Date	19 November 2022
Team ID	PNT2022TMID21703
Project Name	Project - Inventory Management System For Retailers

Project Report

1. INTRODUCTION

1.1 Project Overview:

The main problem every inventory stock manager face is to keep a track of how much stock is purchased and how much stock is spent out. The stock admin does not have an easy way to manage and see stocks that is purchased and is dispatched out in the organization. At present this is done, where manager has to manually keep the reports in hand and calculate the costs eventually. The organization admin also does not have any easy way to look at the details and the amount of stock that is being purchased and dispatched. The cost that is spent on the food items is also not known exactly to the college admins. Going to do our project using flask,python,docker,ibm cloud.

1.2 Purpose:

Inventory Management systems are limited and fixed to a selected range of items and cannot be modified and extended based on the customer's needs. The Inventory Management System focuses on making it expandable and usable easily by the end user and with constant customer support to alter the user interface as needed, separate login credentials are provided for the manager and the organization to insert and manage the stock and view the same respectively. Unlike, other software's that provide similar kind of functionalities, Inventory Management System focuses on making it easier by adding additional functionalities like vendor details and allowing to take attendance and keep a count of number of members belonging to that organization.

2. LITERATURE SURVEY

2.1 Existing problem:

> Title: Inventory Management of Perishable Goods with Overconfident Retailers
Authors: Mingyang Zhang ,Xufeng Yang ,Taichiu Edwin Cheng, Chen Chang
Year: 2022

> Title: Supply Chain Contract Preferences Under Two Uncertain Selling Seasons Considering Intertemporal Inventory Capability.
Author: Jianhu Cai, Yujie Zhang, Qing Zhou, Tingfing Xue, Zhijun Zeng

2.2 References:

> https://www.mdpi.com/2227-7390/10/10/1716/pdf-vor
> https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9343878

2.3 Problem Statement Definition :

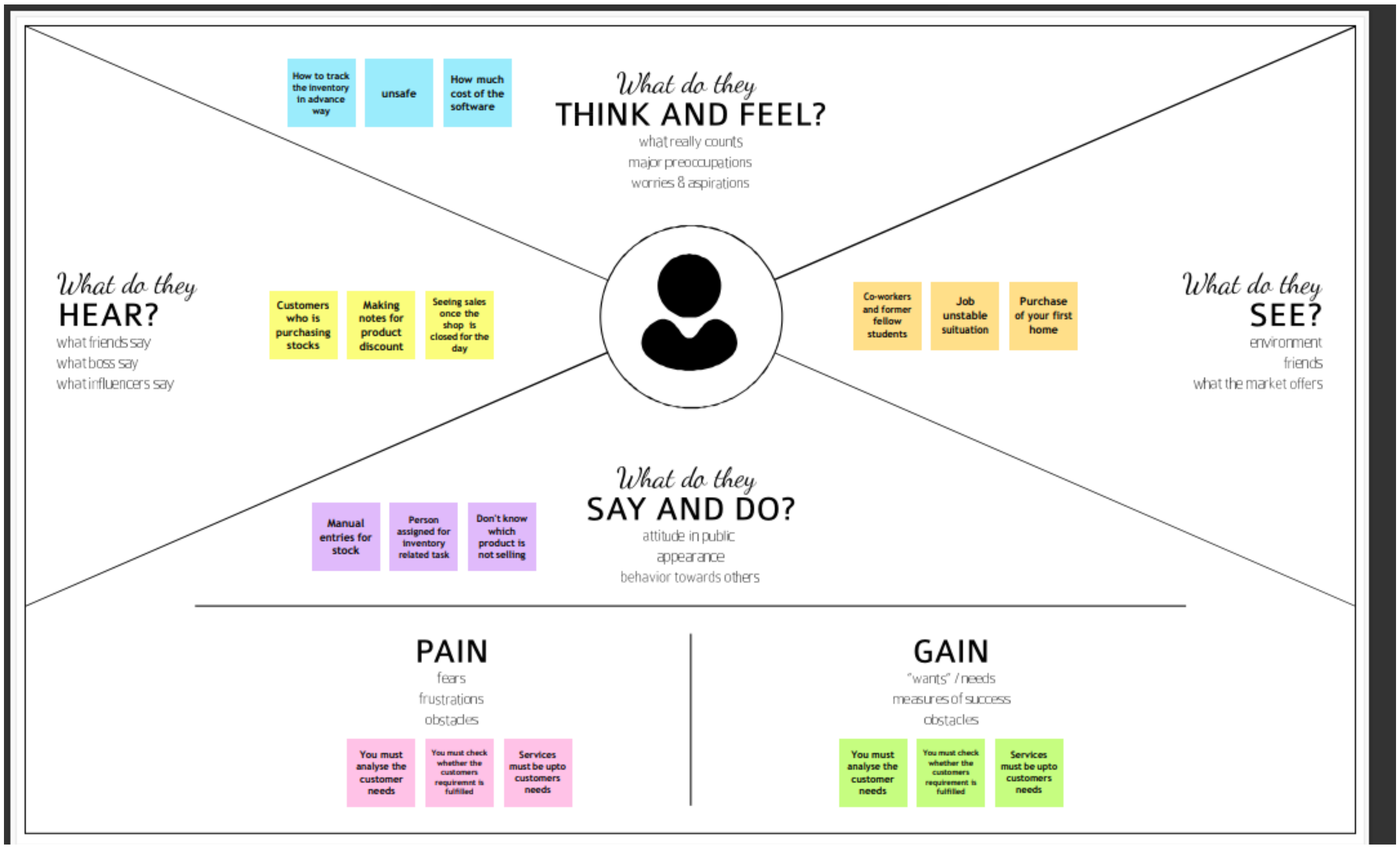
> In recent years, many retailers sell their products through not only offline but also online plaVorms. The sales of perishable goods on e-commerce plaVorms recorded phenomenal growth in 2020. However, some retailers are overconfident and order more products than the opfimal ordering quanfity, resulfing in great losses due to product decay. In this paper, we apply the newsvendor model to analyze the impacts of overconfident behavior on the retailer's opfimal pricing and order quanfity decisions and profit. Our model provides the overconfident retailer with a feasible and effecfive method to adjust opfimal ordering and pricing decisions. Through numerical studies, we examine the retailer's opfimal decisions under the scenarios of complete rafionality, over-esfimafion, and over-precision. We find that the over-esfimafion retailer always orders more products than the opfimal order quanfity, and the over- precision retailer always orders fewer products than the opfimal order quanfity. Under some

condifions, overconfidence hurts the retailer’s revenue to a large extent. Therefore, it is beneficial for the overconfident retailer to adjust its order quanfity according to our research findings.

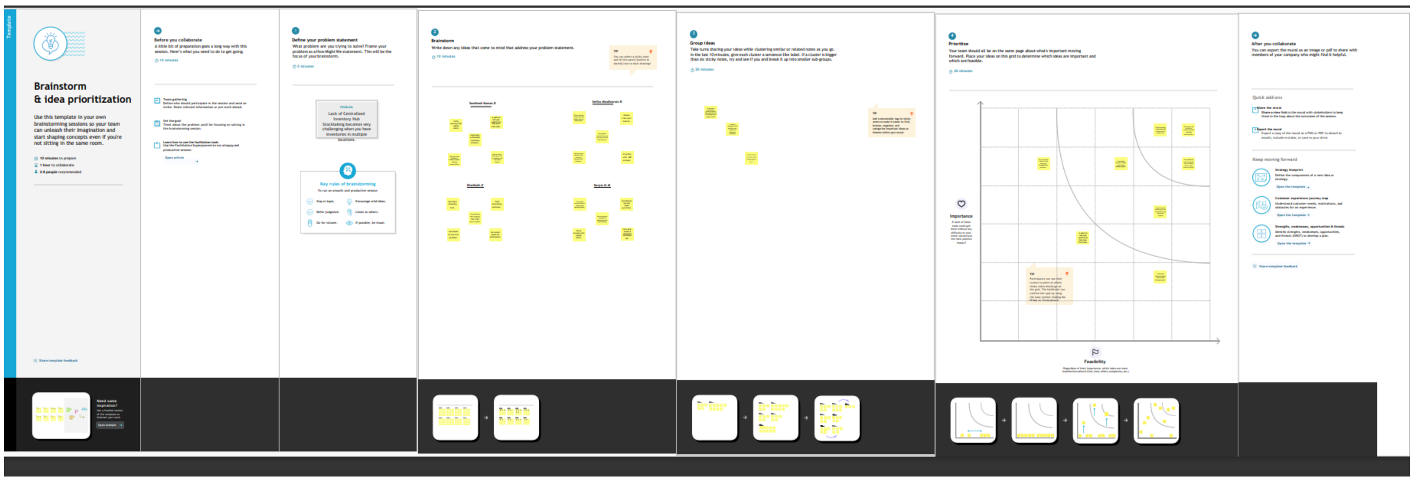
> We propose an intertemporal inventory decision model under demand uncertainty and retail compefifion. Facing the fact that different firms may have different intertemporal inventory capabilifies, the paper introduces two retailers and assumes only one retailer has the ability to carry intertemporal inventories. Two different contracts, i.e., the dynamic wholesale-price contract and the commitment wholesale-price contract are proposed. The opfimal decisions are derived under each contract, and all members' preferences regarding the two contracts are invesfigated. The results show that there exist some situafions in which the supply chain members prefer the same contract, and they can reach an agreement on the contract selecfion. We also find that the intertemporal inventory discriminately affects the supply chain members' expected profits under two different contracts. The retailer's intertemporal inventory capability may be beneficial for all members under the dynamic wholesale-price contract, but it always damages the supplier's expected profit under the commitment wholesale-price contract

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas :


How to track the inventory in advance way
unsafe
How much cost of the software
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
What do they HEAR?
what friends say
what boss say
what influencers say
Customers who is purchasing stocks
Making notes for product discount
Seeing sales once the shop is closed for the day
Co-workers and former fellow students
Job unstable suituation
Purchase of your first home
What do they SEE?
environment
friends
what the market offers
Manual entries for stock
Person assigned for inventory related task
Don't know which product is not selling
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
PAIN
fears
frustrations
obstacles
You must analyse the customer needs
You must check whether the customers requiremnt is fulfilled
Services must be upto customers needs
GAIN
"wants" / needs
measures of success
obstacles
You must analyse the customer needs
You must check whether the customers requirement is fulfilled
Services must be upto customers needs

3.2 Ideation & Brainstorming :



Template

Brainstorm & idea prioritization

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

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

Share template feedback

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

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

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

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

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
Lack of Centralised Inventory Hub
Stocktaking becomes very challenging when you have inventories in multiple locations.

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.

2 Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Santhosh Kumar.D
Sethu Madhavan.K
Shailesh.E
Surya.G.M

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

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

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

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

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

TIP
Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the H key on the keyboard.

After you collaborate
You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

Export the mural
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template

Share template feedback

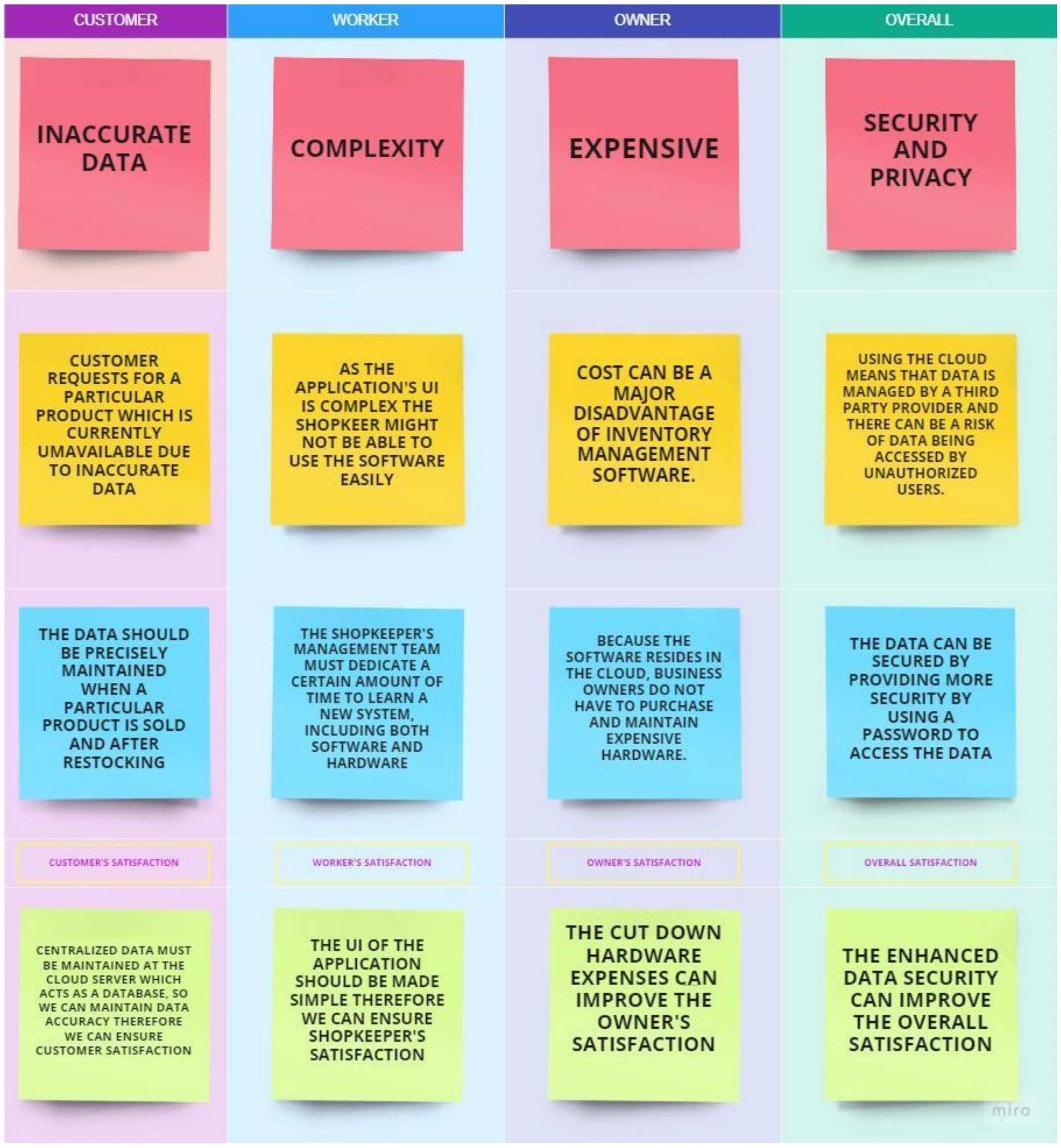
3.3 Proposed Solution :

S.No.	**Parameter**	**Description**
❖	Problem Statement (Problem to be solved)	• The retailers generally facing issues in recording the stocks and its threshold limit available. • The customers are not satisfied with the retailers store since it doesn't have enough supplements and the deliveries were not made on time.
❖	Idea / Solution description	• This proposed system will have a daily update system whenever a product is sold or it is renewed more. • The product availability is tracked daily and an alert system in again kept on to indicate those products which falls below the threshold limit. • All the customers can register their accounts after which they will be given a login credentials which they can use whenever they feel like buying the stocks. • The application allows the customers to know all the present time available stocks and also when the new stock will be available on the store for them to buy.

❖	Novelty / Uniqueness	• Certain machine learning algorithms are used to predict the seasonal high selling products which can be made available during that time. • Prediction of the best selling brand of all certain products based on their popularity, price and customer trust and satisfaction will be implemented. • Notifications will be sent to the retailers if any product that the customers have been looking for is not available so that the product can be stocked up soon.
❖	Social Impact / Customer Satisfaction	• The customers will be highly satisfied since the wasting of time while searching for an unavailable product is reduced. • The work load of the retailers will be minimized if the system is automated every day and during every purchase. • The customer satisfaction will be improved for getting appropriate response from the retailers and that too immediately.
❖	Business Model (Revenue Model)	• Hereby we can provide a robust and most reliable inventory management system by using: 1. ML algorithms for all the prediction purposes using all the past dataset since datasets are undoubtedly available in huge amounts. 2. Can deploy the most appropriate business advertising models. 3. To establish a loss preventing strategy. 4. And to ensure the all time, any where availability of products system.

❖	Scalability of the Solution	• Implementation of anyone and anywhere using system can be helpful for even a commoner to buy the products. • Daily and Each time purchase updation of the stock for preventing inventory shrinkage.

3.4 Problem Solution fit:



CUSTOMER	WORKER	OWNER	OVERALL
INACCURATE DATA	COMPLEXITY	EXPENSIVE	SECURITY AND PRIVACY
CUSTOMER REQUESTS FOR A PARTICULAR PRODUCT WHICH IS CURRENTLY UMAVAILABLE DUE TO INACCURATE DATA	AS THE APPLICATION'S UI IS COMPLEX THE SHOPKEER MIGHT NOT BE ABLE TO USE THE SOFTWARE EASILY	COST CAN BE A MAJOR DISADVANTAGE OF INVENTORY MANAGEMENT SOFTWARE.	USING THE CLOUD MEANS THAT DATA IS MANAGED BY A THIRD PARTY PROVIDER AND THERE CAN BE A RISK OF DATA BEING ACCESSED BY UNAUTHORIZED USERS.
THE DATA SHOULD BE PRECISELY MAINTAINED WHEN A PARTICULAR PRODUCT IS SOLD AND AFTER RESTOCKING	THE SHOPKEEPER'S MANAGEMENT TEAM MUST DEDICATE A CERTAIN AMOUNT OF TIME TO LEARN A NEW SYSTEM, INCLUDING BOTH SOFTWARE AND HARDWARE	BECAUSE THE SOFTWARE RESIDES IN THE CLOUD, BUSINESS OWNERS DO NOT HAVE TO PURCHASE AND MAINTAIN EXPENSIVE HARDWARE.	THE DATA CAN BE SECURED BY PROVIDING MORE SECURITY BY USING A PASSWORD TO ACCESS THE DATA
CUSTOMER'S SATISFACTION	WORKER'S SATISFACTION	OWNER'S SATISFACTION	OVERALL SATISFACTION
CENTRALIZED DATA MUST BE MAINTAINED AT THE CLOUD SERVER WHICH ACTS AS A DATABASE, SO WE CAN MAINTAIN DATA ACCURACY THEREFORE WE CAN ENSURE CUSTOMER SATISFACTION	THE UI OF THE APPLICATION SHOULD BE MADE SIMPLE THEREFORE WE CAN ENSURE SHOPKEEPER'S SATISFACTION	THE CUT DOWN HARDWARE EXPENSES CAN IMPROVE THE OWNER'S SATISFACTION	THE ENHANCED DATA SECURITY CAN IMPROVE THE OVERALL SATISFACTION

4. REQUIREMENT ANALYSIS

4.1 Functional requirement :

FR. No.	**Functional Requirement (Epic)**	**Sub Requirement (Story/Sub-Task)**
FR-1	User Registration	Registration through registration form. Registration through One-Tap Google Sign-in.
FR-2	User Authentication andConfirmation	Authentication via Google Authentication. Confirmation via Email. Confirmation via OTP.
FR-3	Product management	Quickly produce reports for single ormultiple products. Track information of dead and fast-movingproducts. Track information of suppliers andmanufacturers of the product.
FR-4	Audit Monitoring	The technique of tracking crucial data isknown as audit tracking. Monitor the financial expenses carried outthroughout the whole time (from receivingorder of the product to delivery of the product).
FR-5	Historical Data	Data of everything should be stored foranalytics and forecasting.

FR – 6	CRM (Customer Relationship Management)	Track the customer experience via ratingsgiven by them. Get customer reviews regularly or atleast atthe time of product delivery to work on customer satisfaction. User-friendly GUI to increase the customerbase from only techies to normal people.
FR - 7	Security Policy	User data collected must be as secure aspossible. User data must not be misused. They can only be used for user preferred advertisingpurposes.

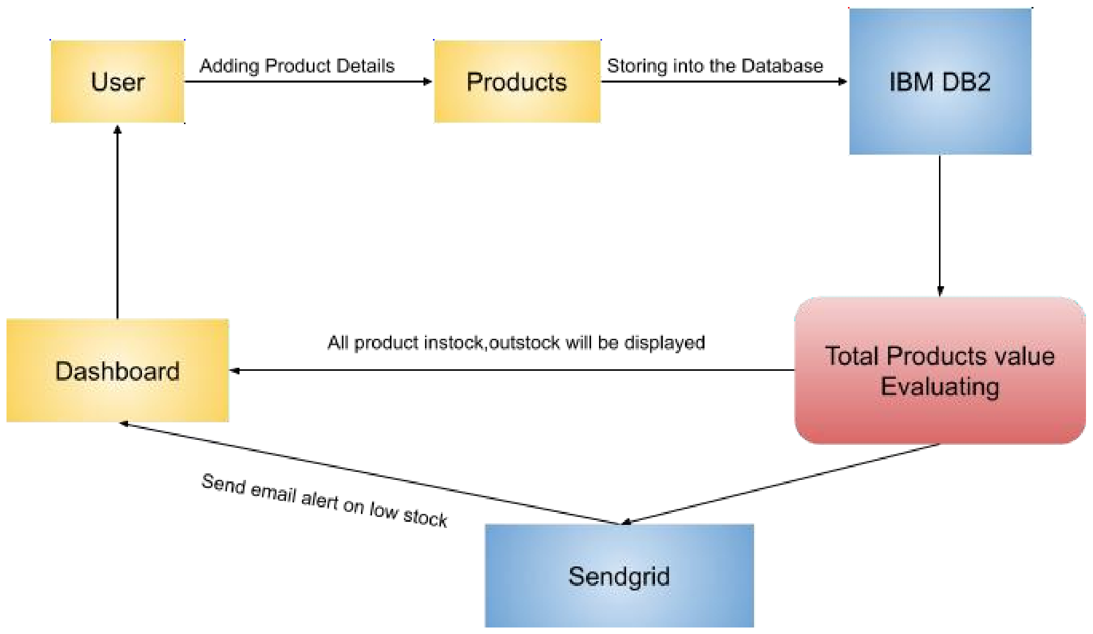
4.2 Non-Functional requirements:

FR No.	**Non-Functional Requirement**	**Des crip tion**
NFR-1	Usability	The UI should be accessible to everybodydespite of there diversity in languages. People with some impairments should also be able to use the application with ease. (Example,integrate google assistant so that blind people can use it). .
NFR-2	Security	The security requirements deal with the primary security. Only authorized users can access the systemwith their credentials. Administrator or the concerned security team shouldbe alerted on any unauthorized access or data breaches so as to rectify it immediately.

NFR-3	Reliability	The software should be able to connect to the database in the event of the server being down due toa hardware or software failure. The users must me intimated by the periodic maintenance break of the server so that they willbe aware of it.
NFR-4	Performance	Performance of the app should be reliable withhigh-end servers on which the software is running.
NFR-5	Availability	The software should be available to the users 24/7with all functionalities working. New module deployment should not impact theavailability of existing modules and their functionalities.
NFR-6	Scalability	The whole software deployed must be easily scalableas the customer base increases.

5. PROJECT DESIGN

5.1 Data Flow Diagrams :


User
Adding Product Details
Products
Storing into the Database
IBM DB2
Dashboard
All product instock,outstock will be displayed
Total Products value Evaluating
Send email alert on low stock
Sendgrid

5.2 Solution & Technical Architecture :

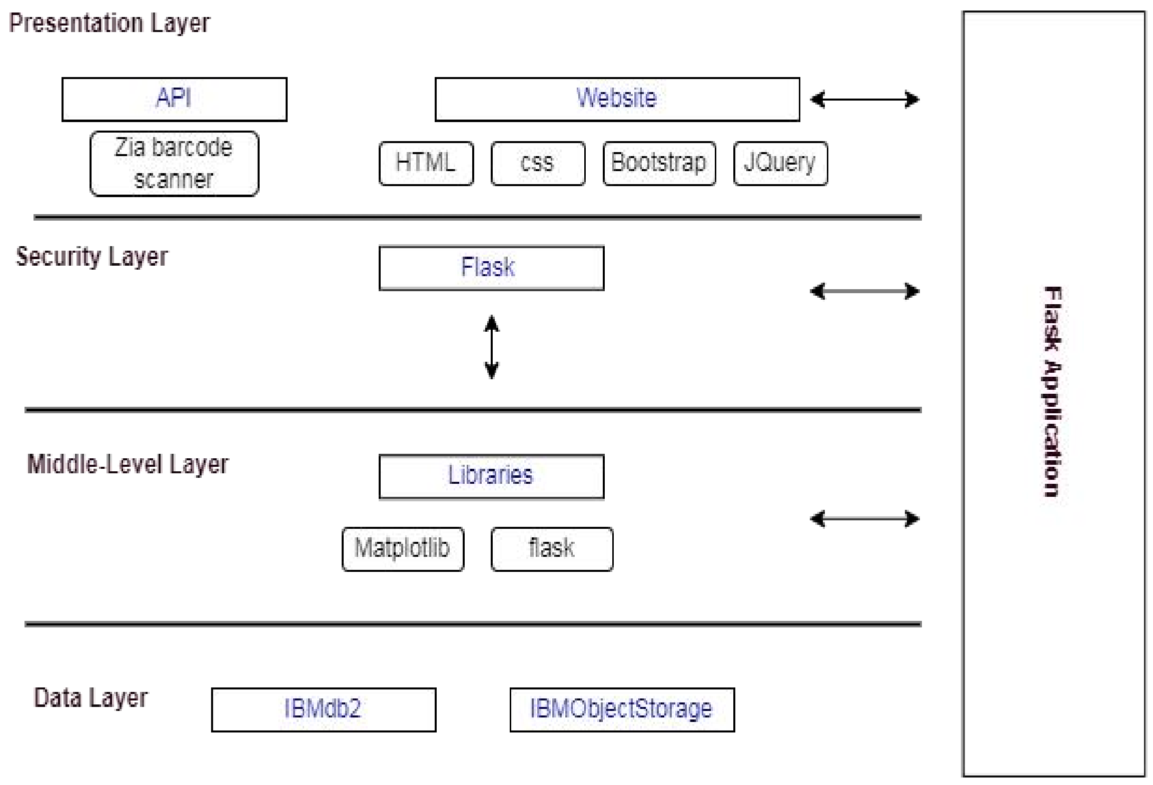


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1	User Interface	Web UI with Chatbot	HTML, CSS, Bootstrap, Jquery
2	Calculating Products Count	By entering barcode details intothe application	Zia Barcode Scanner
3	Showing high demandproduct	By the products data in IBMdb2	Data Visualization using Python Bar plot by MatplotLibrary
4.	Alert and Notification	Alerting the retailers regardingthe low stock count of the product	SendGrid
5	Chat	Chat with watson assistant	IBM Watson Assistant
6	Cloud Database	Database Service on Cloud	IBM DB2
7	File Storage	File storage requirements	IBM Object Storage
8	External API-1 Barcode	To Scan the product barcode	Zia Barcode Scanner

9	Infrastructure (Server /Cloud)	Cloud Server Configuration	Cloud Foundry, Kubernetes

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Styling our page,Python flaskmicroframework	Python Flask, Bootstrap
2.	Security Implementations	For securing our cloud data	SSL Certificates
3.	Scalable Architecture	Three – tier architecture (MVC)	Web server - HTML, CSS,Javascript Application server - Python Flask,Docker, Container Registry Database server - IBM DB2
4.	Availability	availability of application	IBM Load Balancer
5.	Performance	5 requests per seconds, Use of Local Machine CacheMemory	IBM Cloud, CDN

5.3 User Stories :

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Retailer(Webuser)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I will be redirected to loginpage	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 verify the OTP number	Medium	Sprint-1

	Login	USN-5	As a user, I can log into the application by entering email & password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-6	As a user,I can update stock in & out count details	Updation can be made through barcode scanning	High	Sprint -2
	Dashboard	USN-7	As a user,I can check the low stock details through alert message	Alert message can be received by registered mail	High	Sprint -1
		USN-8	As a user,I can check the total product details	I can view the value of total products in the stock	Medium	Sprint -2
		USN-9	As a user,I can check the high demandproduct details	I can update sales details of the products	High	Sprint -2
		USN-10	As a user,I can generate the invoice details	I can add incoming stockdetails	High	Sprint -1

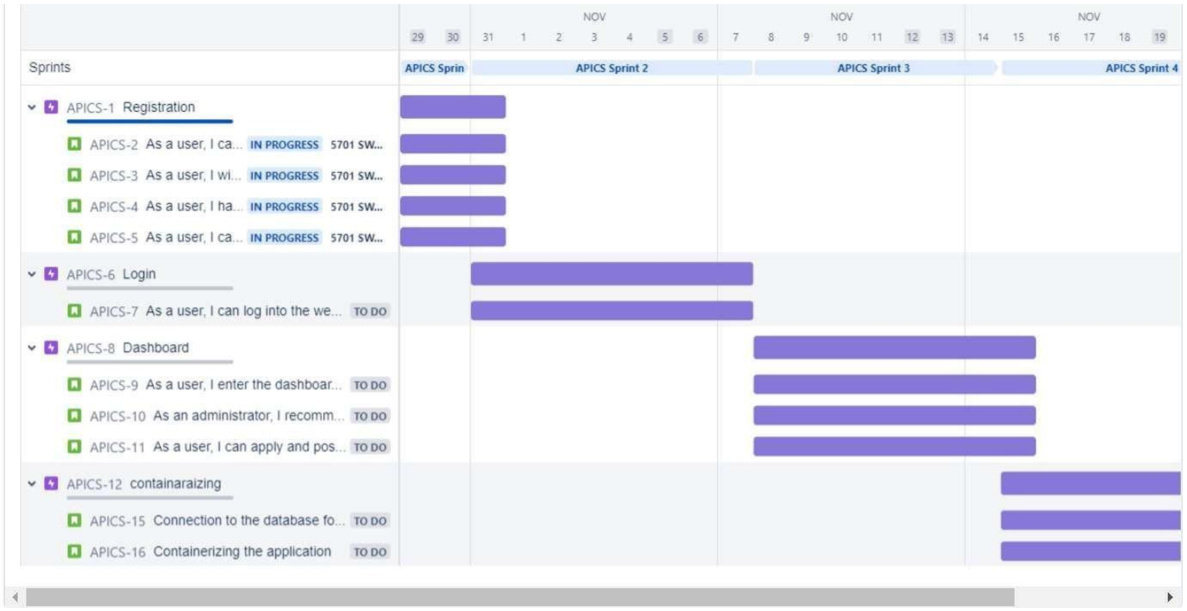
6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation & Delivery Schedule:

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Story Points**	**Priority**	**Team Members**
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	12	High	Aravindan Anusha
Sprint-1		USN-2	As a user, I can register for the application through E-mail	4	Medium	Aravindan Anusha
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the Application	2	Medium	Aravindan Anusha
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High	Aravindan Anusha

Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	10	High	Prasanth Kirunraj
Sprint-2	Add items to cart	USN-6	As a user, I can add the products I wish to buy to the carts.	10	Medium	Prasanth Kirunraj
Sprint-3	Stock Update	USN-7	As a user, I can add products which are not available in the dashboardto the stock list.	20	Medium	Prasanth Aravindan
Sprint-4	Requ est to Cust omer Care and email alert	USN-8	As a user, I can contact the Customer Care Executive and request any services I want from the customer care.	10	Low	Kiruraj Anusha
Sprint-4	Contact Administr ator	USN-9	I can be able to report any difficulties I experience as a report	10	Medium	Kiruraj Anusha

6.2 Reports from JIRA:



APICS-1 Registration
APICS-2 As a user, I ca... IN PROGRESS 5701 SW...
APICS-3 As a user, I wi... IN PROGRESS 5701 SW...
APICS-4 As a user, I ha... IN PROGRESS 5701 SW...
APICS-5 As a user, I ca... IN PROGRESS 5701 SW...
APICS-6 Login
APICS-7 As a user, I can log into the we... TO DO
APICS-8 Dashboard
APICS-9 As a user, I enter the dashboar... TO DO
APICS-10 As an administrator, I recomm... TO DO
APICS-11 As a user, I can apply and pos... TO DO
APICS-12 containaraizing
APICS-15 Connection to the database fo... TO DO
APICS-16 Containerizing the application TO DO

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

7.1 Feature 1

1. Self-service On-Demand

This is one of the most essential and significant characteristics of cloud computing. This means that cloud computing enables clients to regularly monitor the abilities, allotted network storage, and server uptime. Therefore, it is one of the most fundamental features of cloud computing that helps clients control various computing abilities as per their requirements.

2. Resources Pooling

This is also a fundamental characteristic of cloud computing. Pooling resources means that a cloud service provider can distribute resources for more than one client and provide them with different services according to their needs. Resource Pooling is a multi-client plan useful for data storing, bandwidth services and data processing services. The provider administers the data stored in real-time without conflicting with the need for data.

3. Easy Maintenance

This is one of the best cloud characteristics. Cloud servers are easy to maintain with low to almost zero downtime. Cloud Computing powered resources undergo several updates frequently to optimize their capabilities and potential. The updates are more viable with the devices and perform quicker than the previous versions.

4. Economical

This kind of service is economical as it efficiently reduces IT costs and data storage expenditure. Moreover, most cloud computing services are free. Even if there are paid plans, it's only to expand storage capacity, and these costs are often very nominal. This is a massive advantage of using cloud computing services.

5. Rapid Elasticity and Scalability

The best part of using cloud storage is that it can easily handle all the workload and data load concerning storage. Furthermore, as it is fully automated, businesses and organizations can save heavily on manual labor and technical staffing as cloud services are elastic, scalable and automated. This is one of the significant advantages of using cloud services.

API Features:

1. HTTPSISSL certificates Programming cheat sheets
2. Try for free: Red Hat Learning Subscription
3. eBook: An introduction to programming with Bash
   Bash Shell Scripting Cheat Sheet
4. eBook: Modernizing Enterprise Java

The gold standard for the web is HTTPS using SSL certificates, and Let's Encrypt can help you achieve this. It is a free, automated, and open certificate authority from the non-profit Internet Security Research Group (ISRG).

2. Cross-origin resource sharing

CORS is a browser-specific security policy preflight check. If your APIserver is not in the same domain as the requesting client's domain, you will need to deal with CORS. For example, if your server is running onapi.domain-a.com and gets a client request from domain-b.com, Coarsens an HTTP precheck request to see if your API service will accept client-side requests from the client's domain.

3. Authentication and JSON Web Tokens

There are several approaches to validate an authenticated user in your API, but one of the best ways is to use JSON Web Tokens (JWT). These tokens are signed using various types of well-known cryptographic libraries. When a client logs in, an identity-management service provides the client with a JWT. The client can then use this token to make requests to the API. The API has access to a public key or a secret that it uses to verify the token. There are several libraries available to help verify tokens, including honeytoken. For more information about JWT and the libraries that support it in every language, check out JWT.io.

4. Authorizations and scopes

Authentication (or identity verification) is important, but so isauthorization, i.e., does the verified client have the privilege to execute this request? This is where scopes are valuable. When the client authenticates with the identity management server and a JWT token is created, having the identity management service provide the scopes for the given authenticated client can enable the API service to determine if this verified client request can be performed without having to perform an additional costly lookup to an access control list.

Feature 2:

Docker Features:

1. <u>Faster and Easier configuration:</u>
   It is one of the key features of Docker that helps you in configuring the system in a faster and easier manner. Due to this feature, codes can be deployed in less time and with fewer efforts. The infrastructure is not linked with the environment of the application as Docker is used with a wide variety of environments.

2. <u>Application isolation:</u>
   Docker provides containers that are used to run applications in an isolated environment. Since each container is independent, Docker can execute any kind of application.

3. <u>Increase in productivity:</u>
   It helps in increasing productivity by easing up the technical configuration and rapidly deploying applications. Moreover, it not only provides an isolated environment to execute applications, but it reduces the resources as well.

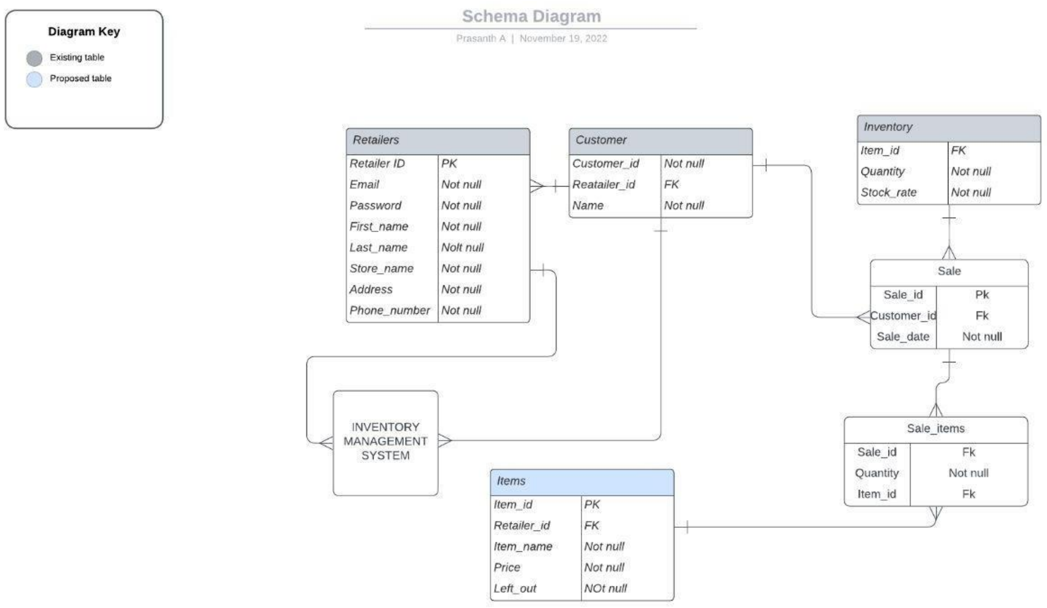
4. <u>Swarm:</u>
   Swarm is a clustering and scheduling tool for Docker containers. At the front end, it uses the Docker API, which helps us to use various tools to control it. It is a self-organizing group of engines that enables pluggable backends.

5. <u>Services:</u>
   Services is a list of tasks that specifies the state of a container inside a cluster. Each task in the Services lists one instance of a container that should be running, while Swarm schedules them across the nodes.

Kubernetes Features:

1. Auto-scaling. Automatically scale containerized applications and their resources up or down based on usage
2. Lifecycle management. Automate deployments and updates with the ability to:
   i. Rollback to previous versions
   ii. Pause and continue a deployment
3. Declarative model. Declare the desired state, and K8s works in the background to maintain that state and recover from any failures.
4. Resilience and self-healing. Auto placement, auto restart, auto replication and auto scaling provide application self-healing
5. Persistent storage. Ability to mount and add storage dynamically
6. Load balancing. Kubernetes supports a variety of internal and external load balancing options to address diverse needs.

7.2 Database Schema:



8. TESTING

8.1 Test Cases:

1. Verify that user is already registered or not
2. Verify the Ul elements in Login/Signup popup
3. Verify user can log into application with Valid credentials
4. Verify that categories of actions(adding items,customer,sale,etc) are shown in homepage
5. Verify that actions is displayed in homepage
6. Verify that when clicked on actions it is redirected to correct page
7. Verify that actions are correctly performed

8.2 User Acceptance Testing:

The purpose of this documents to briefly explain the test coverage and open issues of the Inventory management system for retailers Application project at the time of the release User Acceptance Testing (UAT). This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

1.Defect Analysis

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

2.Test Case Analysis

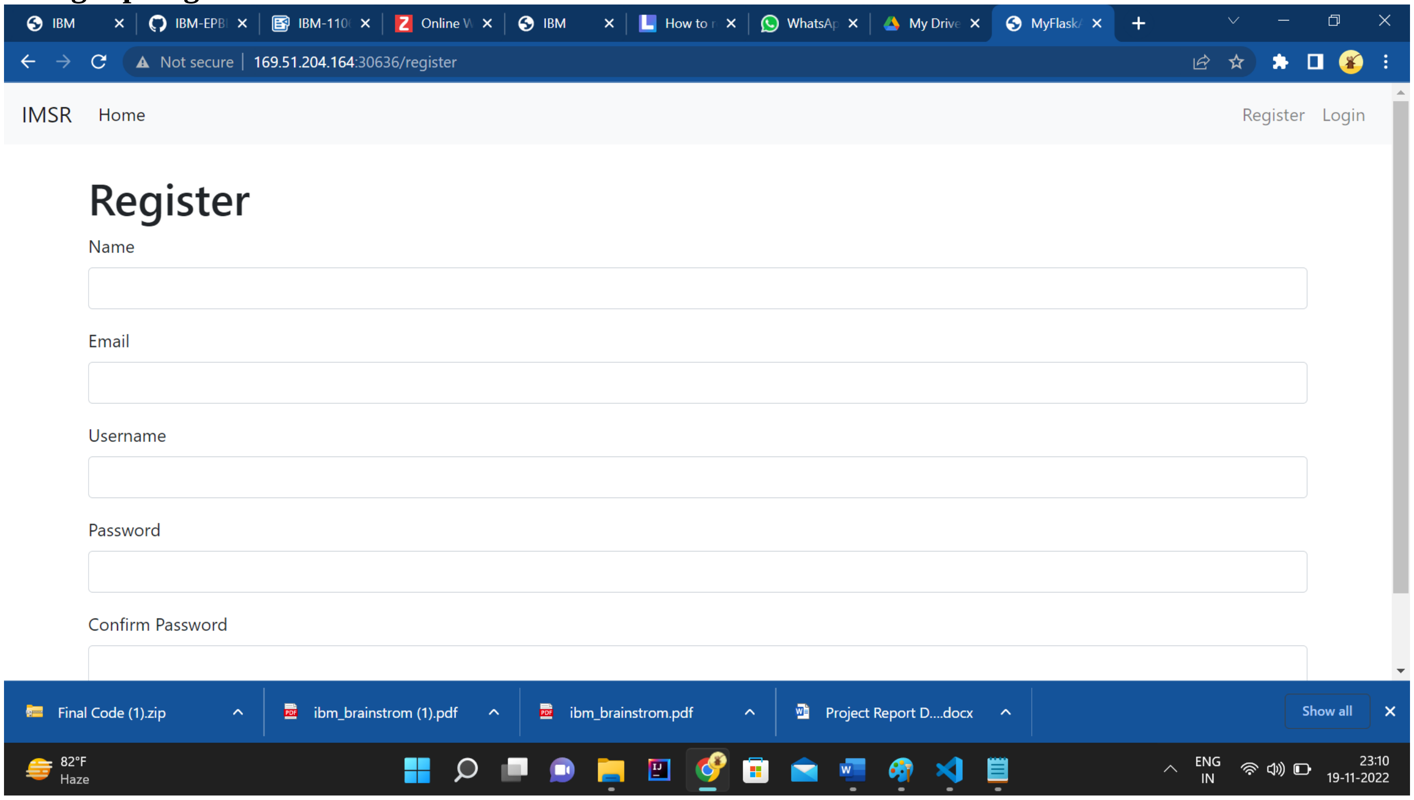
This report shows the number of test cases that have passed,failed, and untested

9. RESULTS

9.1 Performance Metrics:

The application performance index, or Apex score, has become an industry standard for tracking the relative performance of an application. It works by specifying a goal for how long a specific web request or transaction should take. Those transactions are then bucketed into satisfied (fast), tolerating (sluggish), too slow, and failed requests. A simple math formula is then applied to provide a score from 0 to 1.

Signup Page:



Register

Name

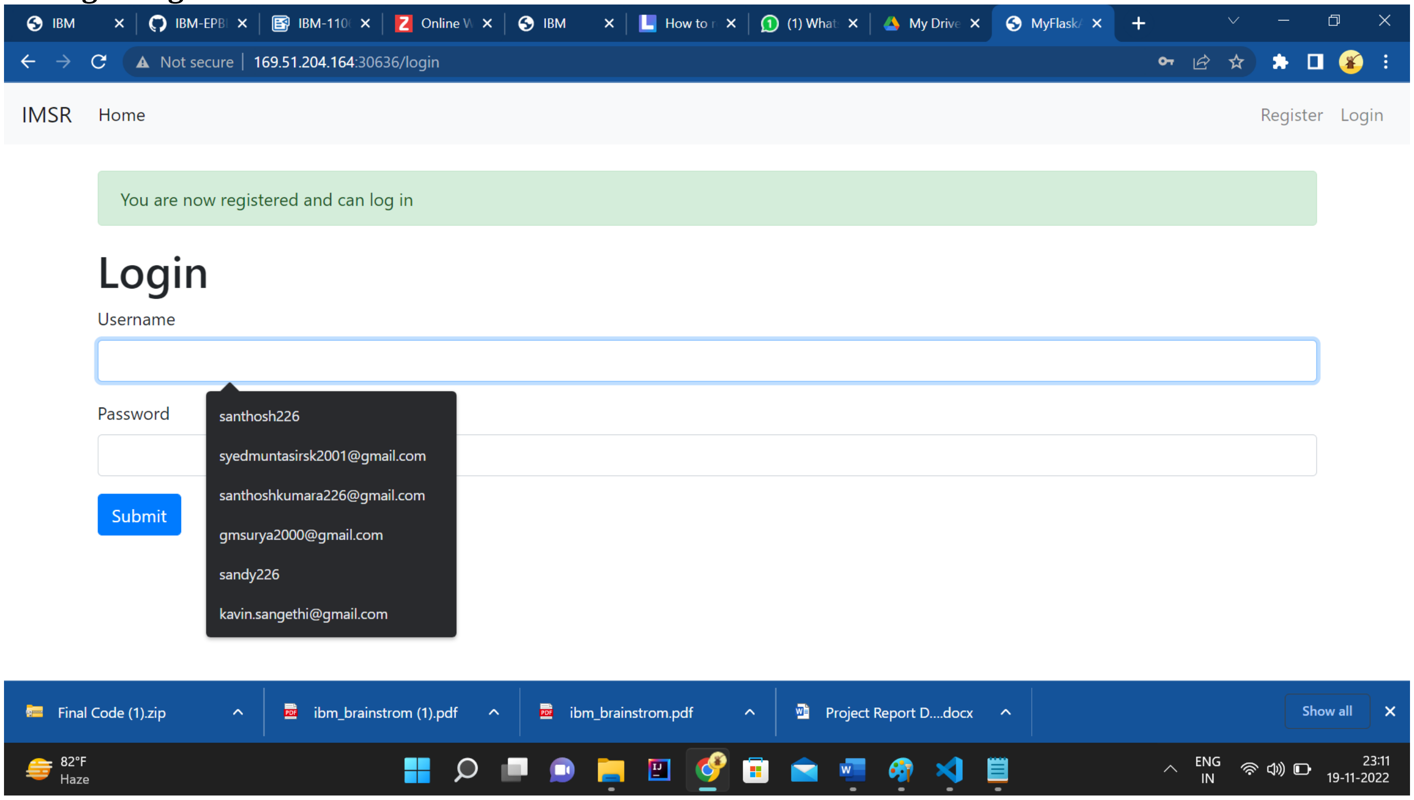
Email

Username

Password

Confirm Password

Login Page:



169.51.204.164:30636/login

IMSR Home Register Login

You are now registered and can log in

Login

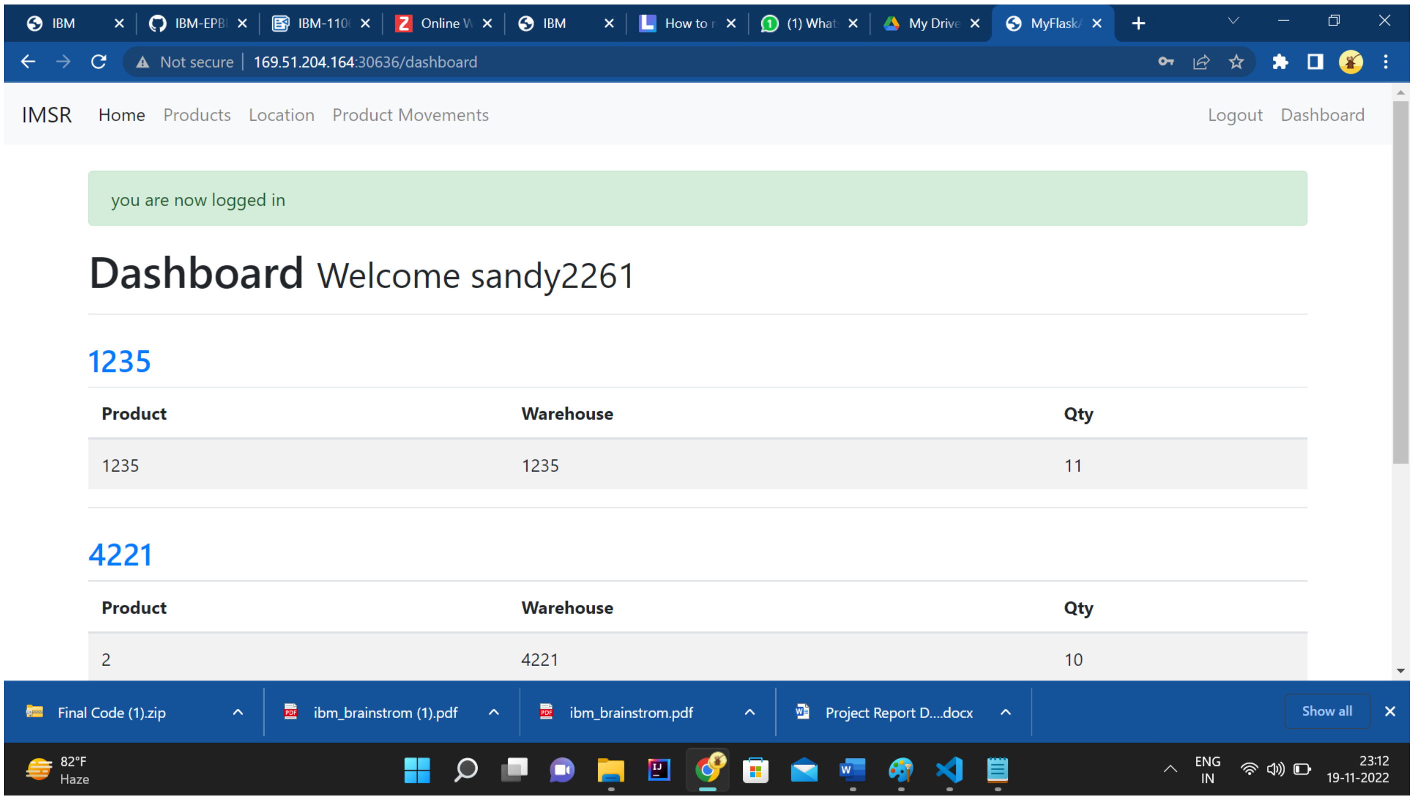
Username

Password

Submit

santhosh226
syedmuntasirsk2001@gmail.com
santhoshkumara226@gmail.com
gmsurya2000@gmail.com
sandy226
kavin.sangethi@gmail.com

Dashboard:



169.51.204.164:30636/dashboard

IMSR Home Products Location Product Movements Logout Dashboard

you are now logged in

Dashboard Welcome sandy2261

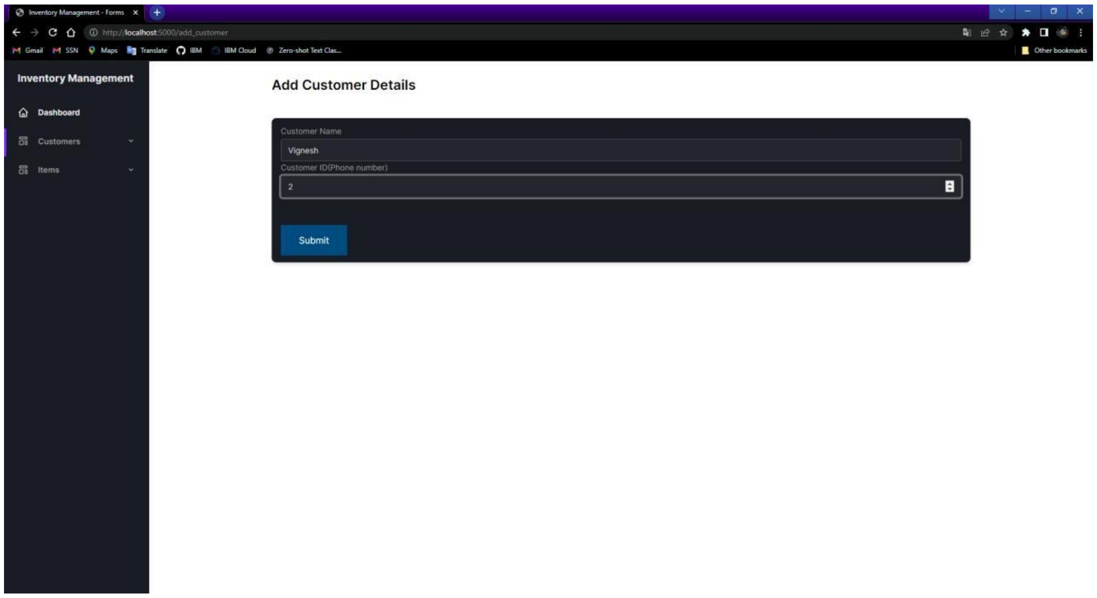
1235

Product	Warehouse	Qty
1235	1235	11

4221

Product	Warehouse	Qty
2	4221	10

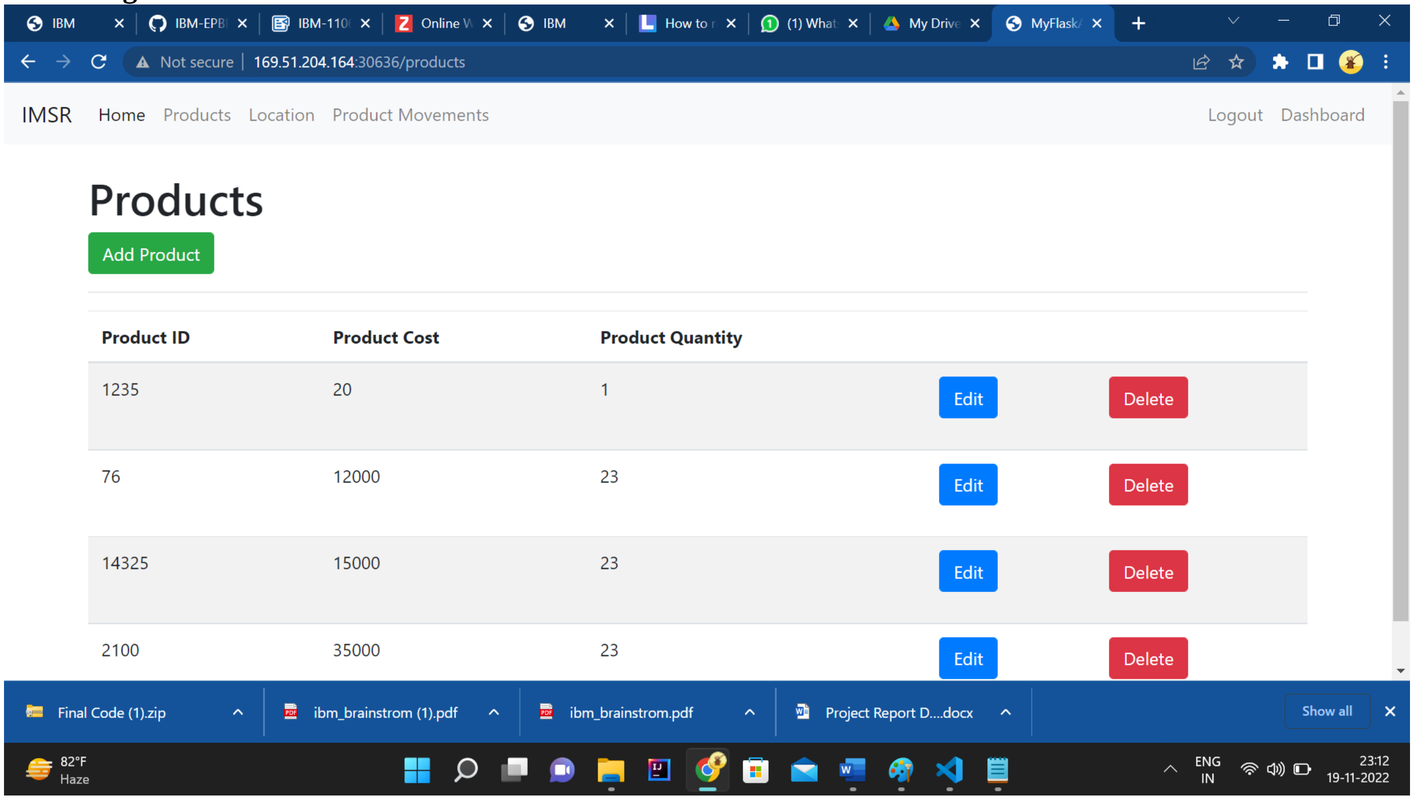
Add Customer:



Add Customer Details

Customer Name
Vignesh
Customer ID(Phone number)
2

Storing In Database:



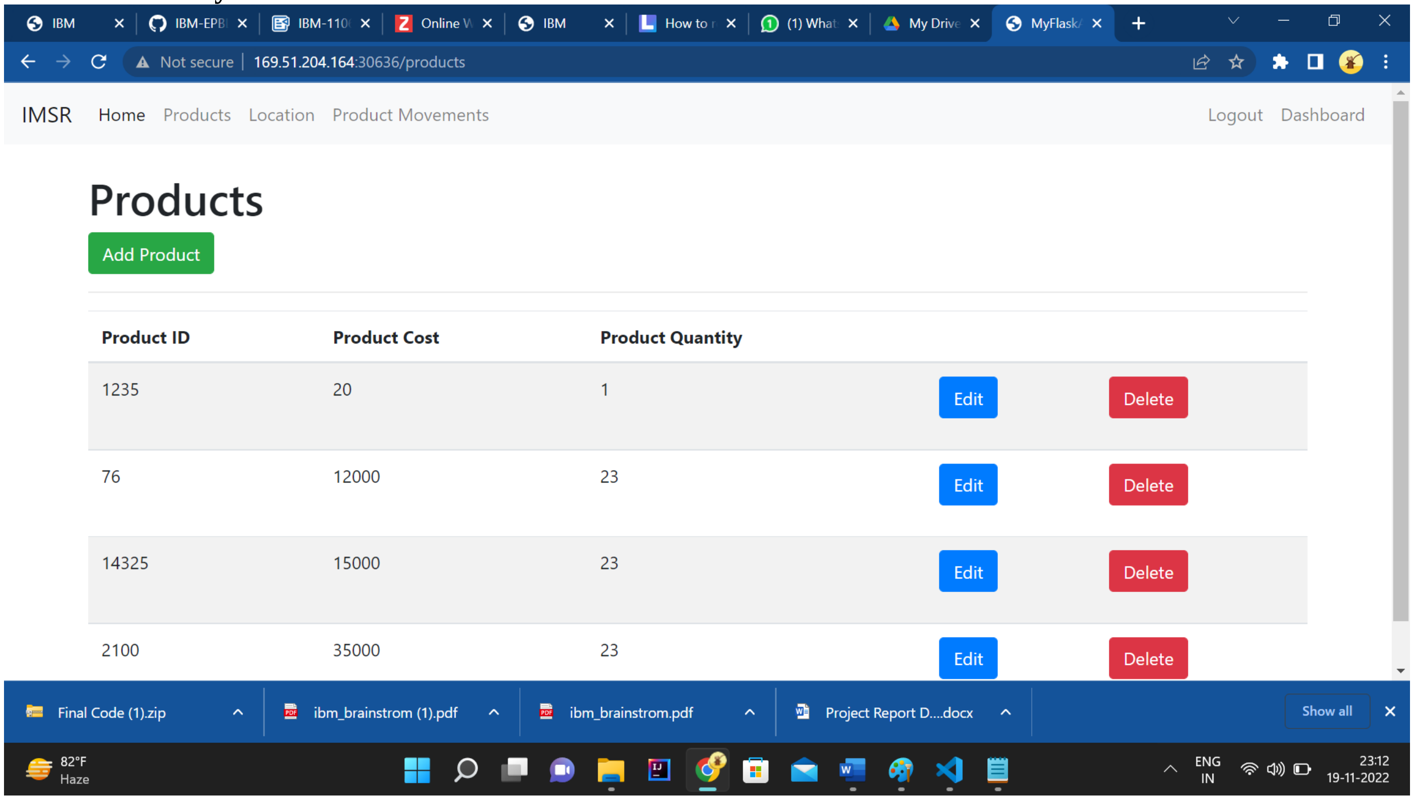
IMSR Home Products Location Product Movements Logout Dashboard

Products

Add Product

Product ID	Product Cost	Product Quantity		
1235	20	1	Edit	Delete
76	12000	23	Edit	Delete
14325	15000	23	Edit	Delete
2100	35000	23	Edit	Delete

Add Inventory:



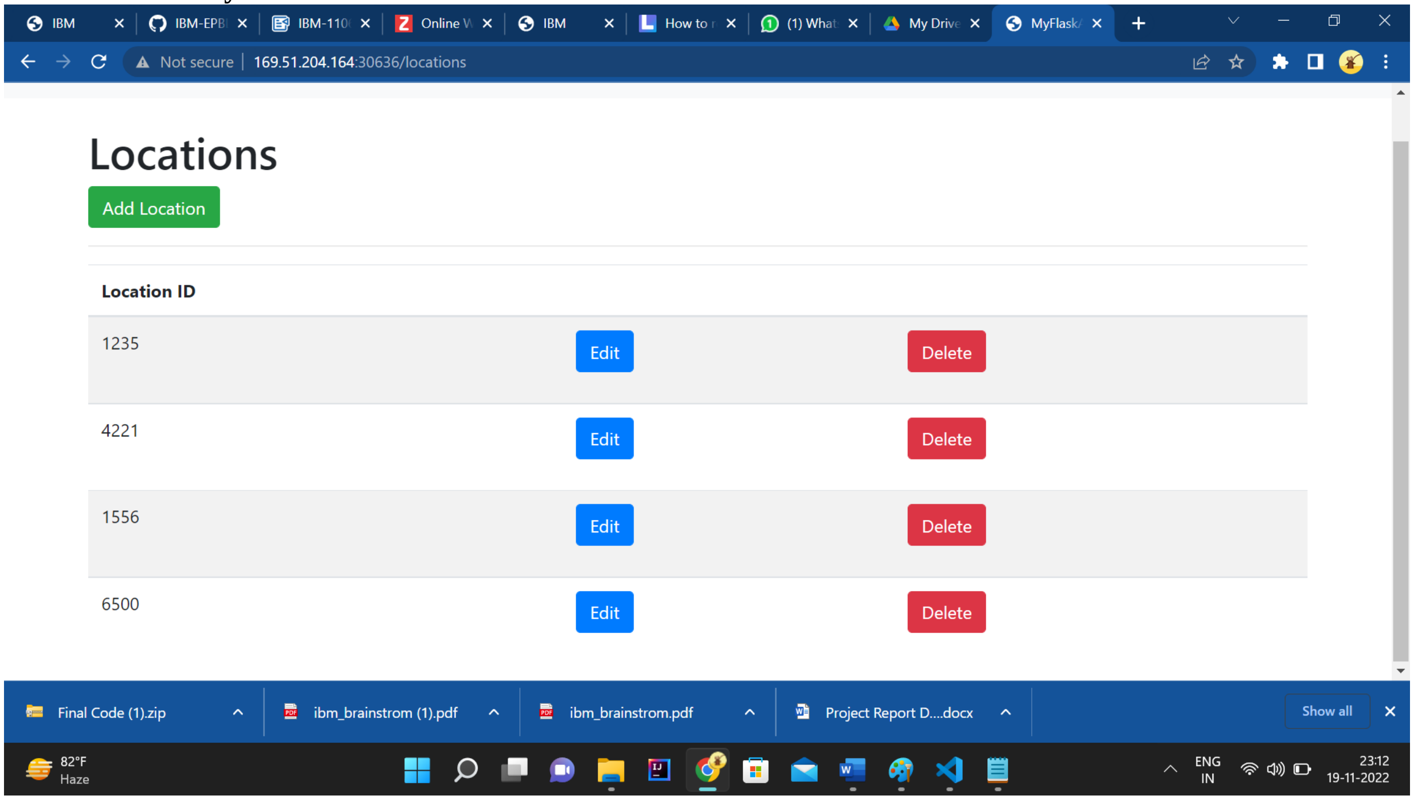
IMSR Home Products Location Product Movements Logout Dashboard

Products

Add Product

Product ID	Product Cost	Product Quantity		
1235	20	1	Edit	Delete
76	12000	23	Edit	Delete
14325	15000	23	Edit	Delete
2100	35000	23	Edit	Delete

View Inventory:



Locations

Add Location

Location ID		
1235	Edit	Delete
4221	Edit	Delete
1556	Edit	Delete
6500	Edit	Delete

10. ADVANTAGES & DISADVANTAGES

Advantages:

1. **It helps to maintain the right amount of stocks:** contrary to the belief that is held by some people, inventory management does not seek to reduce the amount of inventory that you have in stock, however, it seeks to maintain an equilibrium point where your inventory is working at a maximum efficiency and you do not have to have many stocks or too few stocks at hand at any particular point in time. The goal is to find that zone where you are never losing money in your inventory in either direction. With the aid of an efficient inventory management strategy, it is easy to improve the accuracy of inventory order.
2. **It leads to a more organized warehouse:** with the aid of a good inventory management system, you can easily organize your warehouse. If your warehouse is not organized, you will find it very difficult to manage your inventory. A lot of businesses choose to optimize their warehouse by putting the items that have the highest sales together in a place that is easy to access in the warehouse. This ultimately helps to speed up order fulfilment and keeps clients happy.
3. **It saves time and money:** an effective inventory management system can translate to time and money saved on the part of the business. By keeping track of the product that you already have at hand, you can save yourself the hassles of having to do an inventory recount in order to ensure your records are accurate. It also allows you to save cash that would have otherwise been spent on slow moving products.
4. **Improves efficiency and productivity:** inventory management devices like bar code scanners and inventory management software can help to greatly increase the efficiency and productivity of a business. They do this by eliminating the manual way of doing things thus allowing employees to do other more important things for the business.
5. **Schedule maintenance:** once you get hold of a new appliance, you can begin to schedule routine and preventative maintenance, issue work order to your staff and track that the maintenance was actually carried out. This will help to elongate the life span of that particular asset.

Disadvantages:

1. **Bureaucracy:** even though inventory management allows employees at every level of the company to read and manipulate company stock and product inventory, the infrastructure required to build such a system adds a layer of bureaucracy to the whole process and the business in general. In instances where inventory control is in-house, this includes the number of new hires that are not present to regulate the warehouse and facilitate transactions. In instances where the inventory management is in the hands of a third party, the cost is a subscription price and a dependence on another separate company to manage its infrastructure. No matter the choice you go for, it translates to a higher overhead cost and more layers of management between the owner and the customer. From the view point of the customer, a problem that requires senior management to handle will take a longer period of time before it will be trashed out.
2. **Impersonal touch:** another disadvantage of inventory management is a lack of personal touch. Large supply chain management systems make products more accessible across the globe and most provide customer service support in case of difficulty, but the increase in infrastructure can often mean a decrease in the personal touch that helps a company to stand out above the rest. For instance, the sales manager of a small manufacturing company that sells plumbing supplies to local plumbers can throw in an extra box of washers or elbows at no charge to the customer without raising any alarms. This is done for the sake of customer relations and often makes the customer feel like he is special. While free materials can also be provided under inventory management, processing time and paper work make obtaining the material feel more

like a chore for the customer or even an entitlement.

3. **Production problem:** even though inventory management can reveal to you the amount of stock you have at hand and the amount that you have sold off, it can also hide production problems that could lead to customer service disasters. Since the management places almost all of its focus on inventory management to the detriment of quality control, broken or incorrect items that would normally be discarded are shipped along with wholesome items.
4. **Increased space is need to hold the inventory:** in order to hold inventory, you will need to have space so unless the goods you deal in are really small in size, then you will need a warehouse to store it. In addition, you will also need to buy shelves and racks to store your goods, forklifts to move around the stock and of course staff. The optimum level of inventory for a business could still be a lot of goods and they will need space to be stored in and in some cases additional operational costs to manage the inventory. This will in turn increase cost and impact negatively on the amount of profit the business makes.
5. **Complexity:** some methods and strategies of inventory management can be relatively complex and difficult to understand on the part of the staff. This may result in the need for employees to undergo training in order to grasp how the system works.

11. CONCLUSION

We proposed an application for inventory management system task. This application facilitates the opportunities of inventory management process as well as it allows the use of a variety of actions like adding customers,items,sales,inventory and view customers,inventory,items and sale. Moreover, we also contribute making publicly available a new dataset containing inventory management. Future directions of our work will focus on performing a more exhaustive evaluation considering a greater amount of methods and data as well as a comprehensive evaluation of the impact of inventory management

12. FUTURE SCOPE

The future is and will remain unknown to us , but fact is there is a issue of maintaining inventory in the market world, the inventory management system will exist and will grow in proportionate with demand

13. APPENDI

X 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=HOS
TNAME"
   "PORT=30756;"

"SECURITY=SSL;"

"SSLServerCertificat
e=DigiCertGlobalRo
otCA.crt;"
   "UID=uid;"
   "PWD=pwd;", '', '')



# Index
@app.route('/')
```

```
def index():
    return
render_template('hom
e.html')


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

    products = []
    row =
ibm_db.fetch_assoc(s
tmt)
    while (row):

products.append(row)
        row =
ibm_db.fetch_assoc(s
tmt)
    products =
tuple(products)
    # print(products)

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


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

    locations = []
    row =
ibm_db.fetch_assoc(s
tmt)
    while (row):

locations.append(row
)
        row =
ibm_db.fetch_assoc(s
tmt)
    locations =
tuple(locations)
    # print(locations)

    if result > 0:
        return
render_template('loca
tions.html',
locations=locations)
    else:
        msg = 'No
locations found'
        return
render_template('loca
tions.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(s
tmt)
    while (row):

movements.append(r
ow)
        row =
ibm_db.fetch_assoc(s
tmt)
    movements =
tuple(movements)
```

```
    #
print(movements)

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



# Register Form
Class
class
RegisterForm(Form):
    name =
StringField('Name',
[validators.Length(mi
n=1, max=50)])
    username =
StringField('Usernam
e',
[validators.Length(mi
n=1, max=25)])
    email =
StringField('Email',
[validators.length(mi
n=6, max=50)])
    password =
PasswordField('Pass
word', [

validators.DataRequir
ed(),

validators.EqualTo('c
onfirm',
message='Passwords
do not match')
    ])
    confirm =
PasswordField('Confi
rm 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.da
ta))

        sql1 = "INSERT
```

```
INTO users(name,
email, username,
password)
VALUES(?,?,?,?)"
        stmt1 =
ibm_db.prepare(conn,
sql1)

ibm_db.bind_param(s
tmt1, 1, name)

ibm_db.bind_param(s
tmt1, 2, email)

ibm_db.bind_param(s
tmt1, 3, username)

ibm_db.bind_param(s
tmt1, 4, password)

ibm_db.execute(stmt
1)
        # 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('regi
ster.html',
form=form)



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

password_candidate
=
request.form['passwor
d']

        sql1 = "Select *
from users where
username = ?"
        stmt1 =
ibm_db.prepare(conn,
sql1)

ibm_db.bind_param(s
tmt1, 1, username)
        result =
ibm_db.execute(stmt
1)
        d =
ibm_db.fetch_assoc(s
tmt1)
        if result > 0 and
d:
            # Get the
stored hash
            data = d
            password =
data['PASSWORD']

            # compare
passwords
            if
sha256_crypt.verify(p
assword_candidate,
password):
                # Passed

session['logged_in'] =
True

session['username'] =
username

                flash("you
are now logged in",
"success")
                return
redirect(url_for('dash
board'))
            else:
                error =
'Invalid Login'
                return
render_template('logi
n.html', error=error)
            # Close
connection
            cur.close()
        else:
            error =
'Username not found'
            return
render_template('logi
n.html', error=error)
    return
render_template('logi
n.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('/dashboa
rd')
@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(stmt
2)

ibm_db.execute(stmt
3)

    products = []
    row =
ibm_db.fetch_assoc(s
tmt2)
    while (row):

products.append(row)
        row =
ibm_db.fetch_assoc(s
tmt2)
    products =
tuple(products)

    locations = []
    row2 =
ibm_db.fetch_assoc(s
tmt3)
    while (row2):

locations.append(row
2)
        row2 =
ibm_db.fetch_assoc(s
tmt3)
    locations =
tuple(locations)

    locs = []
    for i in locations:

locs.append(list(i.val
ues())[0])

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

```
products found'
        return
render_template('dash
board.html',
msg=msg)


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


# Add Product
@app.route('/add_pro
duct',
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.dat
a
        product_num =
form.product_num.da
ta

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

ibm_db.bind_param(s
tmt1, 1, product_id)

ibm_db.bind_param(s
tmt1, 2, product_cost)

ibm_db.bind_param(s
tmt1, 3,
product_num)


ibm_db.execute(stmt
1)

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

```
        return
redirect(url_for('prod
ucts'))

    return
render_template('add
_product.html',
form=form)



# Edit Product
@app.route('/edit_pro
duct/<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(s
tmt1, 1, id)
    result =
ibm_db.execute(stmt
1)
    product =
ibm_db.fetch_assoc(s
tmt1)

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

```
    # populate product
form fields

form.product_id.data
=
product['PRODUCT_
ID']

form.product_cost.dat
a =
str(product['PRODU
CT_COST'])

form.product_num.da
ta =
str(product['PRODU
CT_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(s
tmt2, 1, product_id)

ibm_db.bind_param(s
tmt2, 2, product_cost)

ibm_db.bind_param(s
tmt2, 3,
product_num)

ibm_db.bind_param(s
tmt2, 4, id)

ibm_db.execute(stmt
2)

        flash("Product
Updated", "success")

        return
redirect(url_for('prod
ucts'))

    return
render_template('edit
_product.html',
form=form)


# Delete Product
@app.route('/delete_p
roduct/<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(s
tmt2, 1, id)

ibm_db.execute(stmt
2)

    flash("Product
Deleted", "success")

    return
redirect(url_for('prod
ucts'))



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



# Add Location
@app.route('/add_loc
ation',
methods=['GET',
'POST'])
@is_logged_in
def add_location():
    form =
LocationForm(reques
t.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(s
tmt2, 1, location_id)

ibm_db.execute(stmt
2)

        flash("Location
Added", "success")

        return
redirect(url_for('locat
ions'))

    return
render_template('add
_location.html',
form=form)
```

```
# Edit Location
@app.route('/edit_loc
ation/<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(s
tmt2, 1, id)
    result =
ibm_db.execute(stmt
2)
    location =
ibm_db.fetch_assoc(s
tmt2)
    # Get form
    form =
LocationForm(reques
t.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(s
tmt2, 1, location_id)

ibm_db.bind_param(s
tmt2, 2, id)

ibm_db.execute(stmt
2)

        flash("Location
Updated", "success")

        return
redirect(url_for('locat
ions'))

    return
render_template('edit
_location.html',
form=form)
```

```
# Delete Location
@app.route('/delete_l
ocation/<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(s
tmt2, 1, id)

ibm_db.execute(stmt
2)

    flash("Location
Deleted", "success")

    return
redirect(url_for('locat
ions'))



# Product Movement
Form Class
class
ProductMovementFor
m(Form):
    from_location =
SelectField('From
Location', choices=[])
```

```
    to_location =
SelectField('To
Location', choices=[])
    product_id =
SelectField('Product
ID', choices=[])
    qty =
IntegerField('Quantit
y')


class
CustomError(Excepti
on):
    pass


# Add Product
Movement
@app.route('/add_pro
duct_movements',
methods=['GET',
'POST'])
@is_logged_in
def
add_product_movem
ents():
    form =
ProductMovementFor
m(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(stmt
2)

ibm_db.execute(stmt
3)

    products = []
    row =
ibm_db.fetch_assoc(s
tmt2)
    while (row):

products.append(row)
        row =
ibm_db.fetch_assoc(s
tmt2)
    products =
tuple(products)

    locations = []
    row2 =
ibm_db.fetch_assoc(s
tmt3)
    while (row2):

locations.append(row
2)
        row2 =
ibm_db.fetch_assoc(s
tmt3)
    locations =
tuple(locations)

    prods = []
    for p in products:

prods.append(list(p.v
alues())[0])

    locs = []
    for i in locations:

locs.append(list(i.val
ues())[0])


form.from_location.c
hoices = [(l, l) for l in
locs]

form.from_location.c
hoices.append(("Mai
n Inventory", "Main
Inventory"))

form.to_location.choi
ces = [(l, l) for l in
locs]

form.to_location.choi
ces.append(("Main
Inventory", "Main
Inventory"))

form.product_id.choi
ces = [(p, p) for p in
prods]

    if request.method
```

```
== 'POST' and
form.validate():
        from_location =
form.from_location.d
ata
        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(s
tmt2, 1, to_location)

ibm_db.bind_param(s
tmt2, 2, product_id)
            result =
ibm_db.execute(stmt
2)
            result =
ibm_db.fetch_assoc(s
tmt2)
            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(s
tmt2, 1, q)

ibm_db.bind_param(s
tmt2, 2, to_location)
```

```
ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.execute(stmt
2)



print(product_id)
                    print(qty)

print(to_location)

print(from_location)
                    sql2 =
"INSERT into
productmovements(pr
oduct_id, qty,
to_location,from_loca
tion) VALUES(?, ?,
?, ?)"
                    stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1, product_id)

ibm_db.bind_param(s
tmt2, 2, qty)

ibm_db.bind_param(s
tmt2, 3, to_location)

ibm_db.bind_param(s
tmt2, 4,
from_location)

ibm_db.execute(stmt
2)
        else:

            sql2 =
"INSERT into
product_balance(prod
uct_id, location_id,
qty) values(?, ?, ?)"
            stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1, product_id)

ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, qty)

ibm_db.execute(stmt
2)

            sql2 =
"INSERT into
productmovements(fr
om_location,
to_location,
product_id, qty)
VALUES(?, ?, ?, ?)"
            stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1,
from_location)
```

```
ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.bind_param(s
tmt2, 4, qty)

ibm_db.execute(stmt
2)

        sql = "select
product_num from
products where
product_id=?"
        stmt =
ibm_db.prepare(conn,
sql)

ibm_db.bind_param(s
tmt, 1, product_id)
        current_num
=
ibm_db.execute(stmt)
        current_num
=
ibm_db.fetch_assoc(s
tmt)

        sql2 =
"Update products set
product_num=?
where product_id=?"
        stmt2 =
ibm_db.prepare(conn,
sql2)
```

```
ibm_db.bind_param(s
tmt2, 1,
current_num['PROD
UCT_NUM'] - qty)

ibm_db.bind_param(s
tmt2, 2, product_id)

ibm_db.execute(stmt
2)

            alert_num =
current_num['PROD
UCT_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(s
tmt2, 1,
from_location)

ibm_db.bind_param(s
tmt2, 2, product_id)
            result =
ibm_db.execute(stmt
2)
            result =
ibm_db.fetch_assoc(s
tmt2)


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(s
tmt2, 1, q)

ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.execute(stmt
2)

                    sql2 =
"INSERT into
productmovements(fr
om_location,
to_location,
product_id, qty)
VALUES(?, ?, ?, ?)"
                    stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1,
from_location)

ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.bind_param(s
tmt2, 4, qty)

ibm_db.execute(stmt
```

2)

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

                    sql =
"select product_num
from products where
product_id=?"
                    stmt =
ibm_db.prepare(conn,
sql)

ibm_db.bind_param(s
tmt, 1, product_id)

current_num =
ibm_db.execute(stmt)

current_num =
ibm_db.fetch_assoc(s
tmt)

                    sql2 =
"Update products set
product_num=?
where product_id=?"
                    stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1,
current_num['PROD
UCT_NUM'] + qty)
```

```
ibm_db.bind_param(s
tmt2, 2, product_id)

ibm_db.execute(stmt
2)



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_i
d, 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(s
tmt, 1, from_location)

ibm_db.bind_param(s
tmt, 2, product_id)
            result =
ibm_db.execute(stmt)
            result =
ibm_db.fetch_assoc(s
tmt)

            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(s
tmt2, 1, q)

ibm_db.bind_param(s
tmt2, 2,
from_location)

ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.execute(stmt
2)
                    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_i
d, 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(s
tmt, 1, to_location)

ibm_db.bind_param(s
tmt, 2, product_id)
                result =
ibm_db.execute(stmt)
                result =
ibm_db.fetch_assoc(s
tmt)

                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(s
tmt2, 1, q)

ibm_db.bind_param(s
tmt2, 2, to_location)
```

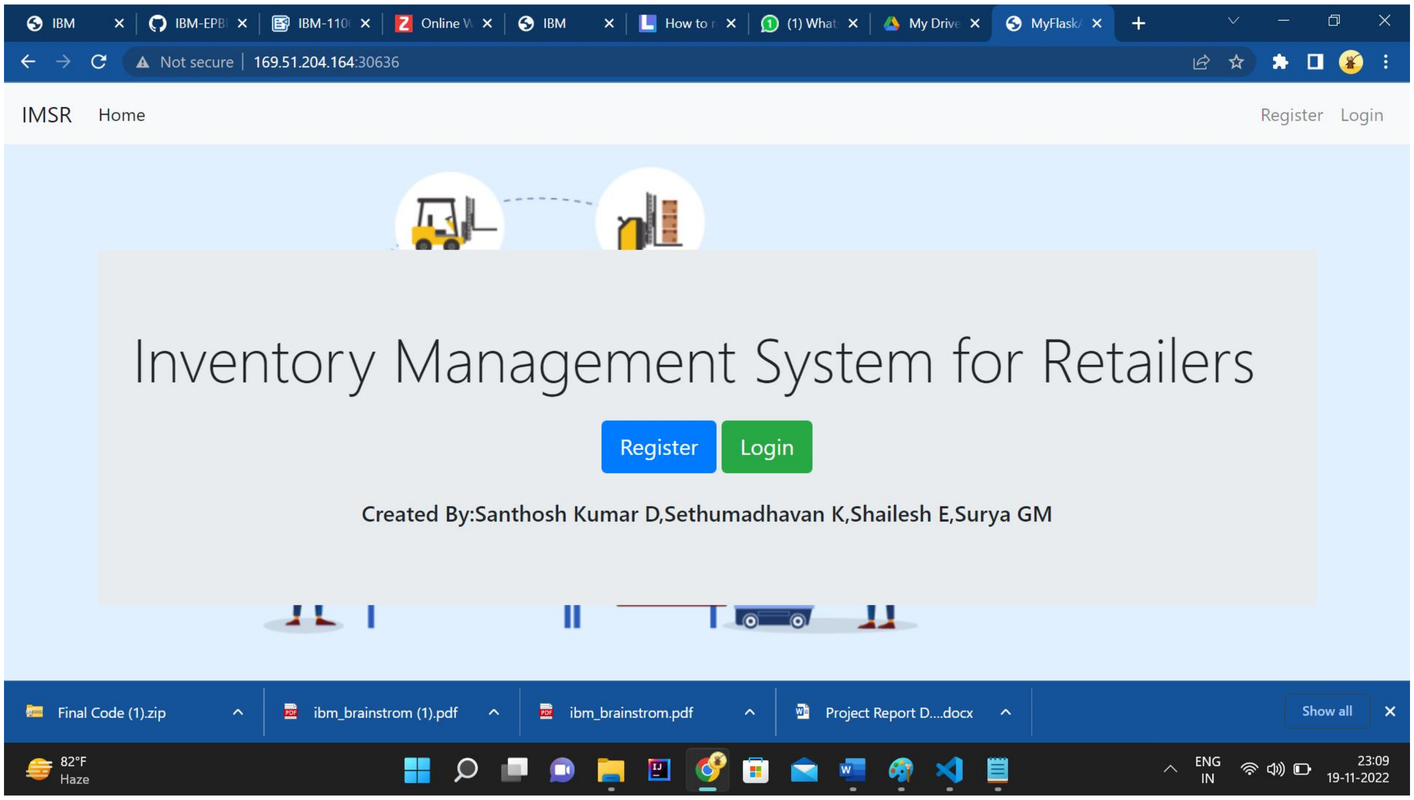
```
ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.execute(stmt
2)

            else:

                sql2 =
"INSERT into
product_balance(prod
uct_id, location_id,
qty) values(?, ?, ?)"
                stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1, product_id)

ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, qty)

ibm_db.execute(stmt
2)
            sql2 =
"INSERT into
productmovements(fr
om_location,
to_location,
product_id, qty)
VALUES(?, ?, ?, ?)"
            stmt2 =
ibm_db.prepare(conn,
sql2)
```

```
ibm_db.bind_param(s
tmt2, 1,
from_location)

ibm_db.bind_param(s
tmt2, 2, to_location)

ibm_db.bind_param(s
tmt2, 3, product_id)

ibm_db.bind_param(s
tmt2, 4, qty)

ibm_db.execute(stmt
2)


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


render_template('prod
ucts.html',
form=form)

        return
redirect(url_for('prod
uct_movements'))

    return
render_template('add
_product_movements
.html', form=form)


# Delete Product
```

```
Movements
@app.route('/delete_p
roduct_movements/<
string:id>',
methods=['POST'])
@is_logged_in
def
delete_product_move
ments(id):
    sql2 = "DELETE
FROM
productmovements
WHERE
movement_id=?"
    stmt2 =
ibm_db.prepare(conn,
sql2)

ibm_db.bind_param(s
tmt2, 1, id)

ibm_db.execute(stmt
2)

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

    return
redirect(url_for('prod
uct_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)
SENDGRID.PY
import smtplib
from
email.mime.multipart
import
MIMEMultipart
from email.mime.text
import MIMEText

def alert(main_msg):
   mail_from =
'santhoshkumara226
@gmail.com'
   mail_to =
'sethumadhavan2002
@gmail.com'
   msg =
MIMEMultipart()
   msg['From'] =
mail_from
   msg['To'] = mail_to
   msg['Subject'] =
'!Alert Mail On
Product Shortage! -
Regards'
   mail_body =
main_msg

msg.attach(MIMETe
xt(mail_body))

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

server.login('apikey',
'api_key')

server.sendmail(mail
_from, mail_to,
msg.as_string())
    server.close()
    print("Mail sent
successfully!")
  except:
    print("Some
Issue, Mail not Sent
:(")
```

OUTPUT:



Inventory Management System for Retailers

Created By:Santhosh Kumar D,Sethumadhavan K,Shailesh E,Surya GM