



**NAALAIYA THIRAN PROJECT - 2022
19ECI01-PROFESSIONAL READINESS FOR
INNOVATION, EMPLOYABILITY AND
ENTREPRENEURSHIP**



INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

A PROJECT REPORT

Submitted by

TEAM ID: PNT2022TMID52830

YUGASHINI A	CITC1904062
AJAY ADDAGALLA	CITC1904064
GOPINATH S	CITC1904078
JANAKARAN M	CITC1904080

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE – 641 014

(Government Aided Autonomous Institution affiliated to Anna University)

ANNA UNIVERSITY: CHENNAI 600025

NOVEMBER 2022

PROJECT CALENDAR

Phase	Phase Description	Week	Dates	Activity Details
1	Preparation Phase (Pre- requisites, Registrations, Environment Set-up, etc.)	2	22 - 27 Aug 2022	Creation GitHub account & collaborate with Project repository in project workspace
2	Ideation Phase (Literature Survey, Empathize, Defining Problem Statement, Ideation)	2	29 Aug – 3rd Sept 2022	Literature survey (Aim, objective, problem statement and need for the project)
		3	5 - 10th Sept 2022	Preparing Empathy Map Canvas to capture the user Pains & Gains
		4	12 - 17 Sept 2022	Listing of the ideas using brainstorming session
3	Project Design Phase -I (Proposed Solution, Problem- Solution Fit, Solution Architecture)	5	19 - 24 Sept 2022	Preparing the proposed solution document
		6	26 Sept - 01 Oct 2022	Preparing problem - solution fit document & Solution Architecture
4	Project Design Phase -II (Requirement Analysis, Customer Journey, Data Flow Diagrams, Technology Architecture)	7	3 - 8 Oct 2022	Preparing the customer journey maps
		8	10 - 15 Oct 2022	Preparing the Functional Requirement Document & Data- Flow Diagrams and Technology Architecture
5	Project Planning Phase (Milestones & Tasks, Sprint Schedules)	9	17 - 22 Oct 2022	Preparing Milestone & Activity List, Sprint Delivery Plan
6	Project Development Phase (Coding & Solutioning, acceptance Testing, Performance Testing)	10	24 - 29 Oct 2022	Preparing Project Development - Delivery of Sprint-1
		11	31 Oct - 5 Nov 2022	Preparing Project Development - Delivery of Sprint-2
		12	7 - 12 Nov 2022	Preparing Project Development - Delivery of Sprint-3
		13	14 - 19 Nov 2022	Preparing Project Development - Delivery of Sprint-4

TABLE OF CONTENTS

CHAPTER NO	CONTENTS	PAGE NO
	ABSTRACT	
	LIST OF FIGURES	
	LIST OF TABLES	
1	INTRODUCTION	1
	1.1 PROJECT OVERVIEW	
	1.2 PURPOSE	
2	LITERATURE SURVEY	2
	2.1 EXISTING SOLUTION	
	2.2 PROBLEM STATEMENT DEFINITION	
3	IDEATION & PROPOSED SOLUTION	5
	3.1 EMPATHY MAPCANVAS	
	3.2 IDEATION & BRAINSTORMING	
	3.3 PROPOSED SOLUTION	
	3.4 PROBLEM SOLUTION FIT	
4	REQUIREMENT ANALYSIS	10
	4.1 FUNCTIONAL REQUIREMENT	
	4.2 NON-FUNCTIONAL REQUIREMENT	
5	PROJECT DESIGN	13
	5.1 DATA FLOW DIAGRAMS	
	5.2 SOLUTION & TECHNICAL ARCHITECTURE	

6	PROJECT PLANNING & SCHEDULING	16
	6.1 SPRINT PLANNING & ESTIMATION	
	6.2 SPRINT DELIVERY SCHEDULE	
7	CODING & SOLUTIONING	18
	7.1 FEATURE	
8	TESTING AND RESULTS	
	8.1 TEST CASES AND RESULTS	21
9	ADVANTAGES & DISADVANTAGES	25
10	CONCLUSION	26
11	FUTURE SCOPE	27

SOURCE CODE

GITHUB & PROJECT DEMO LINK

REFERENCES

ABSTRACT

In a very real way, how well a business manages its inventory can have a significant impact upon its overall success. Any venture that handles stock will need a system to accurately track and control it. Each company will manage stock in their own unique way, depending on the nature and size of their business. Inventory Management is the process of identifying, managing and maintaining the hardware and software assets of an organization. The inventory management system has to be developed in such a way to allow users to add an inventory, delete an inventory, enter inventory quantity and other details, update inventory status and more. This inventory management system scans the Windows desktops and servers in the network periodically to collect the hardware and software details and stores them in the database. Without this information, one may end up with excess stock, eroding your bottom line, or with insufficient stock to meet customer demand. If the software is cloud based, it'll be able to sync it up with your other cloud applications. Thus, this dedicated inventory management software is developed specifically to help one to track and control stock.

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	EMPATHY MAP	4
3.2	IDEATION AND BRAINSTORMING	5
5.1	DATA FLOW DIAGRAM	14
5.2	SOLUTION ARCHITECTURE	15
7.1	IBM CLOUD	18
8.1	REGISTRATION PAGE	21
8.2	LOGIN PAGE	21
8.3	INVENTORY PAGE	22
8.4	CHANGE PASSWORD PAGE	22
8.5	ADD INVENTORY PAGE	22
8.6	EDIT INVENTORY PAGE	23
8.7	REMOVE INVENTORY PAGE	23
8.8	VIEW INVENTORY PAGE	23
8.9	SET THRESHOLD PAGE	24

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
3.1	PROPOSED SOLUTION	7
3.2	PROBLEM SOLUTION FIT	8
4.1	FUNCTIONAL REQUIREMENTS	10
4.2	NON-FUNCTIONAL REQUIREMENT	11
6.1	SPRINT PLANNING AND ESTIMATION	16
6.2	SPRINT DELIVERY PLAN	17

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

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

Inventory management facilitates the smooth functioning of your business and enhances sales, promotes cost-effectiveness, and improves customer experience. Listed below are some of the reasons why businesses need inventory management:

- Managing Finances
- Tracking Inventory
- Avoiding late deliveries
- Managing time and effort
- Predicting future sales
- Enhancing customer loyalty

1.2 PURPOSE

The objective of the project is to create an application that help retailers to track and manage stocks which results in lower costs and a better understanding of sales pattern. Retail inventory management is the process of ensuring retailers meet customer demand without running out of stock or carrying excess supply. By creating an application, retailers can log in to it and can update inventory details, also users will be able to add new stock by submitting essential details related to the stock. Retailers can also 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.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SOLUTION

A systematic literature review was carried out to determine the main trends and indicators of inventory management in Small and Medium-sized Enterprises (SMEs). This focus specifically on the retail sector. In this context, this article aims to analyze and present an extensive literature concerning inventory management, containing multiple definitions and fundamental concepts for the retail sector. The primary outcomes of this study are the leading inventory management systems and models, the Key Performance Indicators (KPIs) for their correct management, and the benefits and challenges for choosing or adopting an efficient inventory control and management system. Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply. In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage their products' stocks. 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.

References:

- A secure and efficient inventory management system for disasters Published by Elsevier Ltd., October 2011, An efficient humanitarian inventory control model and emergency logistics system that plays a crucial role in maintaining a reliable flow of vital supplies to the victims located in the shelters, minimizing the impacts of the unforeseen disruptions that can occur.
- A Material Management in Construction Project Using Inventory Management System Authored by M. Ashika, November 2019, Prepare a scheme of material management in the construction industry for building projects also conducting a survey of the industry and determine the various format for construction material management.

- An IoT Quality Global Enterprise Inventory Management Model for Automation and Demand Forecasting Based on Cloud Authored by Athul Jayaram, December 2017, Industrial Internet of Things (IIoT) collects useful data from machines and sensors which can be used for demand forecasting of the enterprise and automation. The proposed Inventory Management Model can be used for automation and demand forecasting of inventories. Effects of yield and lead-time uncertainty on retailer-managed and vendor-managed inventory management.
- iVentRetail
Reference Link: <https://ivend.com/retail-inventory-management-software/>
- Zoho Inventory Reference Link: <https://www.zoho.com/in/inventory/>
Inventory management for retail companies: A literature review and current trends
Reference Link:
https://www.researchgate.net/publication/352235223_Inventory_management_for_retail_companies_A_literature_review_and_current_trends
- Development of inventory management system
Reference Link: <https://ieeexplore.ieee.org/document/5478077>

2.2 PROBLEM STATEMENT DEFINITION

Common retailers who run their business with large scale or small scale stocks. It is crucial for an organization today to understand its inventory to achieve both efficient and fast operations, that too, at an affordable cost. Lack of the right inventory at the right time can mean back orders, excess inventory, etc. These drive up costs. Late delivery due to stock-outs is bound to give you a bad reputation. Inaccurate calculations of stock and price. Therefore considering the economic crisis of the retailers and to reduce the manpower efficiently while handling data, it is very important to have a best inventory management system for retailers.

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative visualization used to express clearly what one knows about a particular type of user. It externalizes knowledge about users in order to create a shared understanding of user needs, and aid in decision making.

Empathy maps are split into 4 quadrants (Says, Thinks, Does, and Feels), with the user in the middle. Empathy maps provide a glance into who a user is as a whole. The **Says** quadrant contains what the user says or what he needs. The **Thinks** quadrant captures what the user is thinking throughout the experience. The **Does** quadrant encloses the actions the user takes. The **Feels** quadrant is the user's emotional state.

The empathy map for Inventory management system for retailers is shown in Fig 3.1

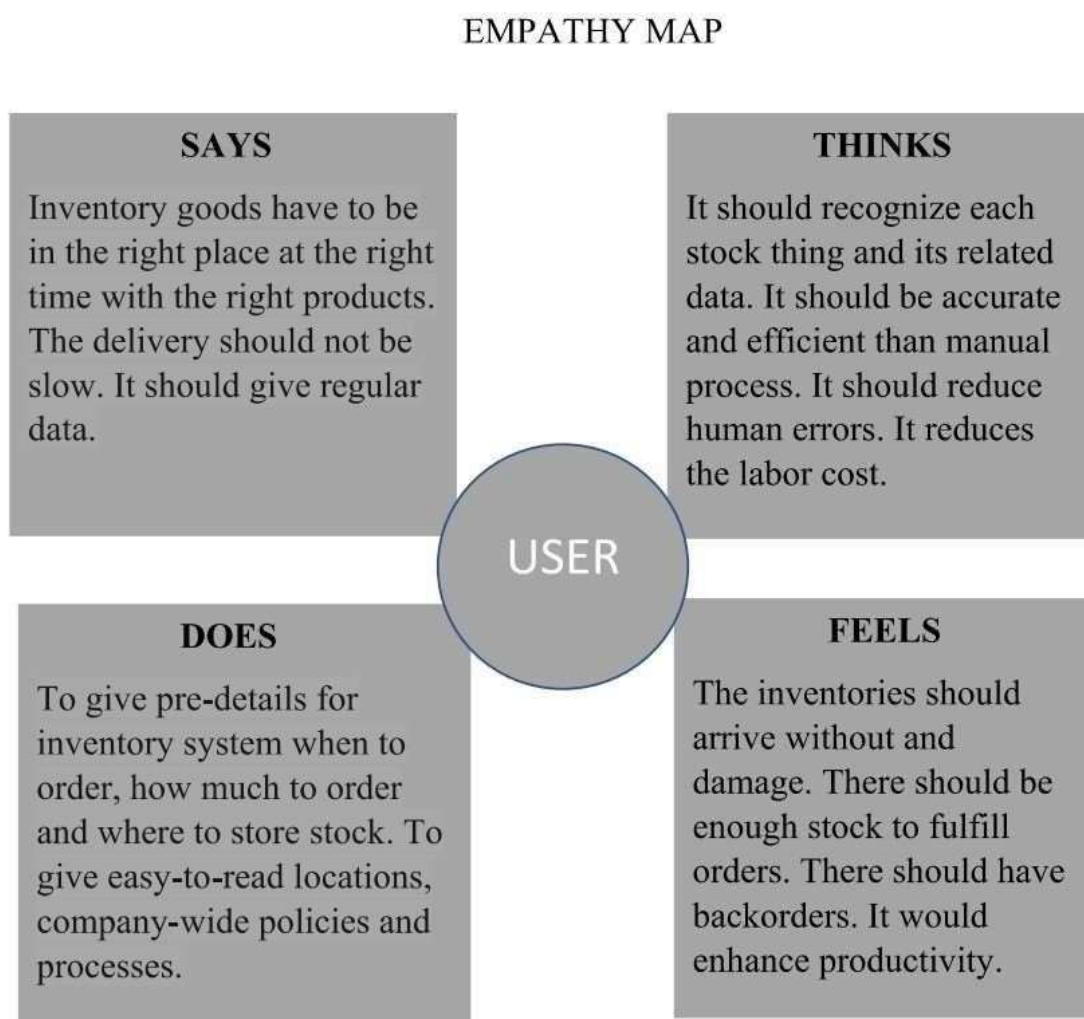


Fig 3.1 Empathy map

3.2 IDEATION AND BRAINSTORMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity. Both brainstorming and ideation are processes invented to create new valuable ideas, perspectives, concepts and insights, and both are methods for envisioning new frameworks and systemic problem solving.

The Ideation chart for Inventory management system for retailers is shown in Table 3.2.



Fig 3.2 Ideation and Brainstorming

3.3 PROPOSED SOLUTION

The proposed solution for Inventory management system for retailers is shown in table 3.3

S.No.	Parameter	Description
❖	Problem Statement (Problem to be solved)	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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.
❖	Scalability of the Solution	<ul style="list-style-type: none"> 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.

❖	Novelty / Uniqueness	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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)	<ul style="list-style-type: none"> • Hereby we can provide a robust and most reliable inventory management system by using: <ol style="list-style-type: none"> 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.

Table 3.1 Proposed Solution

3.4 PROBLEM SOLUTION FIT

The Problem solution fit simply means that one have found a problem with the customer and that the solution one have realized for it actually solves the customers problem. The problem solution fit is an important step towards the Product-Market Fit. The structure of problem solution fit is given below.

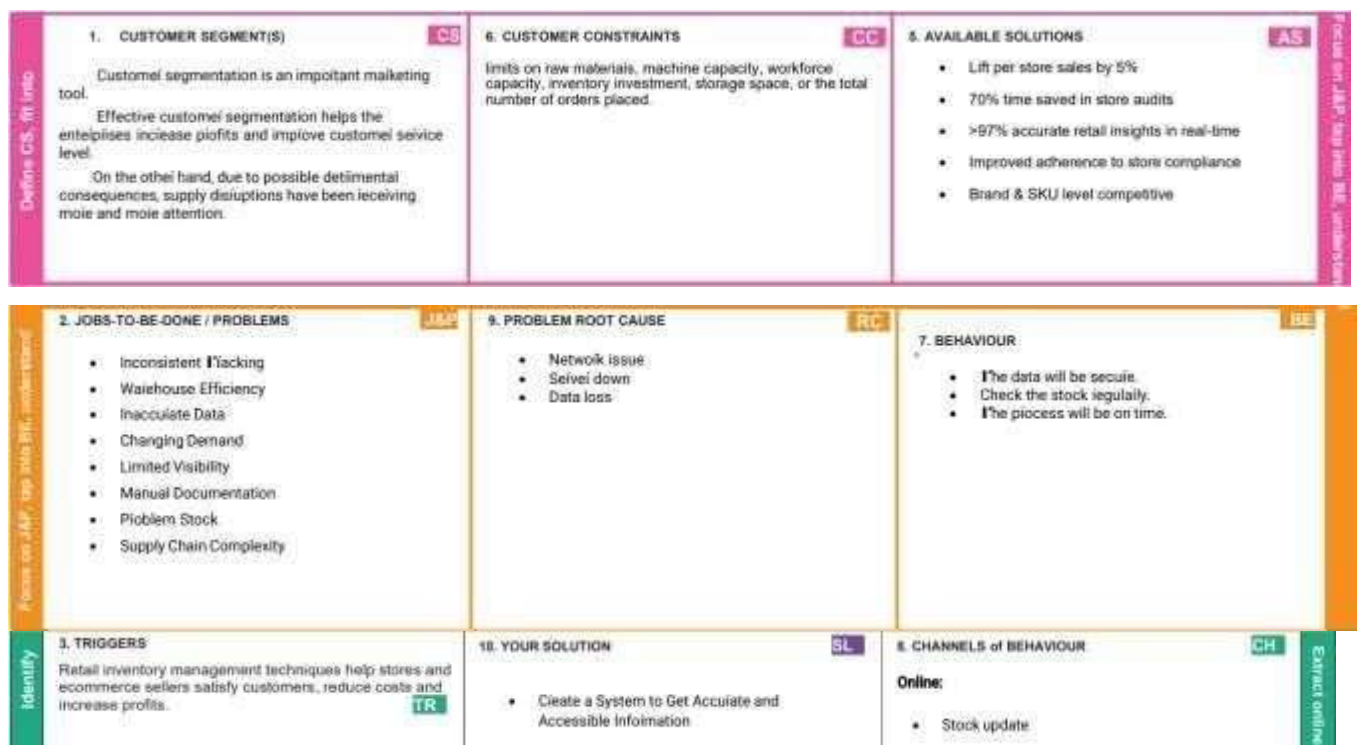
Customer state fit: To make sure one understands the target group, their limitations and their currently available solutions, against which one is going to compete.

Problem-Behavior fit: To help one to identify the most urgent and frequent problems, understand the real reasons behind them and see which behavior supports it.

Communication-Channel fit: To help one to sharpen the communication with strong triggers, emotional messaging and reaching customers via the right channels.

Solution guess: Translate all the validated data one have gathered into a solution that fits the customer state and his/her limitations, solves a real problem and taps into the common behavior of the target group.

The problem solution fit for Inventory management system for retailers is shown in Fig 3.4



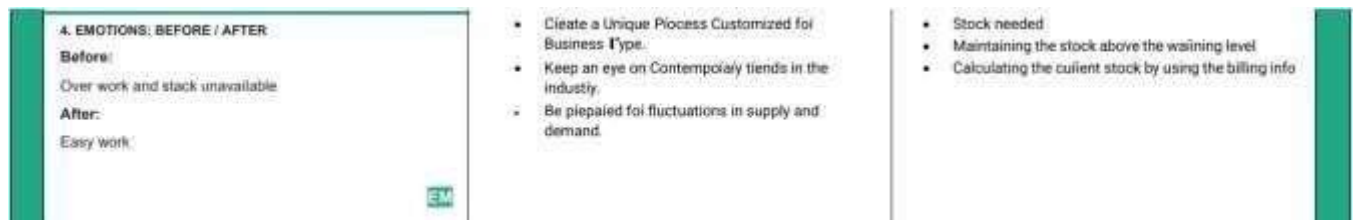


Fig 3.2 Problem Solution Fit

CHAPTER 4

REQUIREMENT ANALYSIS

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

4.1 FUNCTIONAL REQUIREMENTS

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. Following are the functional requirements of the proposed solution shown in table 4.1.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Email
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Login	Log into the application by entering the Email and Password
FR-4	Dashboard	View the products availability
FR-5	Add items to cart	Users they wish to buy products, they can add it to the cart.
FR-6	Stock Update	If the desired product is unavailable, they can update the products into the list for buying products.

Table 4.1 Functional Requirements for the cloud Based Inventory management System

4.2 NON-FUNCTIONAL REQUIREMENTS

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. Following are the non-functional requirements of the proposed solution shown in table 4.2.

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	While usability determines how effective implementing an inventory tracking system is in your business. If it takes hours for your staff to learn the ins and outs of the software, then it's probably not worthbuying.
NFR-2	Security	The process of ensuring the safety and optimum management control of stored goods.It is of central importance for optimum warehouse management because the performance of a company stands or falls with the safety and efficiency of awarehouse.
NFR-3	Reliability	Relying on manual inventory counts to know what you have will only guarantee high inefficiencies and a loss of customers.
NFR-4	Performance	Creating systems to log products, receive them into inventory, track changes when sales occur, manage the flow of goods from purchasing to final sale and check stock counts.
NFR-5	Availability	Whether a specific item is available for customer orders. Additional information provided by retailers may include the quantity available.

NFR-6	Scalability	They should use an automated inventory management system for inventory tracking. This will make your business much more scalable so that you can continue building consistent growth and take advantage of increased sales.
-------	--------------------	---

Table 4.2 Non-Functional Requirements of cloud-based Inventory management System

CHAPTER 5

PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That’s why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems. There are four main elements of a DFD — external entity, process, data store, and data flow.

External entity

An external entity, which are also known as terminators, sources, sinks, or actors, are an outside system or process that sends or receives data to and from the diagrammed system. They’re either the sources or destinations of information, so they’re usually placed on the diagram’s edges. External entity symbols are similar across models except for Unified, which uses a stick-figure drawing instead of a rectangle, circle, or square.

Process

Process is a procedure that manipulates the data and its flow by taking incoming data, changing it, and producing an output with it. A process can do this by performing computations and using logic to sort the data, or change its flow of direction. Processes usually start from the top left of the DFD and finish on the bottom right of the diagram.

Data store

Data stores hold information for later use, like a file of documents that's waiting to be processed. Data inputs flow through a process and then through a data store while data outputs flow out of a data store and then through a process.

Data flow

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.

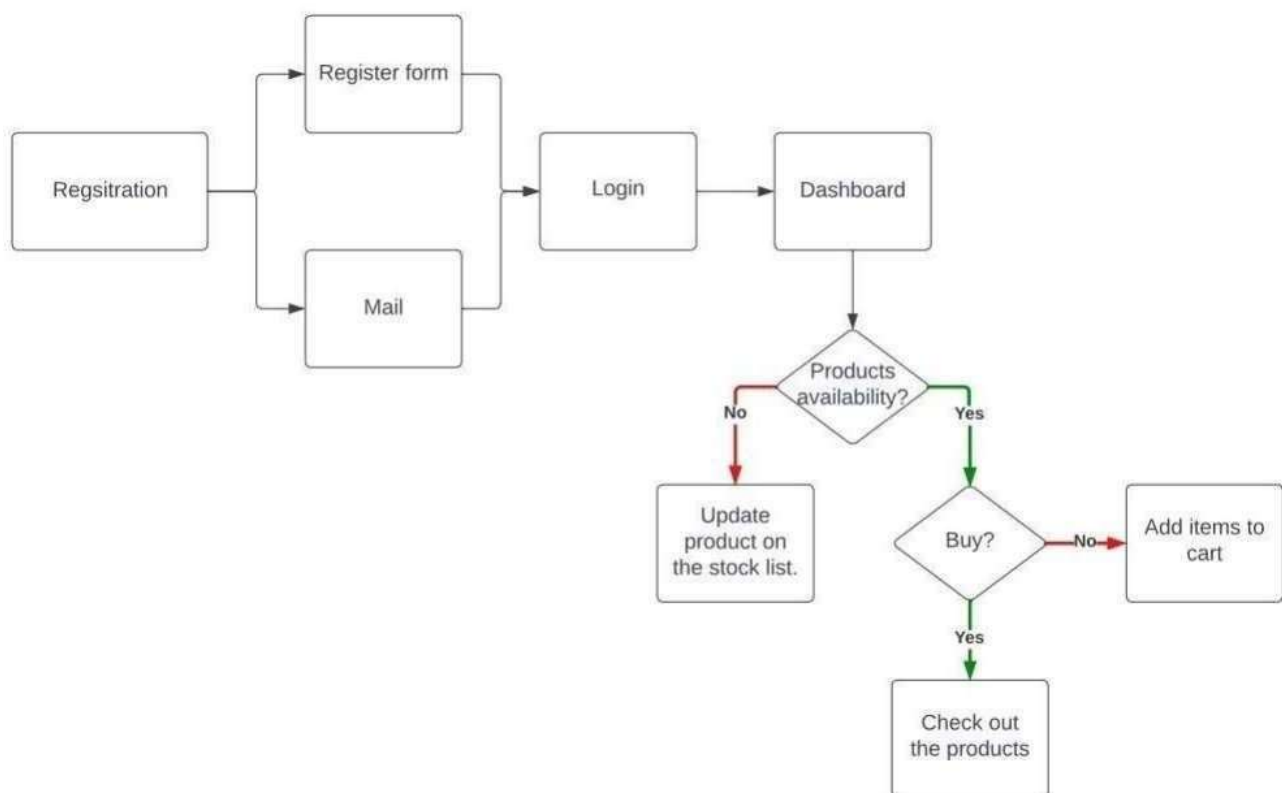


Fig 5.1: Data Flow Diagram for Inventory Management System for Retailers

5.2 SOLUTION AND TECHNICAL ARCHITECHTURE

Solution architecture is a complex process with many sub-processes – that bridges the gap between business problems and technology solutions. A web application is designed to address the mentioned issues with the following functionalities: The web application will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. Retailers can add products since it consumes less amount of time

than manual entry. 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. An alert is sent automatically by the inventory management system if the stock left count reaches a threshold value and as soon as the alert is received, the stocks required are ordered and as a result pausing of sale is avoided. A simple E-commerce web page is developed to check the stock management. We can order the new products from a particular retailer.

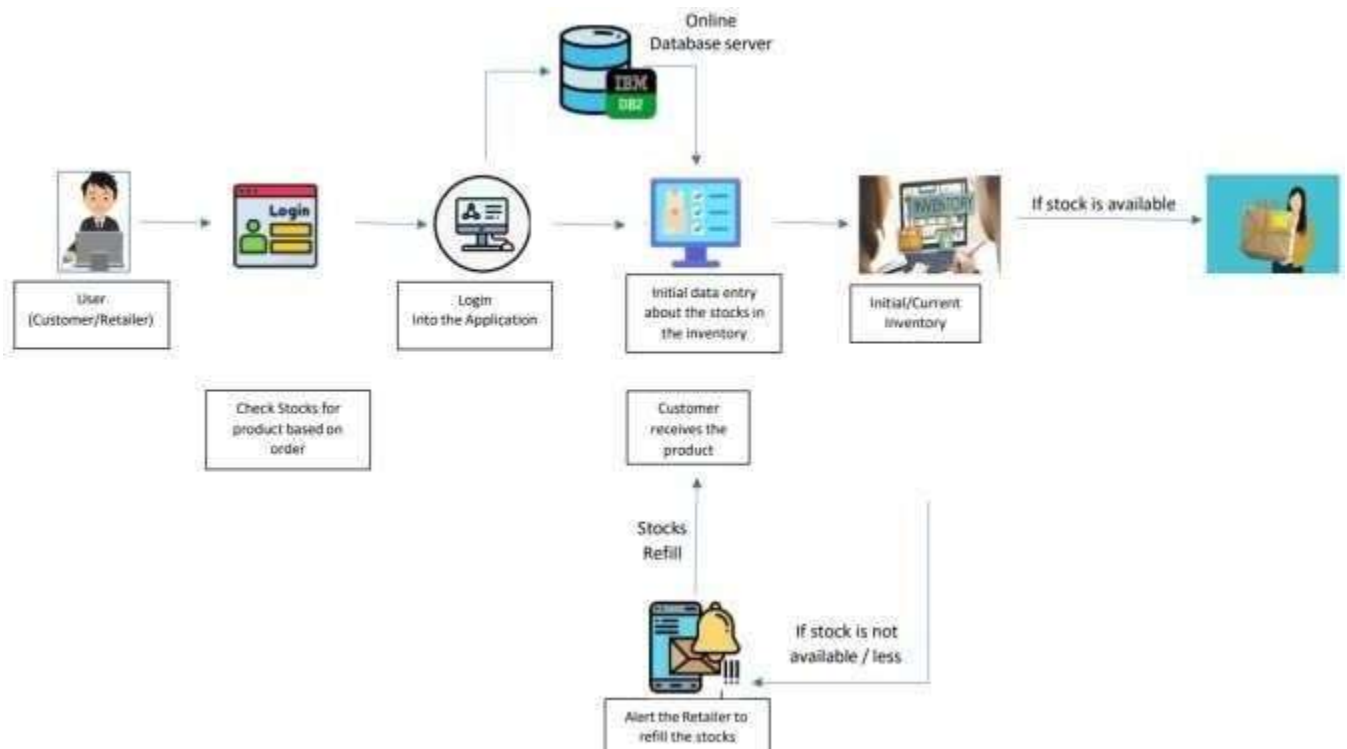


Fig 5.2: Solution architecture of Inventory Management for retailers

CHAPTER 6

PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Sprint planning is an event in scrum that kicks off the sprint. The purpose of sprint planning is to define what can be delivered in the sprint and how that work will be achieved. Sprint planning is done in collaboration with the whole scrum team. The sprint is a set period of time where all the work is done. However, before leap into action it is necessary to set up the sprint. It needs to decide on how long the time box is going to be, the sprint goal, and where it is going to start. The sprint planning session kicks off the sprint by setting the agenda and focus. Sprint Planning and estimation for Inventory Management system for Retailers is shown in table 6.1.

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	4
Sprint-1		USN-2	As a user, I can register for the application through E-mail	1	Medium	4
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	2	Medium	4
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High	4
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	4	High	4
Sprint-2	Add items to cart	USN-6	As a user, I can add the products I wish to buy to the carts.	5	Medium	4
Sprint-3	Stock Update	USN-7	As a user, I can add products which are not available in the dashboard to the stock list.	5	Medium	4

Sprint-4	Request to Customer Care	USN-8	As a user, I can contact the Customer Care Executive and request any services I want from the customer care.	5	Low	4
Sprint-4	Contact Administrator	USN-9	I can be able to report any difficulties I experience as areport	5	Medium	4

Table 6.1: Sprint Planning and estimation for Inventory Management system for Retailers

6.1 SPRINT DELIVERY SCHEDULE

The sprint delivery plan is scheduled accordingly as shown in the below table 6.2 which consists of the sprints with respective to their duration, sprint start and end date and the releasing data. Sprint Planning done for Inventory Management system for Retailers is shown in table 6.2.

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date(Planned)	Story Points Completed (Planned End Date)	Sprint Release Date (Actual)
Sprint-1	7	6 Days	24 Oct 2022	29 Oct 2022	7	29 Oct 2022
Sprint-2	9	6 Days	31 Oct 2022	05 Nov 2022	9	05 Nov 2022
Sprint-3	5	6 Days	07 Nov 2022	12 Nov 2022	5	12 Nov 2022
Sprint-4	10	6 Days	14 Nov 2022	19 Nov 2022	10	19 Nov 2022

Table 6.2: Sprint Planning done for Inventory Management system for Retailers

CHAPTER 7

CODING & SOLUTIONING

IBM Cloud

The IBM Cloud platform combines platform as a service (PaaS) with infrastructure as a service (IaaS) to provide an integrated experience as shown in fig 7.1. The platform scales and supports both small development teams and organizations, and large enterprise businesses. Globally deployed across data centers around the world, the solution you build on IBM Cloud spins up fast and performs reliably in a tested and supported environment you can trust!

IBM Cloud provides solutions that enable higher levels of compliance, security, and management, with proven architecture patterns and methods for rapid delivery for running mission-critical workloads.

IBM Cloud Platform

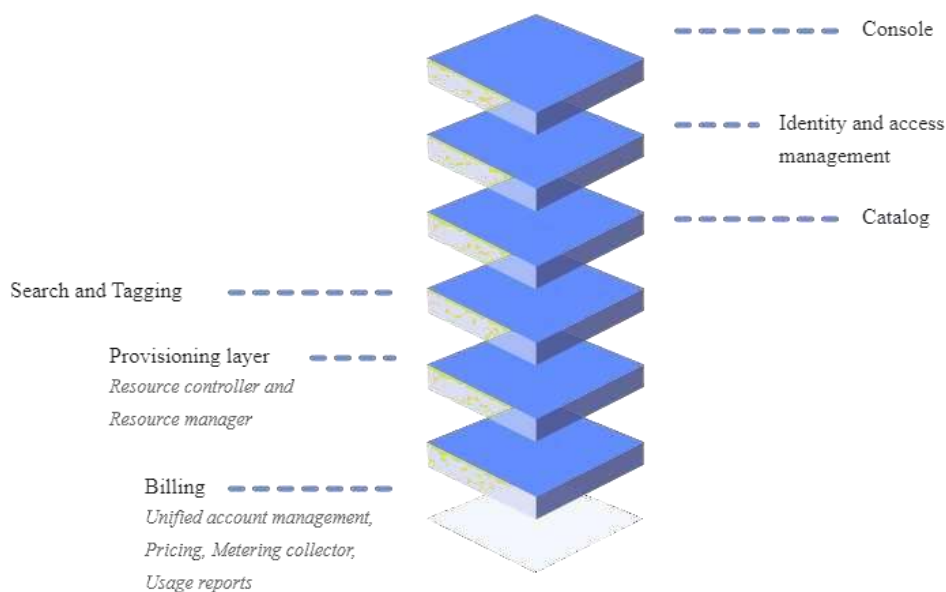


Fig 7.1 IBM Cloud

Flask framework

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-

party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

IBM DB2 Module

Module features allow you to

- Extend schema support by allowing you to group together, in a named set, a collection of related data type definitions, database object definitions and other logic elements including:
 - SQL procedures
 - A module initialization procedure for implicit execution upon module initialization
 - User-defined data type definitions including: distinct type, array type, associative array type, row type, and cursor type
- Define a namespace such that objects defined within the module can refer to other objects defined in the module without providing an explicit qualifier.
- Add object definitions that are private to the module. These objects can only be referenced by other objects within the module.
- Add object definitions that are published. Published objects can be referenced from within the module or from outside of the module.
- Define published prototypes of routines without routine-bodies in modules and later implement the routine-bodies using the routine prototype.
- Initialize the module by executing the module initialization procedure for the module. This procedure can include SQL statements, SQL PL statements, and can be used to set default values for global variables or to open cursors.
- Reference objects defined in the module from within the module and from outside of the module by using the module name as a qualifier (2-part name support) or a combination of the module name and schema name as qualifiers (3-part name support).
- Drop objects defined within the module.
- Drop the module.
- Manage who can reference objects in a module by allowing you to grant and revoke the EXECUTE privilege for the module.

Docker CLI

The Docker client enables users to interact with Docker. The Docker client can reside on the same host as the daemon or connect to a daemon on a remote host. A docker client can communicate with more than one daemon. The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon.

The main purpose of the Docker Client is to provide a means to direct the pull of images from a registry and to have it run on a Docker host. Common commands issued by a client are:

- docker build
- docker pull
- docker run

IBM cloud CLI

IBM Cloud CLI provides full management of your IBM Cloud account via command line. Some installation steps described along this guide may need the IBM Cloud Command Line Interface (CLI) available to be performed.

SendGrid API

SendGrid's web API allows users to pull information about their email program without having to actually log on to SendGrid.com. Users can pull lists, statistics, and even email reports.

In addition to this, users can send email via the web API without using traditional SMTP.

Kubernetes

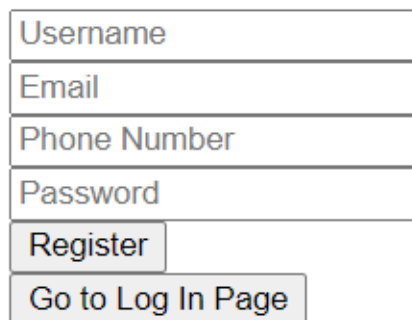
Kubernetes is an open-source Container Management tool which automates container deployment, container scaling, and descaling and container load balancing (also called as container orchestration tool). It is written in Golang and has a huge community because it was first developed by Google and later donated to CNCF (Cloud Native Computing Foundation). Kubernetes can group 'n' number of containers into one logical unit for managing and deploying them easily. It works brilliantly with all cloud vendors i.e. public, hybrid and on- premises. Kubernetes is an open-source platform that manages Docker containers in the form of a cluster. Along with the automated deployment and scaling of containers, it provides healing by automatically restarting failed containers and rescheduling them when their hosts die. This capability improves the application's availability.

CHAPTER 8

TESTING AND RESULTS

This Chapter presents the results of Inventory Management System for Retailers.

Registration Page



Username
Email
Phone Number
Password
Register
Go to Log In Page

Fig 8.1: Registration page for Inventory Management System for Retailers

Login Page



Username
Password
Log in

[Want to register?](#)

Fig 8.2: Login Page for Inventory Management System for Retailers

LOG OUT

Inventory

Edit Inventory

Remove

Send Alert Mail

View Inventory

Set Threshold

Add stock

Change password

Fig 8.3: Inventory Page for Inventory Management system for Retailers

[LOG OUT](#)

Change Password

CURRENT PASSWORD

Current Password

NEW PASSWORD


New Password

RE-ENTER NEW PASSWORD

Re-Enter New Password

Login

Fig 8.4: Change password page for Inventory Management system for Retailers



LOG OUT

Add Stocks to Inventory

PRODUCT ID

ITEM NAME

QUANTITY

THRESHOLD

UNIT

Add stock

Fig 8.5: Add Inventory Page for Inventory Management system for Retailers




LOGOUT

Edit Stocks

Item Name	Quantity	
<pre>{%for dicts in params%} {%endfor%}</pre>		
<input type="text" value="{{dicts['Item']}}"/>	<input type="text" value="{{dicts['Quantity']}}"/>	<input type="text" value="{{dicts['Unit']}}"/> <input type="button" value="Submit"/>

Fig 8.6: Edit Inventory Page for Inventory Management system for Retailers



LOGOUT

Remove stocks from inventory

SELECT PRODUCT

Fig 8.7: Remove Inventory Page for Inventory Management system for Retailers



LOGOUT

Available Stocks

Items	Quantity
<pre>{%for dicts in params%} {%if dicts["Quantity"]>=dicts["Treshold"]%} {%else %} {%endif %} {%endfor%}</pre>	
<input type="text" value="{{dicts['Item']}}"/>	<input type="text" value="{{dicts['Quantity']}}"/>
<input type="text" value="{{dicts['Unit']}}"/>	

Fig 8.8: View Inventory Page for Inventory Management system for Retailers



LOGOUT

Set Threshold

Item Name	Threshold
{%for dicts in params%} {%endfor%}	
{dicts['Item']}	{dicts['Threshc']}
<input type="button" value="Submit"/>	

Fig 8.9: Set Threshold Inventory Page for Inventory Management system for Retailers

CHAPTER 9

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

The inventory management system for retailers is a web-based application. A dashboard is given to retailers where they can able to update, manage and create a product and assign the number of quantities for the products. Whenever a product goes out of stock the inventory management system for retailers alerts the retailers through email stating product goes out of stocks. Thus, inventory management system helps retailers to increase the profit and reduce the risk of holding large quantity of a particular stock. The inventory management system helps the retailers to efficiently utilizes the inventory area i.e., where the store all the products

DISADVANTAGES

The inventory management system for retailers helps them in many ways, but it needs a manual way of updating the quantity of the product. The retailers need to create and update the quantity of stock in the web-based application. The inventory management system necessitates manual updating of stock quantities, which adds to their workload.

CHAPTER 10

CONCLUSION

A web-based application is created to manage inventory stocks. Retailers can able to update, create, and manage products in this web application.

The inventory management system gives a mail alert, if a product goes out of stock, stating that a product has gone out of stock.

This allows retailers to increase their profit, reduce the risk of having too much stock, and make better use of their inventory space.

CHAPTER 11

FUTURE SCOPE

The future scope of the web-based inventory management application for retailers includes building a charting system into the application that helps them know the sales performance of a product for different time periods like a day, a week, or a year.

Automation of updating the quantity of stock for each product using technologies like barcodes, QR codes, etc.

Analyze each product's sales performance in relation to key performance indicators to determine when to offer discounts and offers on products with good and bad stock performance.

SOURCE CODE

App.py

```
from flask import Flask,render_template,url_for,request,redirect,session
import pandas as pd
import numpy as np
import re
from flask_session import Session
import ibm_db
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
#initializing session
Session(app)

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b70af05b-76e4-4bca-a1f5-
23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32716;Security=SSL;
SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=nld81217;PWD=xQrgrKIM3oCr2bVd
","","")

@app.route('/')
def home():
    return render_template("/register.html")

@app.route('/user_home')
def user_home():
    return render_template("/inv_index.html")
```

```

@app.route('/login_val',methods=["POST"])
def login_val():

    uname = request.form.get("username")
    password = request.form.get("password")

    stmt = ibm_db.exec_immediate(conn, "SELECT * FROM CREDTABLE WHERE
USERNAME='"+uname+"'")
    result = ibm_db.fetch_both(stmt)
    print(result)
    if result and result["PASSWORD"] == password:
        session['user'] = uname
        session['email'] = result["EMAIL"]
        return redirect(url_for('user_home'))

    return redirect(url_for('login'))

@app.route('/register',methods=["POST","GET"])
def register():
    username = request.form.get('username')
    password = request.form.get('password')
    email = request.form.get('email')
    phno = request.form.get('phno')

    ##write to db
    stmt = ibm_db.exec_immediate(conn, "INSERT INTO CREDTABLE VALUES ('"+
username+"','"+password+"','"+email+"','"+phno +"')")

    return redirect(url_for('login_page'))

@app.route('/login_page',methods=["POST","GET"])
def login_page():

    return render_template("login.html")

```

```

@app.route('/logout_sess')
def logout_sess():
    if "user" in session:
        session.pop('user',None)
        session.pop('email',None)

    return redirect(url_for("login_page"))

@app.route('/view_inventory' , methods=['GET','POST'])
def view_inventory():
    if "user" not in session:
        return redirect(url_for("home"))

    lt = []
    stmt = ibm_db.exec_immediate(conn, "SELECT * FROM inventory_table where
username='"+session['user']+"'")
    while ibm_db.fetch_row(stmt) != False:
        lt.append( {"prod_id":ibm_db.result(stmt, 0),"Item":ibm_db.result(stmt,
1),"Quantity":ibm_db.result(stmt, 2),"Unit":ibm_db.result(stmt,
3),"Treshold":ibm_db.result(stmt, 5)})

    return render_template("view_inventory.html",params=lt)

#route to visit app stocks in inventory
@app.route("/add_inventory",methods=["POST","GET"])
def add_inventory():
    if "user" not in session:
        return redirect(url_for("home"))
    return render_template("add_inv.html")

#api for adding items into inventory which is called from add_inv.html
@app.route("/add_inv_items" , methods=['GET','POST'])
def add_inv_items():

```

```

if "user" not in session:
    return redirect(url_for("home"))
if request.method == "POST":
    print(request.json)
    print("INSERT INTO INVENTORY_TABLE VALUES ('"+
request.json["Prod_id"]+"','"+request.json["Item"]+"','"+request.json["Quantity"]+"','"+request.j
son["Unit"]+"','"+session['user']+"','"+request.json["Treshold"]+"')")
    print(request)
    #myquery = {"Prod_id":request.json["Prod_id"],"Item":
request.json["Item"],"Quantity":float(
request.json["Quantity"]), "Treshold":float(request.json["Treshold"]), "Unit":request.json["Unit"
]}
    #inv.insert_one(myquery)
    #id = inv.find_one(myquery)["_id"]
    #inv_treshold.insert_one({"Item": request.json["Item"], "Treshold":3, "inv_id":id})

    stmt = ibm_db.exec_immediate(conn, "INSERT INTO INVENTORY_TABLE VALUES
('"+
request.json["Prod_id"]+"','"+request.json["Item"]+"','"+request.json["Quantity"]+"','"+request.j
son["Unit"]+"','"+session['user']+"','"+request.json["Treshold"]+"')")

    print(stmt)
    print("After request")
    return "hello"

#route for removing stocks form inventory
@app.route("/remove_inventory",methods=["GET","POST"])
def remove_inventory():
    if "user" not in session:
        return redirect(url_for("home"))
    lt = []
    stmt = ibm_db.exec_immediate(conn, "SELECT * FROM inventory_table where
username='"+session['user']+"';")
    while ibm_db.fetch_row(stmt) != False:

```

```

        lt.append({"prod_id":ibm_db.result(stmt, 0),"Item":ibm_db.result(stmt, 1)})

    return render_template("remove_inv.html",params=lt)

#api for remove inventory items called from remove_inv.html
@app.route("/remove_inv_items" , methods=['GET','POST'])
def remove_inv_items():
    if "user" not in session:
        return redirect(url_for("home"))
    if request.method == "POST":
        print(request.json)
        print(request)

        stmt = ibm_db.exec_immediate(conn, "delete from inventory_table where
prod_name='"+request.json["Item"]+"'")

        print("After request")
        return "hello"

#route for change password of inventory login
@app.route('/ivcpass' , methods=["POST", "GET"])
def ivcpass():
    if "user" not in session:
        return redirect(url_for("home"))

    return render_template('ivcpass.html')

#api for changing password for inventory which is called form ivcpass.html
@app.route('/ivcpassword',methods=['GET','POST'])
def ivcpassword():
    if "user" not in session:
        return redirect(url_for("home"))
    msg=""

```

```

if request.method=="POST":
    details=request.form
    name=session['user']
    passw=details['icpassw']
    passw1=details['inpassw']
    npass=details['irnpassw']
    if passw1 != npass:
        return render_template('ivcpass.html')
    stmt = ibm_db.exec_immediate(conn, "SELECT USERNAME,PASSWORD FROM
CREDTABLE WHERE USERNAME='"+name+"'")
    result = ibm_db.fetch_both(stmt)
    upass=result["PASSWORD"]
    uname=session['user']
    #search for password and name if name and password in collection allow them to change
password
    if upass==passw:
        myquery = "UPDATE CREDTABLE SET PASSWORD='"+passw1+"' WHERE
USERNAME='"+uname+"';"
        stmt = ibm_db.exec_immediate(conn, myquery)
        msg="Password Updated Sucessfully!!!"
        return render_template('login.html',msg=msg)
    else:
        msg="Invalid Credentials !! Try Again"
        return render_template('ivcpass.html',msg=msg)

#route for editing the inventory in inventory dashboard
@app.route("/edit_inventory", methods=['GET','POST'])
def edit_inventory():
    if "user" not in session:
        return redirect(url_for("home"))
    lt = []
    stmt = ibm_db.exec_immediate(conn, "SELECT * FROM inventory_table where
username='"+session['user']+"'")
    while ibm_db.fetch_row(stmt) != False:

```



```

        lt.append({"_id":ibm_db.result(stmt, 0),"Item":ibm_db.result(stmt,
1),"Quantity":ibm_db.result(stmt, 2),"Unit":ibm_db.result(stmt,
3),"Treshold":ibm_db.result(stmt, 5)})

    return render_template("edit_inventory.html",params=lt)

#api for editing inventory item values called from edit_inventory.html
@app.route("/edit_inv_items", methods=['GET','POST'])
def edit_inv_items():
    if "user" not in session:
        return redirect(url_for("home"))
    if request.method == "POST":

        myquery = "UPDATE INVENTORY_TABLE SET
PROD_NAME='"+request.json["Item"]+"',QTY="+request.json["Quantity"]+" ,PROD_UNIT='
"+request.json["Unit"]+"' WHERE PROD_ID='"+request.json["_id"]+"';"
        stmt = ibm_db.exec_immediate(conn, myquery)

        #myquery = {"_id": ObjectId(request.json["_id"])}
        #newvalues = { "$set": { "Item":request.json["Item"] ,
"Quantity":float(request.json["Quantity"]), "Unit":request.json["Unit"]} }
        #inv.update_one(myquery, newvalues)

    return "Succes"

#route for set treshold
@app.route("/set_treshold", methods=["POST", "GET"])
def set_treshold():
    if "user" not in session:
        return redirect(url_for("home"))

    lt = []

    stmt = ibm_db.exec_immediate(conn, "SELECT * FROM inventory_table where
username='"+session['user']+"'")

```

```

while ibm_db.fetch_row(stmt) != False:
    lt.append({"_id":ibm_db.result(stmt, 0),"Item":ibm_db.result(stmt,
1),"Quantity":ibm_db.result(stmt, 2),"Unit":ibm_db.result(stmt,
3),"Treshold":ibm_db.result(stmt, 5)})

return render_template("set_treshold.html",params=lt)

#api for set Threshold called form set_treshold.html
@app.route("/change_set_treshold" , methods=["POST","GET"])
def change_set_treshold():
    if "user" not in session:
        return redirect(url_for("home"))
    if request.method == "POST":
        print("surasdf")
        print("UPDATE INVENTORY_TABLE SET
THRESHOLD="+request.json["Treshold"]+" WHERE PROD_ID="+request.json["_id"]+";")
        myquery = "UPDATE INVENTORY_TABLE SET
THRESHOLD="+request.json["Treshold"]+" WHERE PROD_ID="+request.json["_id"]+";"
        stmt = ibm_db.exec_immediate(conn, myquery)
        return "hello"

@app.route("/send_mail")
def send_mail():
    if "user" not in session:
        return redirect(url_for("home"))

    username = session["user"]

    query = "select * from inventory_table where username='"+username+" ' and
qty<threshold;"

    lt = []
    stmt = ibm_db.exec_immediate(conn,query)
    while ibm_db.fetch_row(stmt) != False:

```

```
lt.append([ibm_db.result(stmt, 0),ibm_db.result(stmt, 1),str(ibm_db.result(stmt,
2)),ibm_db.result(stmt, 3),str(ibm_db.result(stmt, 5))])
```

```
if not lt:
```

```
    print("No data found")
```

```
    return "No Data Found"
```

```
starter = "<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
table {
```

```
font-family: arial, sans-serif;
```

```
border-collapse: collapse;
```

```
width: 100%;
```

```
}
```

```
td, th {
```

```
border: 1px solid #dddddd;
```

```
text-align: left;
```

```
padding: 8px;
```

```
}
```

```
tr:nth-child(even) {
```

```
background-color: #dddddd;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Inventory Stocks List</h2>
```

```
<table>
```

```
<tr>
```

```
    <th>Product Id</th>
```

```
    <th>Product Name</th>
```

```
    <th>Quantity</th>
```

```

        <th>Unit</th>
        <th>Threshold</th>
    </tr>"""

    mid = ""
    for l in lt:
        mid +=
    "<tr><td>" + l[0] + "</td><td>" + l[1] + "</td><td>" + l[2] + "</td><td>" + l[3] + "</td><td>" + l[4] + "</td></tr>"

    mid += "</table></body></html>"
    message = Mail(
        from_email='works.suryav@gmail.com',
        to_emails=session["email"],
        subject='Inventory Stocks running Low !!',
        html_content=starter+mid)
    try:
        sg =
    SendGridAPIClient("SG.SkGBFae1SXC8qFBY4gx03Q.JZTQDSKd0Rtw3xL2P_dbSrDojB7K
    MbKLa4l1gG7nVpM")
        response = sg.send(message)
        print(response.status_code)
        print(response.body)
        print(response.headers)
    except Exception as e:
        print(e.message)

    return "Mail Sent"

if __name__ == '__main__':
    app.run(debug=True,port=5002)

```

GITHUB LINK:

<https://github.com/IBM-EPBL/IBM-Project-15611-1659601573.git>

DEMO LINK:

<https://drive.google.com/file/d/1rdtHg9eLbWnThxivnhJhg5LzM6kNi39j/view?usp=sharing>

REFERENCES

1. Y. Fan, 2010, "Development of inventory management system," 2nd IEEE International Conference on Information Management and Engineering, 2010, pp. 207-210, DOI: 10.1109/ICIME.2010.5478077.
2. A. Milella, A. Petitti, R. Marani, G. Cicirelli and T. D’orazio, "Towards Intelligent Retail: Automated on-Shelf Availability Estimation Using a Depth Camera," in IEEE Access, vol. 8, pp. 19353-19363, 2020, DOI: 10.1109/ACCESS.2020.2968175.
3. Inventory management for retail companies, “A literature review and current trends”, March 2021, Conference: 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST), DOI:10.1109/ICI2ST51859.2021.00018.