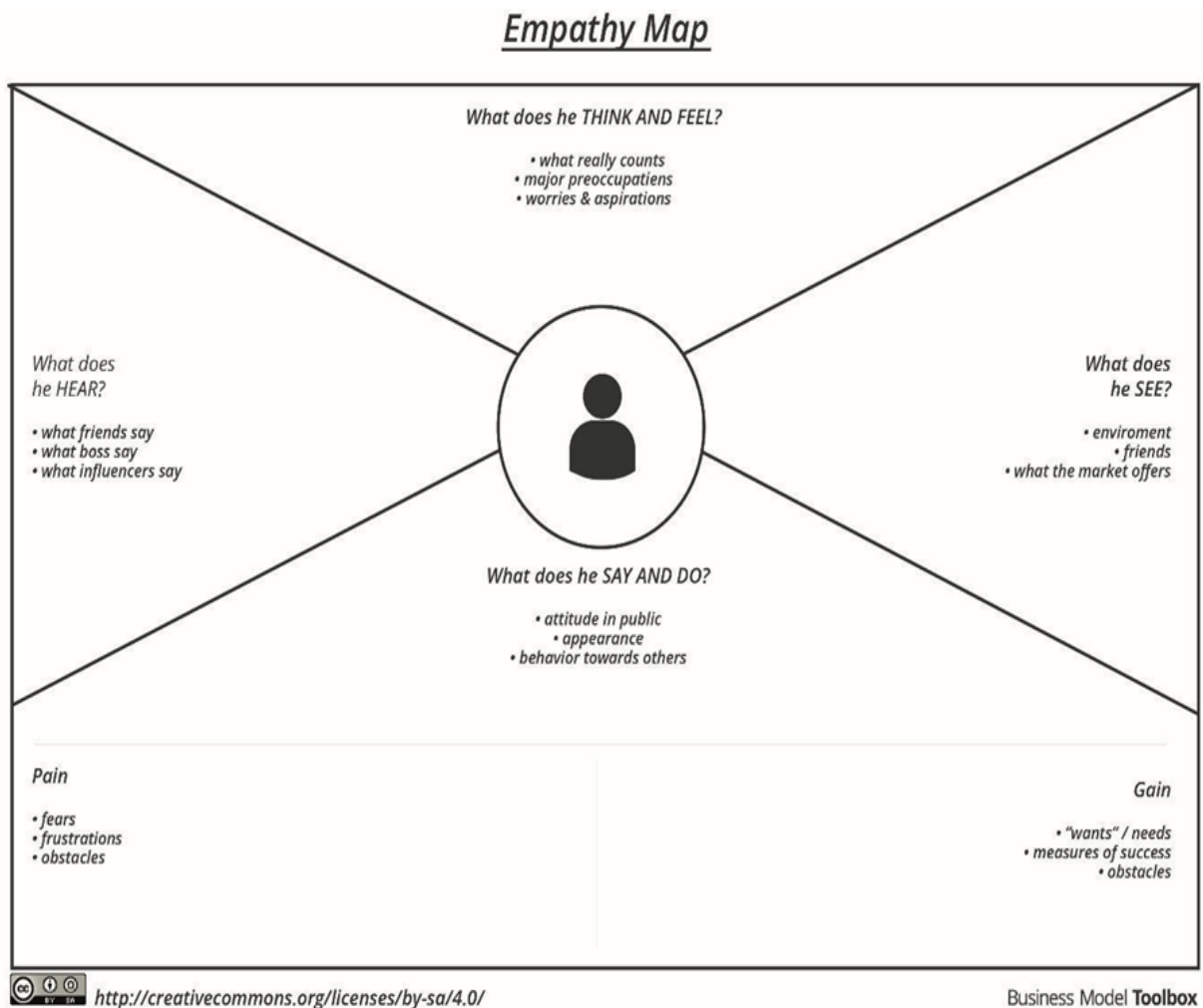
3.1 EMPATHY MAP CANVAS

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

It is a useful tool to help teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

EXAMPLE




3.2 IDEATION AND BRAINSTORMING

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

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

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



Brainstorm & idea prioritization

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

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

➔

Before you collaborate

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

🕒 10 minutes

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

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

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

[Open article](#) ➔

1


Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

🕒 5 minutes

PROBLEM

How might we [your problem statement]?



Key rules of brainstorming

To run a smooth and productive session

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

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

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

🕒 10 minutes

TP

You can select a sticky note and hit the pencil (switch to sketch) icon to start drawing!

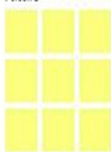
Amar



Yuktesh



Person 3



Person 4



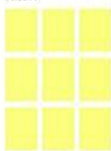
Person 5



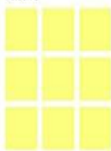
Person 6



Person 7



Person 8



3

Group Ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

Person 4

TP

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

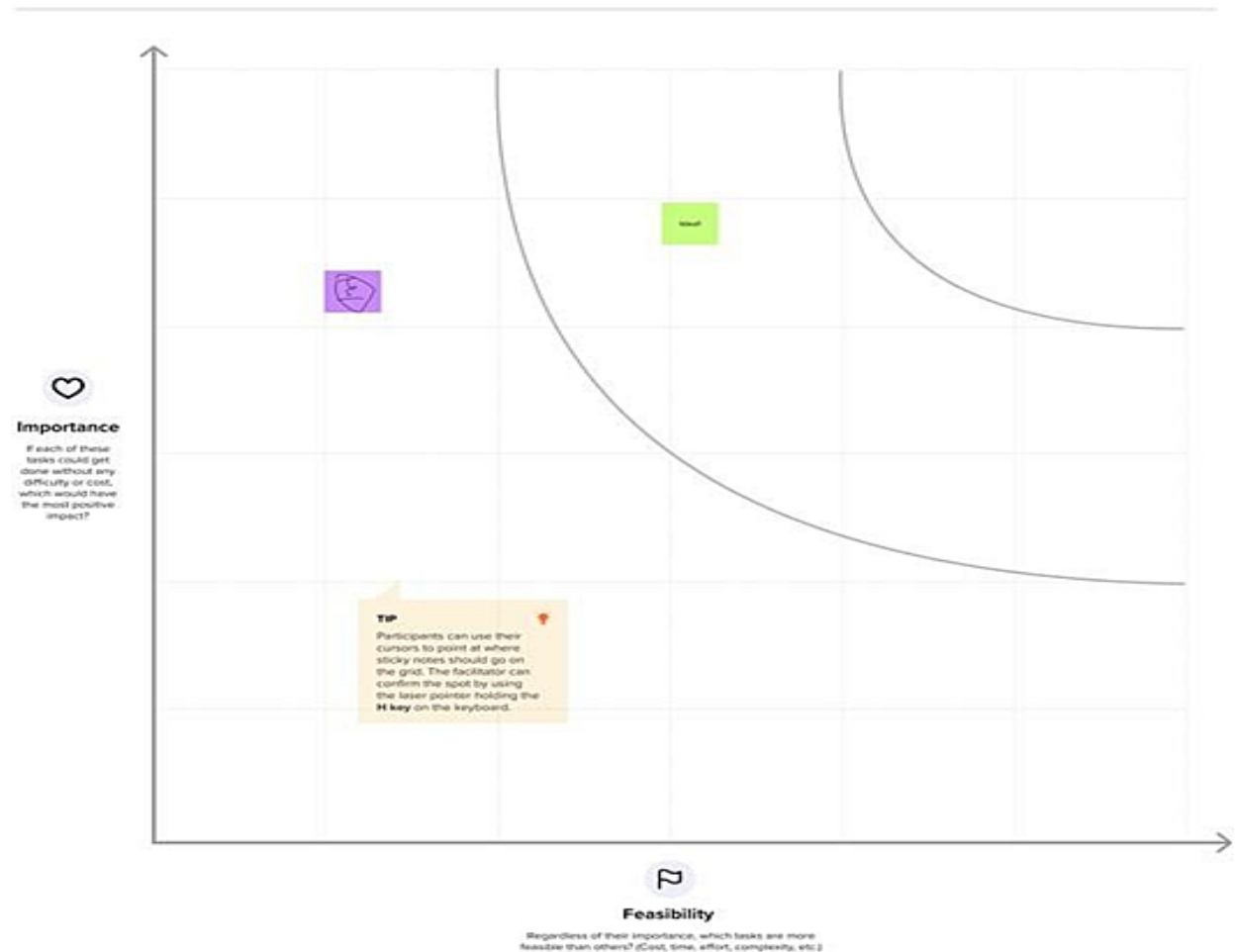
Step-3: Idea Prioritization

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

🕒 20 minutes






PROBLEM STATEMENT

The problem faced by the company is they do not have any systematic system to record and keep their inventory data. It is difficult for the admin to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.



Key rules of brainstorming

To run an smooth and productive session

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



BRAIN STORM

Benson R

Avoid shortages	avoid outdated items	Reduce the risk of loss
Warehouse management	Improve cash flows	Reduce wasted inventory
Save time	Asset life cycle	Reduce risk of human error

Aakash A

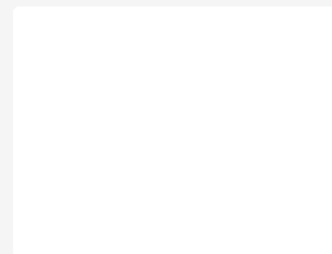
Tagging and barcoding	Forecasting of the inventory	Optimized inventory
Reporting the business activities	Real time inventory tracking	Performs stock counts
Stock projection	Multiple device support	Better customer experience

Arun Kumar G

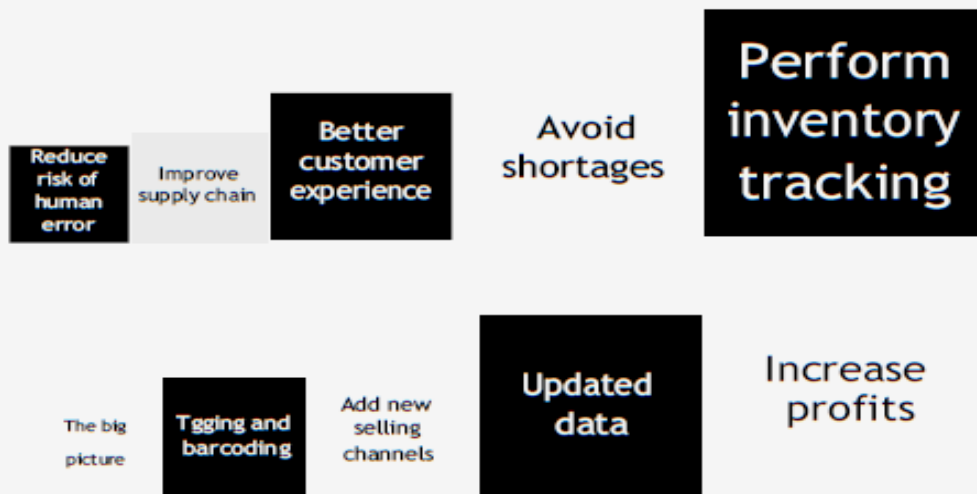
Perform inventory tracking	Prevent stock overstealing	Reducing inaccuracies
Cut cost and increase profits	Data security	Happy customers
No more manual work	Cost saving	Improve supply chain

Eashan Sai

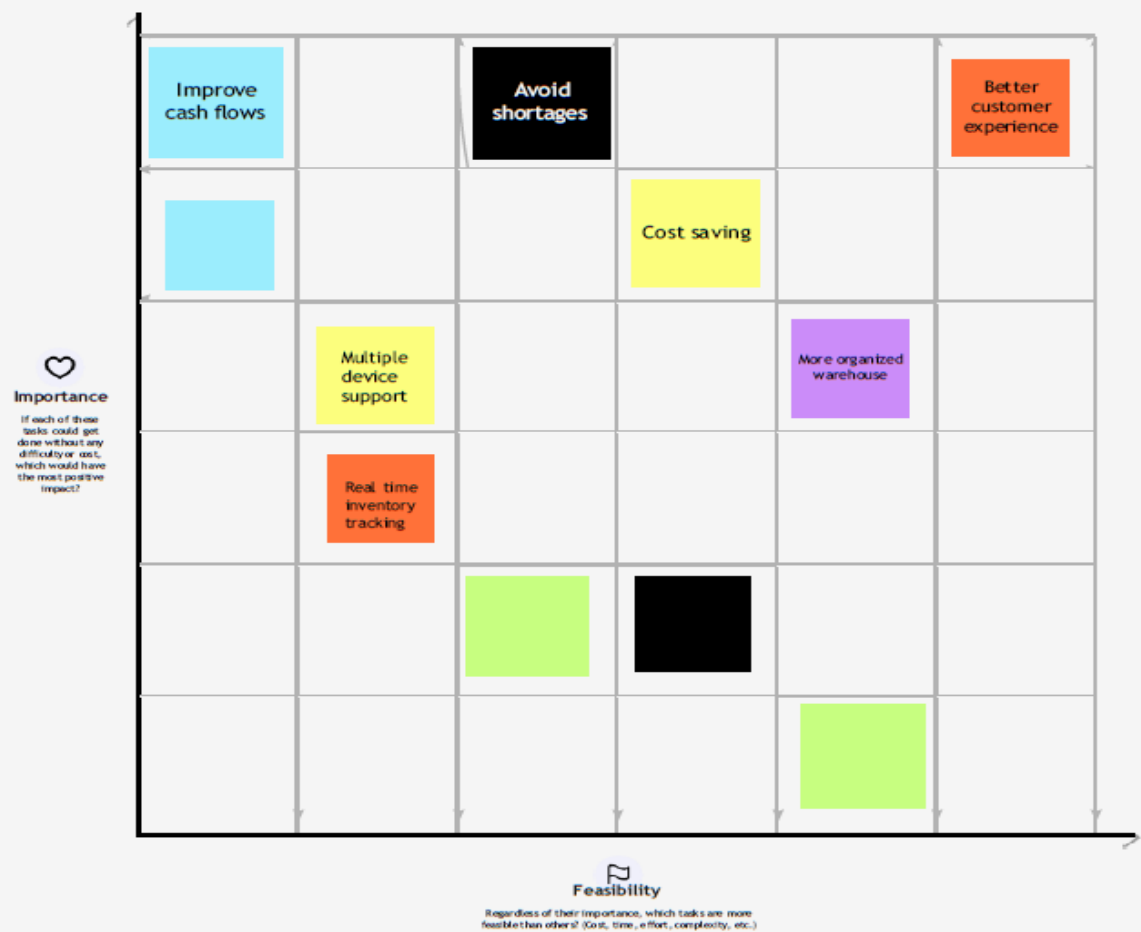
Increase profits	Greater insights	Improve business planning
Improve accuracy	Reduce redundant work	Find excess stock
Reduce risk of overstealing	Data and asset security	Add new selling channels



GROUP IDEA



IDEA PRIORITIZATION



3.3 PROPOSED SOLUTION

S.N	Parameter	Description
0.		
1.	Problem Statement (Problem to be solved)	Retailers do not have any systematic system to record and keep their inventory data.
2.	Idea / Solution description	To develop a cloud web application that will help retailers to manage, track, and control their stocks.
3.	Novelty / Uniqueness	Real time inventory tracking and secured user interactions.
4.	Social Impact/ Customer Satisfaction	Better interface to understand the tracking of stocks and better reliability over stock management.
5.	Business Model (Revenue Model)	This cloud web application will get higher usage and acceptance in market and among people of this generation.
6.	Scalability of the Solution	9 out of 10

3.4 PROBLEM SOLUTION FIT

Project Title: Inventory Management System For Retailers

Solution Fit Template

Team Id: PNT2022TMID00863

Define CS, fit into CC	<p>1. CUSTOMER SEGMENT(S) CS</p> <p>Inventory management aids businesses in determining which merchandise to order when and in what quantities. These are used by managers, accountants and production team to keep a track of demand and supply of products in company.</p>	<p>6. CUSTOMER CONSTRAINTS CC</p> <p>The customer might initially face constraints with application usage and might need stock knowledge</p>	<p>5. AVAILABLE SOLUTIONS AS</p> <p>Manual inventory tracking is possible, but it results in longer order processing, more labour expenses, and larger inventory write-offs at the end of the year</p>	Explore AS, differentiate
Focus on J&P, tap into BE, understand RC	<p>2. JOBS-TO-BE-DONE / PROBLEMS J&P</p> <p>The primary challenges of inventory management are having too much inventory and not being able to sell it, not having enough inventory to fulfill orders, and not understanding what items you have in inventory and where they're located.</p> <p>Other obstacles include:</p> <ul style="list-style-type: none"> - Getting Accurate Stock Details - Poor Processes - Changing Customer Demand - Using Warehouse Space Well 	<p>9. PROBLEM ROOT CAUSE RC</p> <ul style="list-style-type: none"> - Inaccurate information about stock movement - Demands of consumers change day by day 	<p>7. BEHAVIOUR BE</p> <ul style="list-style-type: none"> - Track and monitor the incoming and outgoing of stocks. - To update available quantity information onto cloud frequently. - Know the market trends and adapt accordingly to increase the profitability. - Manage the inventory efficiently to avoid out of stock 	Focus on J&P, tap into BE, understand RC
Identify strong TR & EM	<p>3. TRIGGERS TR</p> <ul style="list-style-type: none"> - Fluctuating customer demand - Market competition - Management of resources - Management of demand and supply 	<p>10. YOUR SOLUTION SL</p> <p>The issue arises due to the lack of communication between the supplier and the production managers. So, the aim is to develop a cloud application that provides an easy way to manage the sales and purchases and track inventory.</p>	<p>8. CHANNELS of BEHAVIOUR CH</p> <p>ONLINE</p> <ul style="list-style-type: none"> - Alerting the particular person about the stocks limits, either full or empty or even about the reach of a particular limit - Updating of flow of the stocks regularly <p>OFFLINE</p> <ul style="list-style-type: none"> - Manual Checking - Stock Distribution among the Inventory 	Identify strong TR & EM
	<p>4. EMOTIONS: BEFORE / AFTER EM</p> <p>Before - The user might feel confused and stressed about tracking their orders and might have struggles with the application UI.</p> <p>After - As they get acquainted with the software, it becomes easy and comfortable to use and track their order.</p>			

CHAPTER 4

REQUIREMENT ANALYSIS

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story/ Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIn
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Business regulations	Many needs may fit under this category
FR-4	Product management	Easily track product information Quickly produce reports for single or multiple sold products
FR-5	Audit Monitoring	The technique of tracking crucial data is known as audit tracking
FR-6	Historical Data	Specify the amount of storage you need to handle this expansion

4.2 NON-FUNCTIONAL REQUIREMENTS

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

FR No.	Non-FunctionalRequirement	Description
NFR-1	Usability	Backups for database areavailable
NFR-2	Security	The security requirements deal with the primary security. only authorized users can access the system with username and password of administrator
NFR-3	Reliability	The software will not be able to connect to the database in the event of the server being down due to a hardware or software failure
NFR-4	Performance	Easy tracking of records and updating can be done .
NFR-5	Availability	The software will be available only to administrator of the organization and the product as well as customer details will be recorded by him. He can add customers, Update and delete them as well as add new products and manage them
NFR-6	Scalability	The ability of a system to handle a growing amountof work

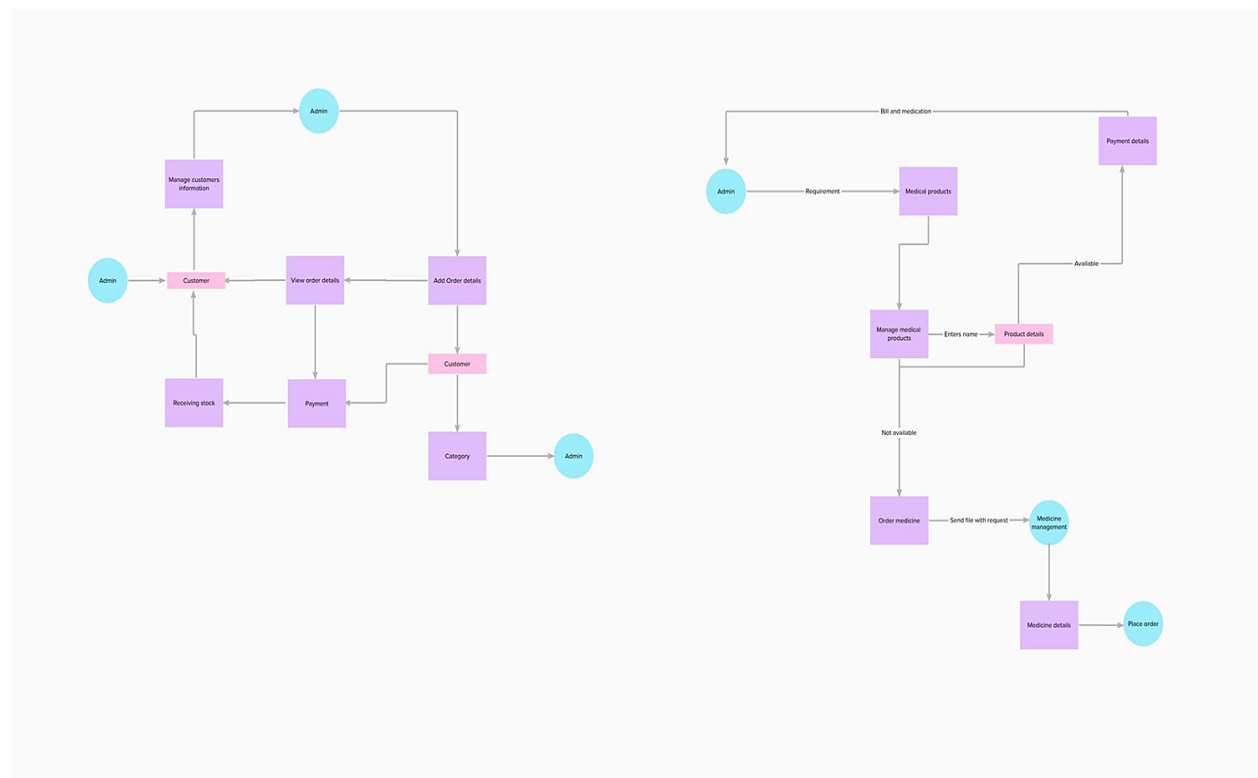
CHAPTER 5

PROJECT DESIGN

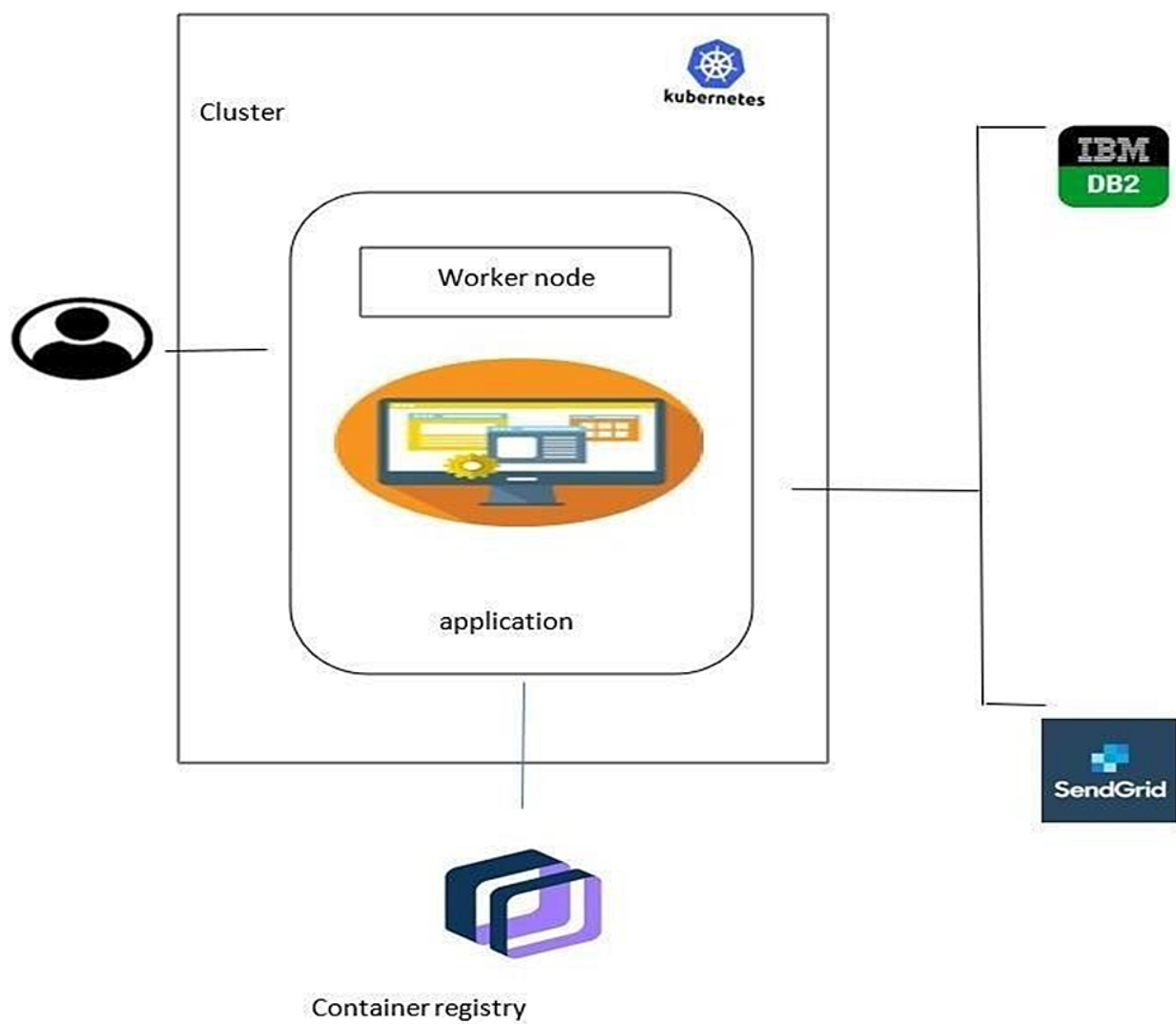
PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

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



5.2 SOLUTION & TECHNICAL ARCHITECTURE



5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register & application Through Gmail	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can access my account	High	Sprint-1
	Dashboard	USN-6	As a user, I can log into my account for the mobile	I can access my account /Dashboard	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-2
		USN-10	As a user, I can upload a Profile photo and add my name to my account	I can upload my Profile photo/Name in my account	Medium	Sprint-1
Customer Care Executive	Customer Support	USN-11	As a user, I can support for customers to handle queries and complaints from their customers	I can support for customers to clear complaints	High	Sprint-1
Administrator	Responsibility	USN-12	As a system administrator I want to be able to add new users when required so that	I Can add new users	High	Sprint-1

CHAPTER 6

PROJECT PLANNING & SCHEDULING

PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by using my email & password and confirming mylogin credentials.	3	High	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C
Sprint-1	Confirmation	USN-3	As a user, I can receive my confirmation emailonce I have registered for the application.	2	High	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C

Sprint-1	Login	USN-4	As a user, I can log in to the authorized accountby entering the registered emailand password.	3	Medium	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C
----------	-------	-------	--	---	--------	---

Sprint-2	Stocks update	USN-6	As a user, I can add products which are not available in the inventory and restock the products.	3	Medium	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C
Sprint-3	Sales prediction	USN-7	As a user, I can get access to sales prediction tool which can help me to predict better restock management of product.	6	Medium	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C
Sprint-4	Request for customer care	USN-8	As a user, I am able to request customer care to get in touch with the administrators and enquire the doubts and problems.	4	Medium	Benson Joseph R, Arunkumar G, Aakash A, Eashansai U.C

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Total StoryPoints	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned EndDate)	Sprint Release Date(Actual)
Sprint-1	11	6 Days	24 Oct 2022	29 Oct 2022	11	29 Oct 2022
Sprint-2	7	6 Days	31 Oct 2022	05 Nov 2022	7	05 Nov 2022
Sprint-3	6	6 Days	07 Nov 2022	12 Nov 2022	6	12 Nov 2022
Sprint-4	7	6 Days	14 Nov 2022	19 Nov 2022	7	19 Nov 2022

6.3 REPORTS FROM JIRA

SPRINTS:

Projects / IMSR

Backlog

Q

AA AG B E Epic

Insights

▼ IMSR Sprint 1 24 Oct – 29 Oct (4 issues)

0 0 0 Complete sprint

- IMSR-1 As a user, I can register for the application by using my email & password and confirming my login credentials. TO DO AA
- IMSR-2 As a user, I can login through my E-mail. TO DO AG
- IMSR-3 As a user, I can receive my confirmation email once I have registered for the application. TO DO B
- IMSR-4 As a user, I can log in to the authorized account by entering the registered email and password. TO DO E

+ Create issue

▼ IMSR Sprint 2 31 Oct – 5 Nov (2 issues)

0 0 0 Complete sprint

- IMSR-5 As a user, I can view the products that are available currently. TO DO AA
- IMSR-6 As a user, I can add products which are not available in the inventory and restock the products. TO DO B

+ Create issue

Projects / IMSR

Backlog

Q

AA AG B E Epic

Insights

+ Create issue

▼ IMSR Sprint 3 7 Nov – 12 Nov (1 issue)

0 0 0 Complete sprint

- IMSR-7 As a user, I can get access to sales prediction tool which can help me to predict better restock management of product. TO DO AA

+ Create issue

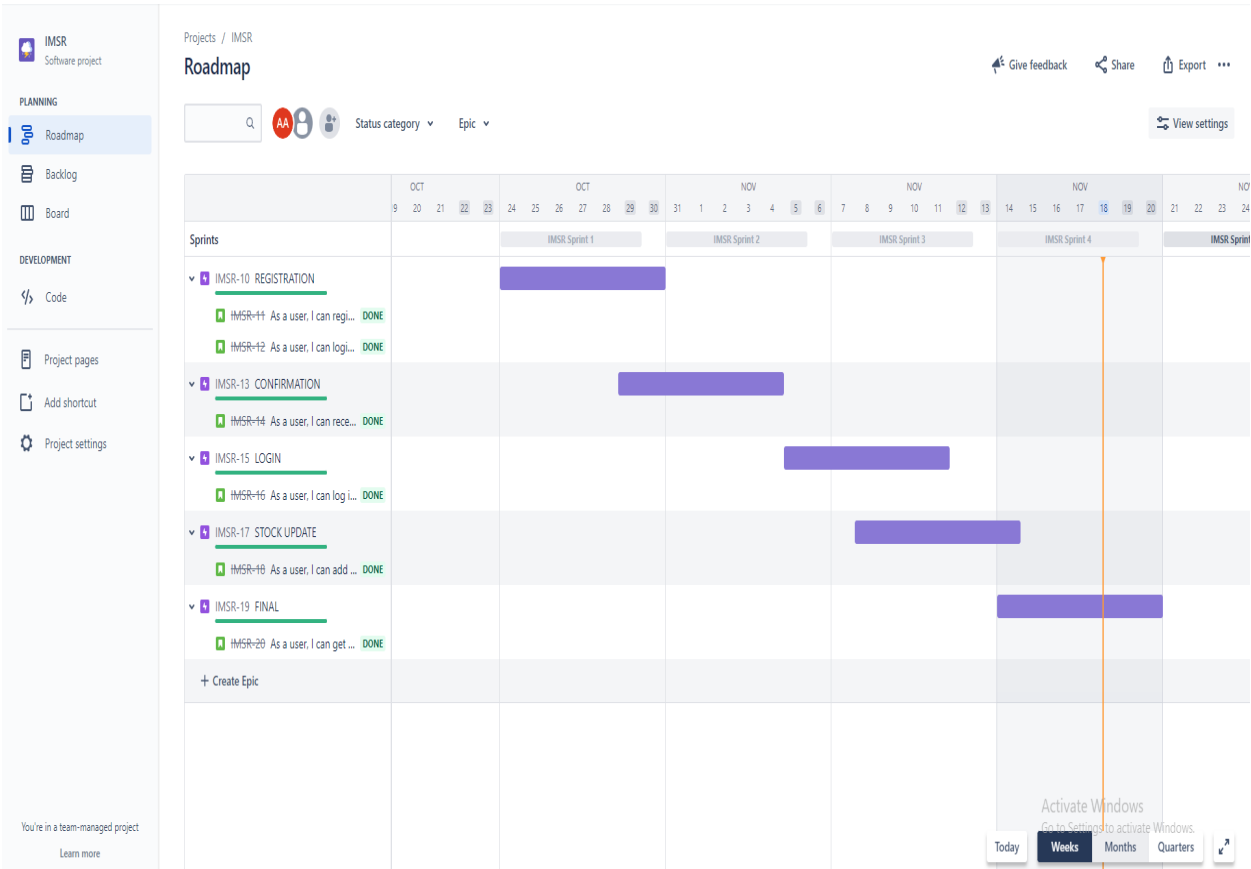
▼ IMSR Sprint 4 14 Nov – 19 Nov (2 issues)

0 0 0 Complete sprint

- IMSR-8 As a user, I am able to request customer care to get in touch with the administrators and enquire the doubts and problems. TO DO AG
- IMSR-9 As a user, I am able to send feedback forms reporting any ideas for improving or resolving any issues I am facing to get it resolved. TO DO E

+ Create issue

ROADMAP



CHAPTER 7

CODING & SOLUTIONING

CODING & SOLUTIONING

7.1 FEATURE 1

Front-End Development:

A front-end development architects and develops websites and applications using web technologies (i.e., HTML, CSS, DOM, and JavaScript), which run on the Open Web Platform or act as compilation input for non-web platform environments.

The main features that we added in our software are

- Product Movements
- Location.

PRODUCT MOVEMENT CODE:

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

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

```
<h1>Product Movements</h1>
```

```
<a class="btn btn-success" href="/add_product_movements">Add Product  
Movements</a>
```

```
<hr>
```

```
<table class="table table-striped">
```

```
<thead>
```

```
<tr>
```

```
<th>Movement ID</th>
```

```
<!--<th>Time</th-->
```

```
<th>From Location</th>
```

```
<th>To Location</th>
```

```
<th>Product ID</th>
```

```

        <th>Quantity</th>

    </tr>

</thead>

<tbody>

    {% for movement in movements %}

    <tr>

        <td>{{ movement.MOVEMENT_ID }}</td>

        <!--<td>{{ movement.TIME }}</td>-->

        <td>{{ movement.FROM_LOCATION }}</td>

        <td>{{ movement.TO_LOCATION }}</td>

        <td>{{ movement.PRODUCT_ID }}</td>

        <!--<td><a href="edit_product_movement/{{ movement.MOVEMENT_ID }}"
class="btn btn-primary pull-right">Edit</a></td>-->

        <td>

            <form action="{{ url_for('delete_product_movements',
id=movement.MOVEMENT_ID) }}" method="POST">

                <input type="hidden" name="method" value="DELETE">

                <input type="submit" value="Delete" class="btn btn-danger">

            </form>

        </td>

    </tr>

    {% endfor %}

</tbody>

</table>

```

```
{% endblock %}
```

ADD PRODUCT MOVEMENTS CODE:

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

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

```
<h1>Add Product Movements</h1>
```

```
{% from "includes/_formhelpers.html" import render_field %}
```

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

```
    <div class="form-group">
```

```
        {{ render_field(form.from_location, class_="form-control") }}
```

```
    </div>
```

```
    <div class="form-group">
```

```
        {{ render_field(form.to_location, class_="form-control") }}
```

```
    </div>
```

```
    <div class="form-group">
```

```
        {{ render_field(form.product_id, class_="form-control") }}
```

```
    </div>
```

```
    <div class="form-group">
```

```
        {{ render_field(form.qty, class_="form-control", type="number") }}
```

```
    </div>
```

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

```
</form>
```

```
{% endblock %}
```

7.2 FEATURE 2

Back-end development means **working on server-side software, which focuses on everything you can't see on a website**. Back-end developers ensure the website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers.

we used the sendgrid software to send the emails to the customers and service providers.

SendGrid:

SendGrid is a cloud-based SMTP provider that **allows you to send email without having to maintain email servers**. SendGrid manages all of the technical details, from scaling the infrastructure to ISP outreach and reputation monitoring to whitelist services and real time analytics.

SENDGRID CODE:

```
import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

from email.mime.base import MIMEBase


def alert(main_msg):

    mail_from = '19i361@psgtech.ac.in'

    mail_to = '19i303@psgtech.ac.in'

    msg = MIMEMultipart()

    msg['From'] = mail_from

    msg['To'] = mail_to

    msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
```



```
mail_body = main_msg

msg.attach(MIMEText(mail_body))


try:

    server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)

    server.ehlo()

    server.login('apikey', 'SEND GRID API KEY')

    server.sendmail(mail_from, mail_to, msg.as_string())

    server.close()

    print("Mail sent successfully!")

except:

    print("Some Issue, Mail not Sent :(")
```

CHAPTER 8

TESTING

8.1 TEST CASES

TEST CASE: 1

PRODUCT: INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

USECASE: SIGNUP AND LOGIN

TEST CASE ID	TESTCASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
1	Fill all the appropriate details in the displayed form to enrol as new user and click submit button	Registered successfully	As expected	PASS
2	Enter username and password in order to login	Home page is displayed after successful login	As expected	PASS

TEST CASE: 2

PRODUCT: INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

USECASE: ADD PRODUCTS AND LOCATIONS

TEST CASE ID	TESTCASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
1	Add the products in the product field	Added successfully	As expected	PASS
2	Add the locations in the location field	Added Successfully	As expected	PASS

TEST CASE: 23

PRODUCT: INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

USECASE: EXCHANGE AND MAINTAINING THE STOCKS

TEST CASE ID	TESTCASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
1	Maintaining the stocks	Working successfully	As expected	PASS
2	Exchanging the stocks	Working successfully	As expected	PASS

8.2 USER ACCEPTANCE TESTING

1. Final Stage, before handing over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
2. The system under consideration is tested for user acceptance and constantly keeping touch with the prospective system user at the time of developing and making changes whenever required.
3. It involves planning and execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.
4. Two set of acceptance test to be run:
 - i. Those developed by quality assurance group
 - ii. Those developed by customer

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS

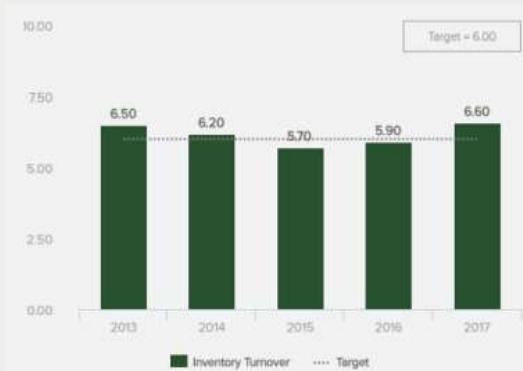
Supply chain management

Supply Chain Management

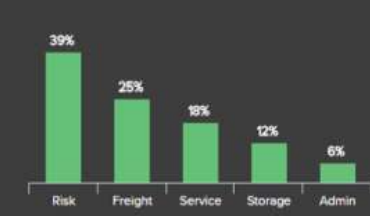
Inventory To Sales - This Year



Inventory Turnover - Last 5 Years



Carrying Cost Of Inventory



Inventory Accuracy

Congrats!
You hit the
target of
91.4%.

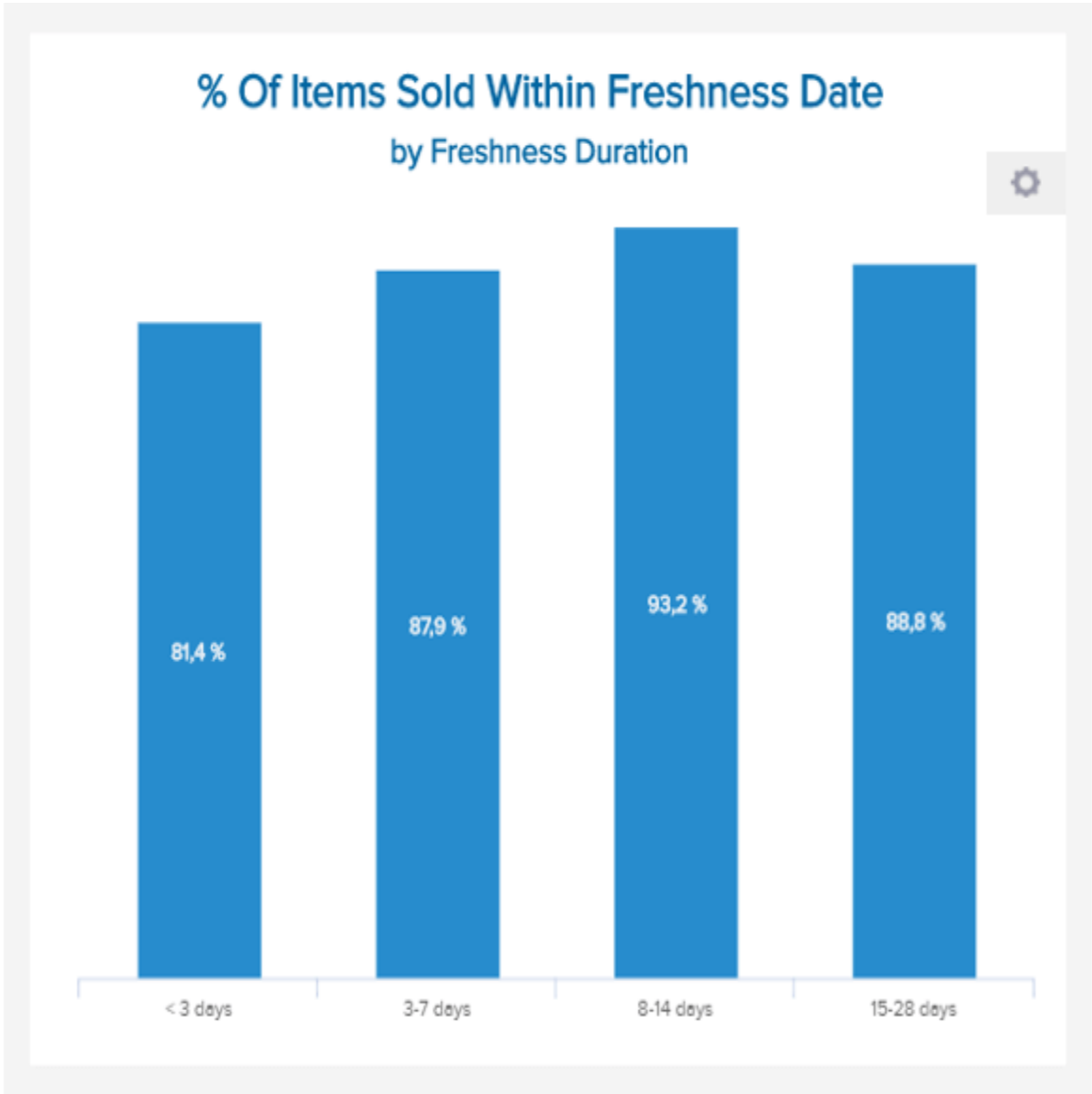


% Out Of Stock Items

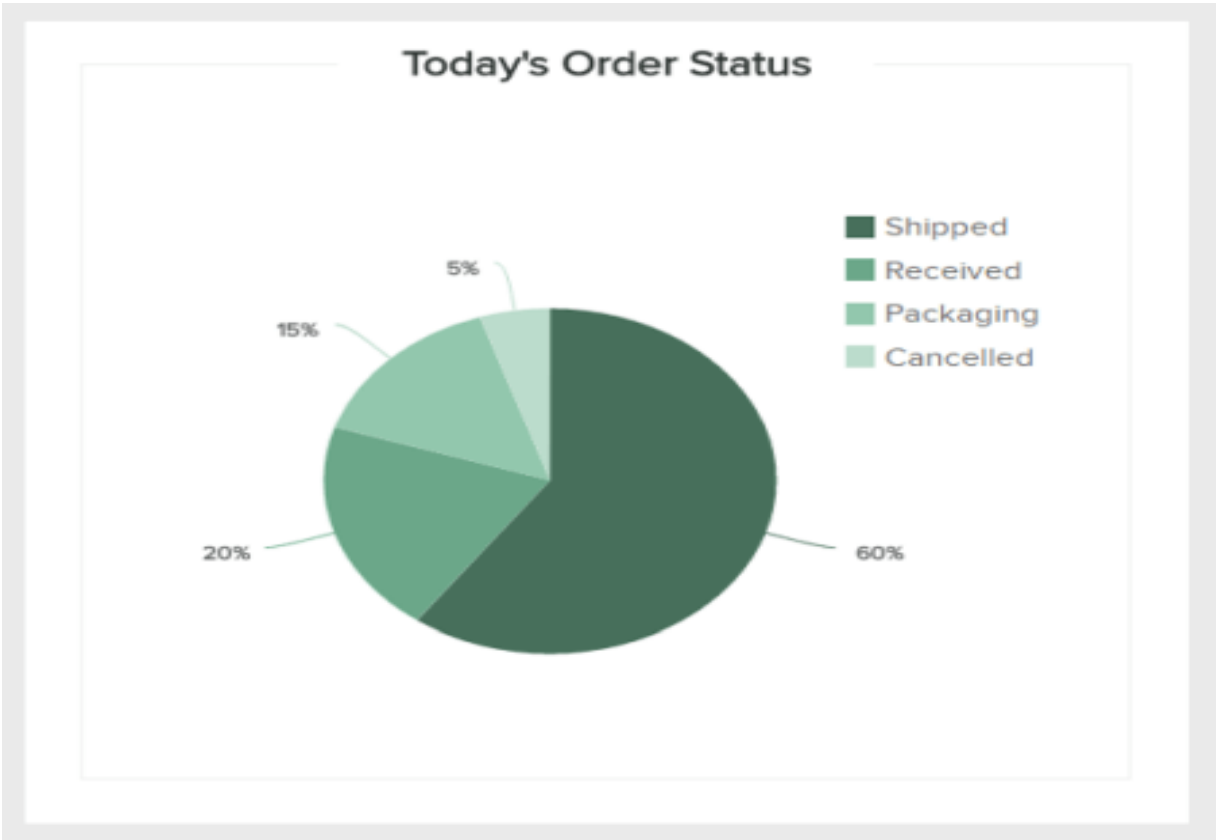
0.94%
This Year



Percentage of sold products within freshness date



Order status inventory metric



CHAPTER 10

ADVANTAGES & DISADVANTAGES

ADVANTAGES & DISADVANTAGES

ADVANTAGES

Automated Reordering and In-Stock Information

Computerized inventory informs employees and customers within seconds whether an item is in stock. Because the inventory is synced with sales, there is a **running tally of what is in stock and what isn't**. This helps flag reordering needs and provides better service to customers. As inventory drops below a specific threshold, new orders are placed with vendors and tracked to let customers know when the new products will arrive.

Integration With Accounting

Many of the computerized inventory platforms **integrate with accounting software to track cash flow**. This makes the process of transferring inventory costs and assets between programs seamless and reduces the need for additional bookkeeping costs. Financial statements are more easily generated with shared data between inventory and bookkeeping.

Forecasting and Planning

Inventory management software does more than track where inventory is located and when to reorder it. A data collection system is used to create needed **forecasting and strategic planning reports**. Business owners review trends regarding which products do well in certain months or during specific cyclical seasons. Business owners use this data to plan for growth and order inventory intelligently to best utilize cash flow resources.

DISADVANTAGES

System Crash

One of the biggest problems with any computerized system is the potential for a system crash. A corrupt hard drive, power outages and other technical issues can result in the loss of needed data. At the least, businesses are interrupted when they are unable to access data they need. Business owners should back up data regularly to protect against data loss.

Malicious Hacks

Hackers look for any way to get company or consumer information. An inventory system connected to point-of-sale devices and accounting is a valuable resource to hack into in search of potential financial information or personal details of owners, vendors or clients. Updating firewalls and anti-virus software can mitigate this potential issue.

Reduced Physical Audits

When everything is automated, it is easy to forego time-consuming physical inventory audits. They may no longer seem necessary when the computers are doing their work. However, it is important to continue to do regular audits to identify loss such as spoilage or breakage. Audits also help business owners identify potential internal theft and manipulation of the computerized inventory system.

CHAPTER 11

CONCLUSION

CONCLUSION

Inventory management is a process of keeping in-depth track of all your products. With proper inventory management, you will ensure that your company correctly orders, handles, and tracks your entire stock. On the most fundamental level, inventory control is aimed to make sure that your shelves or virtual platforms are adequately stocked and you know your current inventory levels at any moment.

Inventory management is a very complex but essential part of the supply chain. An effective inventory management system helps to reduce stock-related costs such as warehousing, carrying, and ordering costs. As you have read above, there are different techniques that businesses can utilize to simplify and optimize stock management processes and control systems.

CHAPTER 12

FUTURE SCOPE

FUTURE SCOPE

The future scope of inventory management systems are

1. The Fourth Industrial Revolution will continue to drive technological change that will impact the way that we manage inventories.
2. Successful companies will view inventory as a strategic asset, rather than an aggravating expense or an evil to be tolerated.
3. Collaboration with supply chain partners, coupled with a holistic approach to supply chain management, will be key to effective inventory management.
4. The nature of globalization will change, impacting inventory deployment decisions dramatically.
5. Increased focus on supply chain security, and concerns about the quality of inventory itself, will be primary motivators to changing supply chain and inventory strategy.

CHAPTER 13

APENDIX

APPENDIX

SOURCE CODE

app.py

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField,
IntegerField
import ibm_db
from 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=ba99a9e6-d59e-4883-8fc0-
d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=31321;SECURITY=SS
L;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=cmm18332;PWD=qJ12LhPS1fuLW8
GS;",",")
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=54a2f15b-5c0f-46df-8954-
7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32733;SECURITY=SS
L;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=tbj18827;PWD=nL6zFtk4HV3qyDx7;
",",")

#TIME

#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)
```

GITHUB & PROJECT DEMO LINK

[**https://github.com/IBM-EPBL/IBM-Project-3418-1658560496**](https://github.com/IBM-EPBL/IBM-Project-3418-1658560496)