



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



IT - ITeS SSC  
NASSCOM



**INVENTORY MANAGEMENT SYSTEM FOR RETAILERS**

**A PROJECT REPORT**

*Submitted by*

<b>LOKESH IYYAPPAN A</b>	<b>953619104026</b>
<b>ATHIVIGNESHKUMAR N</b>	<b>953619104006</b>
<b>CHANDHRU R</b>	<b>953619104008</b>
<b>PRASATH A</b>	<b>953619104031</b>

<b>Date</b>	<b>21 November 2022</b>
<b>Team ID</b>	<b>PNT2022TMID51018</b>
<b>Project Name</b>	<b>Inventory Management System for Retailers</b>

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**RAMCO INSTITUTE OF TECHNOLOGY, RAJAPALAYAM – 626 117**

**ANNA UNIVERSITY: CHENNAI 600025  
NOVEMBER 2022**

# **RAMCO INSTITUTE OF TECHNOLOGY**

**RAJAPALAYAM– 626117**

**ANNA UNIVERSITY: CHENNAI 600 025**

## **BONAFIDE CERTIFICATE**

Certified that this report **“INVENTORY MANAGEMENT SYSTEM”** is the bonafide work of **LOKESH IYYAPPAN A (953619104026), ATHIVIGNESHKUMAR N (953619104006), CHANDHRU R (953619104008) AND PRASATH A (953619104031)** who carried out **19ECI01 Professional Readiness for Innovation, Employability and Entrepreneurship** project offered by IBM and Anna University, Chennai.

## PROJECT CALENDER

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, DataFlow 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 & Suctioning, 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 ofSprint-2
		12	7 - 12 Nov 2022	Preparing Project Development - Delivery ofSprint-3
		13	14 - 19 Nov 2022	Preparing Project Development Delivery of Sprint-4

## TABLE OF CONTENTS

CHAPTER NO	CONTENTS	PAGE NO
	<b>LIST OF FIGURES</b>	
	<b>LIST OF TABLES</b>	
<b>1</b>	<b>INTRODUCTION</b> <b>1.1 PROJECT OVERVIEW</b> <b>1.2 PURPOSE</b>	<b>1</b>
<b>2</b>	<b>LITERATURE SURVEY</b> <b>2.1 EXISTING SOLUTION</b> <b>2.2 PROBLEM STATEMENT</b> <b>DEFINITION</b>	<b>2</b>
<b>3</b>	<b>IDEATION &amp; PROPOSED SOLUTION</b> <b>3.1 EMPATHY MAP CANVAS</b> <b>3.2 IDEATION AND BRAINSTORMING</b> <b>3.3 PROPOSED SOLUTION</b> <b>3.4 PROBLEM SOLUTION FIT</b>	<b>3</b>
<b>4</b>	<b>REQUIREMENT ANALYSIS</b> <b>4.1 FUNCTIONAL REQUIREMENTS</b> <b>4.2 NON FUNCTIONAL REQUIREMENTS</b>	<b>7</b>
<b>5</b>	<b>PROJECT DESIGN</b> <b>5.1 DATA FLOW DIAGRAMS</b> <b>5.2 SOLUTION AND TECHNICAL</b> <b>ARCHITECTURE</b>	<b>10</b>
<b>6</b>	<b>PROJECT PLANNING &amp; SCHEDULING</b> <b>6.1 SPRINT PLANNING AND</b> <b>ESTIMATION</b> <b>6.2 SPRINT DELIVERY SCHEDULE</b>	<b>13</b>
<b>7</b>	<b>CODING &amp; SOLUTIONING</b> <b>7.1 IBM CLOUD</b> <b>7.2 FLASK FRAMEWORK</b> <b>7.3 IBM DB2 MODULE</b> <b>7.4 DOCKER CLI</b> <b>7.5 IBM CLOUD CLI</b> <b>7.6 SENDGRID API</b> <b>7.7 KUBERNETES</b>	<b>16</b>
<b>8</b>	<b>TESTING AND RESULTS</b>	<b>19</b>
<b>9</b>	<b>PERFORMANCE RESULTS</b> <b>9.1 PERFORMANCE METRICS</b>	<b>22</b>
<b>10</b>	<b>ADVANTAGES &amp; DISADVANTAGES</b> <b>10.1 ADVANTAGES</b> <b>10.2 DISADVANTAGES</b>	<b>23</b>
<b>11</b>	<b>CONCLUSION</b>	<b>24</b>
<b>12</b>	<b>FUTURE SCOPE</b>	<b>25</b>

**SOURCE CODE**

**GITHUB LINK**

## LIST OF FIGURES

<b>FIGURE NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
<b>3.1</b>	<b>EMPATHY MAP</b>	<b>3</b>
<b>3.4</b>	<b>PROBLEM SOLUTION FIT</b>	<b>6</b>
<b>5.1</b>	<b>DATA FLOW DIAGRAM FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>11</b>
<b>5.2</b>	<b>SOLUTION ARCHITECTURE FOR INVENTORY MANAGEMENT FOR RETAILERS</b>	<b>12</b>
<b>7.1</b>	<b>IBM CLOUD PLATFORM</b>	<b>16</b>
<b>8.1</b>	<b>SIGN UP PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>19</b>
<b>8.2</b>	<b>LOGIN PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>19</b>
<b>8.3</b>	<b>DASHBOARD PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>20</b>
<b>8.4</b>	<b>ORDERS PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>20</b>
<b>8.5</b>	<b>PROFILE PAGE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>21</b>
<b>9.1</b>	<b>PERFORMANCE METRICES</b>	<b>22</b>

## LIST OF TABLES

<b>TABLE NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
<b>3.2</b>	<b>IDEATION AND BRAINSTORMING</b>	<b>4</b>
<b>3.3</b>	<b>PROPOSED SOLUTION</b>	<b>5</b>
<b>4.1</b>	<b>FUNCTIONAL REQUIREMENTS FOR THE CLOUD BASED INVENTORY MANAGEMENT SYSTEM</b>	<b>8</b>
<b>4.2</b>	<b>NON-FUNCTIONAL REQUIREMENTS OF CLOUD-BASED INVENTORY MANAGEMENT SYSTEM</b>	<b>9</b>
<b>6.1</b>	<b>SPRINT PLANNING AND ESTIMATION FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>13</b>
<b>6.2</b>	<b>SPRINT PLANNING DONE FOR INVENTORY MANAGEMENT SYSTEM FOR RETAILERS</b>	<b>15</b>

# **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**

Retail inventory management is the process of ensuring retailers meet customer demand without running out of stock or carrying excess supply. 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. 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**

In recent years, the correct management of inventories has become a fundamental pillar for achieving success in enterprises. Unfortunately, studies suggesting the investment and adoption of advanced inventory management and control systems are not easy to find. 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. 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. 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. Findings indicate that SMEs do not invest resources in sophisticated systems; instead, a simple Enterprise Resource Planning (ERP) system or even programs such as Excel or manual inventories are mainly used.

#### **2.2 PROBLEM STATEMENT DEFINITION**

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.



## 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

# Inventory Management System for Retailers

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.

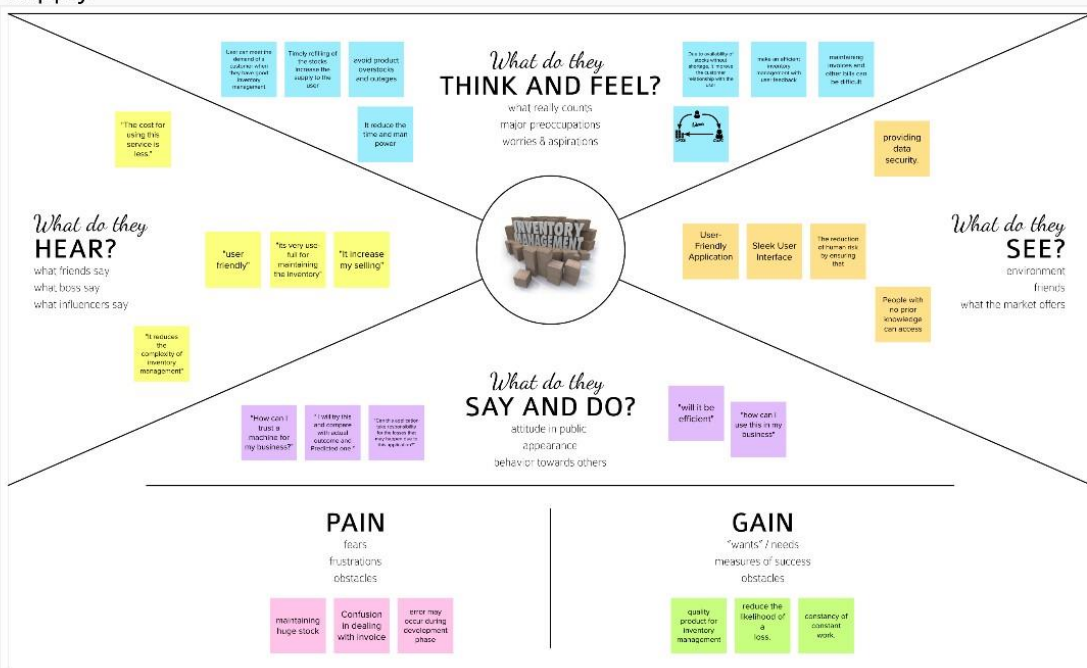
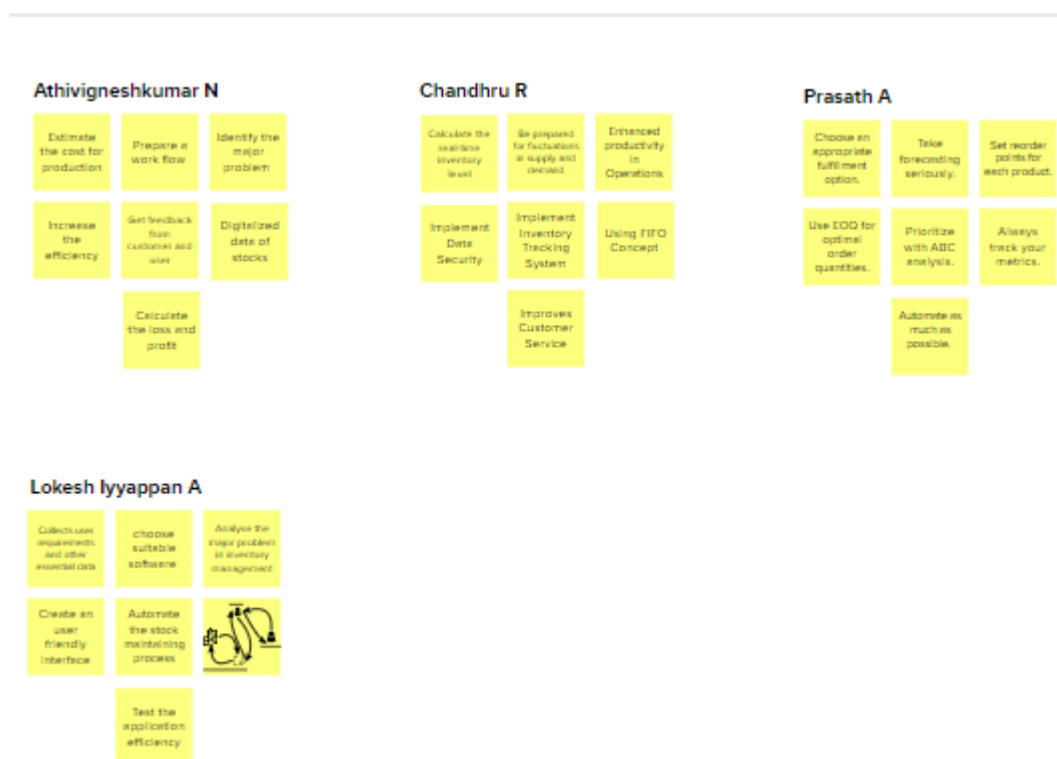


Fig 3.1 Empathy map

### 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.



**Table 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
1.	<b>Problem Statement (Problem to be solved)</b>	Inventory systems, demand is usually uncertain, and the lead-time can also vary. To avoid shortages, managers often maintain a safety stock. In such situations, it is not clear what order quantities and reorder points will minimize expected total inventory cost.
2.	<b>Idea / Solution description</b>	The right inventory management platform can automate processes, improve inventory practices and enhance customer experiences
3.	<b>Novelty / Uniqueness</b>	Track inventory across multiple locations, automatically manage reorder points, forecast demand and plan production and distribution.
4.	<b>Social Impact / Customer Satisfaction</b>	Inventory management helps you manage the customer experience when it comes to product returns. An inventory management system can track important data concerning returned items and giving you the option to maintain additional inventory levels that mirror your return rates
5.	<b>Business Model (Revenue Model)</b>	Balance demand and supply, integrate financial and operational planning, and link high-level strategic plans with mid- and long-term business plans.
6.	<b>Scalability of the Solution</b>	To increase the scalability of your business, you 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. An automated inventory management system will give your business the structure and real-time metrics it needs to remain competitive and achieve growth goals.

**Table 3.3 Proposed Solution**

### 3.4 PROBLEM SOLUTION FIT

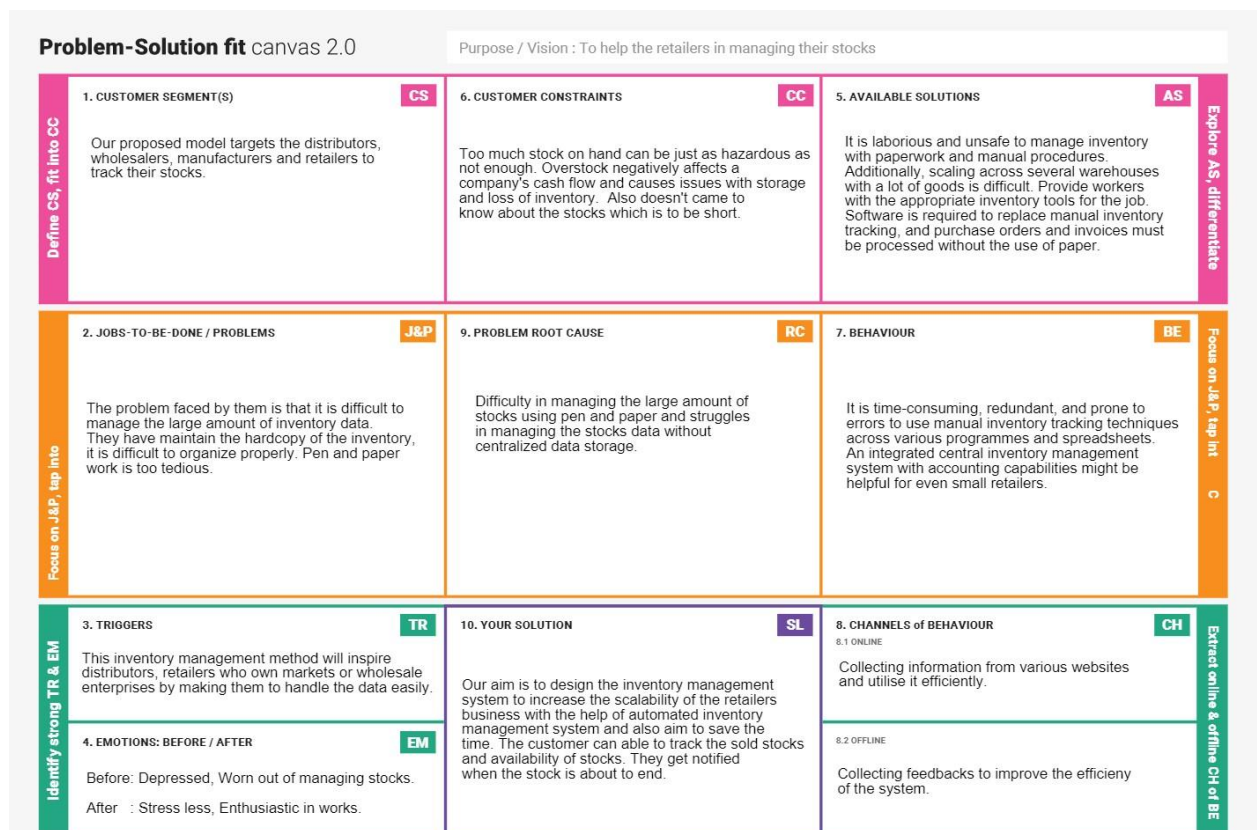
The Problem solution fit simply means that one have found a problem with the customer and that the solution one have realised for it actually solves the customers problem. The problem solution fit is an important step towards the Product-Market Fit. The structure of problem solution fit is given below.

**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 below fig 3.4 shows the problem solution fit



**Fig 3.4 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. The below table 4.1 shows the Functional Requirements for the cloud Based Inventory management System

<b>FR. No.</b>	<b>Functional Requirement (Epic)</b>	<b>Sub Requirement (Story/Sub-Task)</b>
FR-1	User Registration	Registration through registration form.  Registration through One-Tap Google Sign-in.
FR-2	User Authentication and Confirmation	Authentication via Google Authentication.  Confirmation via Email.  Confirmation via OTP.
FR-3	Product management	Quickly produce reports for single or multiple products.  Track information of dead and fast-moving products.  Track information of suppliers and manufacturers of the product.

FR-4	Audit Monitoring	<p>The technique of tracking crucial data is known as audit tracking.</p> <p>Monitor the financial expenses carried out throughout the whole time (from receiving order of the product to delivery of the product).</p>
FR-5	Historical Data	Data of everything should be stored for analytics and forecasting.
FR – 6	CRM (Customer Relationship Management)	<p>Track the customer experience via ratings given by them.</p> <p>Get customer reviews regularly or at least at the time of product delivery to work on customer satisfaction.</p> <p>User-friendly GUI to increase the customer base from only techies to normal people.</p>
FR - 7	Security Policy	<p>User data collected must be as secure as possible.</p> <p>User data must not be misused. They can only be used for user preferred advertising purposes.</p>

**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. The below table 4.2 shows the Non-Functional Requirements for the cloud Based Inventory Management System

<b>FR No.</b>	<b>Non-Functional Requirement</b>	<b>Description</b>
NFR-1	Usability	<p>The UI should be accessible to everybody despite of their diversity in languages.</p> <p>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).</p> <p>.</p>
NFR-2	Security	<p>The security requirements deal with the primary security. Only authorized users can access the system with their credentials.</p> <p>Administrator or the concerned security team should be alerted on any unauthorized access or data breaches so as to rectify it immediately.</p>
NFR-3	Reliability	<p>The software should be able to connect to the database in the event of the server being down due to a hardware or software failure.</p>
		<p>The users must me intimated by the periodic maintenance break of the server so that they will be aware of it.</p>
NFR-4	Performance	<p>Performance of the app should be reliable with high-end servers on which the software is running.</p>
NFR-5	Availability	<p>The software should be available to the users 24/7 with all functionalities working.</p> <p>New module deployment should not impact the availability of existing modules and their functionalities.</p>
NFR-6	Scalability	<p>The whole software deployed must be easily scalable as the customer base increases.</p>

**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.

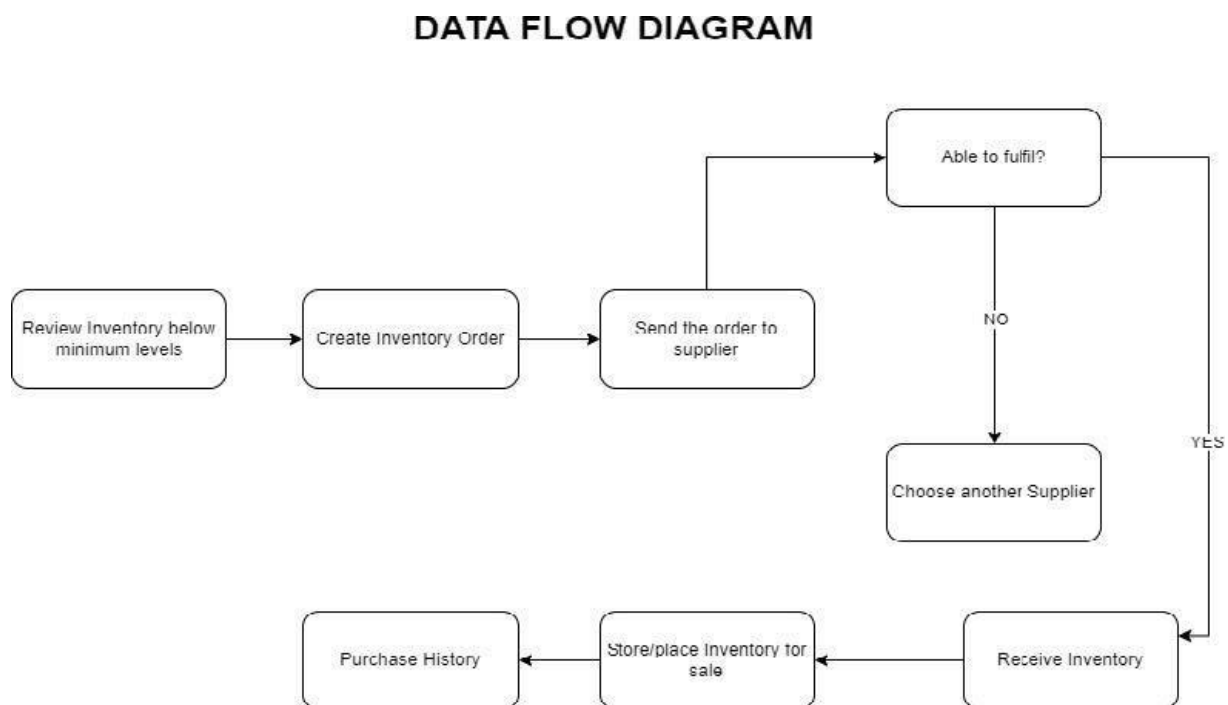


- **Data store**

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**

Data flow is the path the system's information taken from external entities through processes and data stores. With arrows and succinct labels, the DFD can show the direction of the data flow. The below Fig 5.1 shows the Data Flow Diagram for Inventory Management system for retailers



**Fig 5.1: Data Flow Diagram for Inventory Management system for retailers**

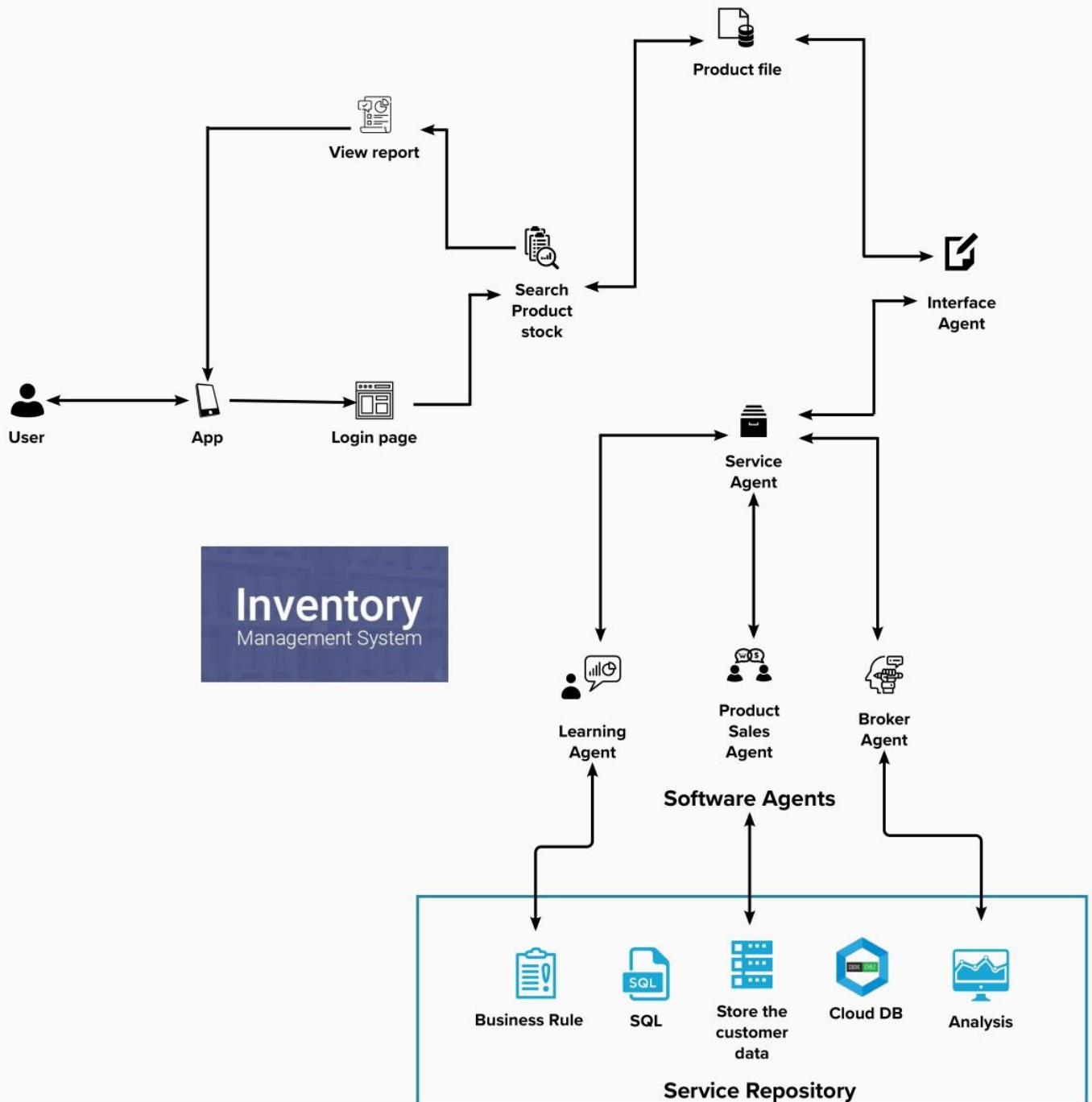
## 5.2 SOLUTION AND TECHNICAL ARCHITECHTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

The below figure 5.2 shows the Solution architecture of Inventory Management for retailers



**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. If done correctly, it also creates an environment where the team is motivated, challenged, and can be successful. The below table 6.1 shows the Sprint Planning and estimation for Inventory Management system for Retailers

Release	Functional Requirement (Epic)	User Story Number	User Story / Task	Score 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.	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-3		USN-3	As a user, I can register for the application through Facebook	2	Low	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-2		USN-4	As a user, I can register for the application through Gmail	2	Medium	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P

Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1	Dashboard	USN-6	As a user, I can track data of sales of products and inventory levels	4	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1	Registration (Customer)	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password.	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1		USN-8	As a user, I will receive confirmation email once I have registered for the application	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-3		USN-9	As a user, I can register for the application through Facebook	2	Low	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-2		USN-10	As a user, I can register for the application through Gmail	2	Medium	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1	Login (Customer)	USN-11	As a user, I can log into the application by entering email & password	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P
Sprint-1	Dashboard (Customer)	USN-12	As a user, I can track data of sales of products and inventory levels	3	High	Chandhru R Lokesh Iyyappan A Athivigneshkumar N Prasad P

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

## 6.2 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.

<b>Sprint</b>	<b>Total Story Points</b>	<b>Duration</b>	<b>Sprint Start Date</b>	<b>Sprint End Date (Planned)</b>	<b>Story Points Completed (as on Planned End Date)</b>	<b>Sprint Release Date (Actual)</b>
Sprint-1	28	11 Days	24 Oct 2022	03 Oct 2022	28	03 Nov 2022
Sprint-2	4	4 Days	04 Nov 2022	07 Nov 2022	4	07 Nov 2022
Sprint-3	4	4 Days	09 Nov 2022	12 Nov 2022	4	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**

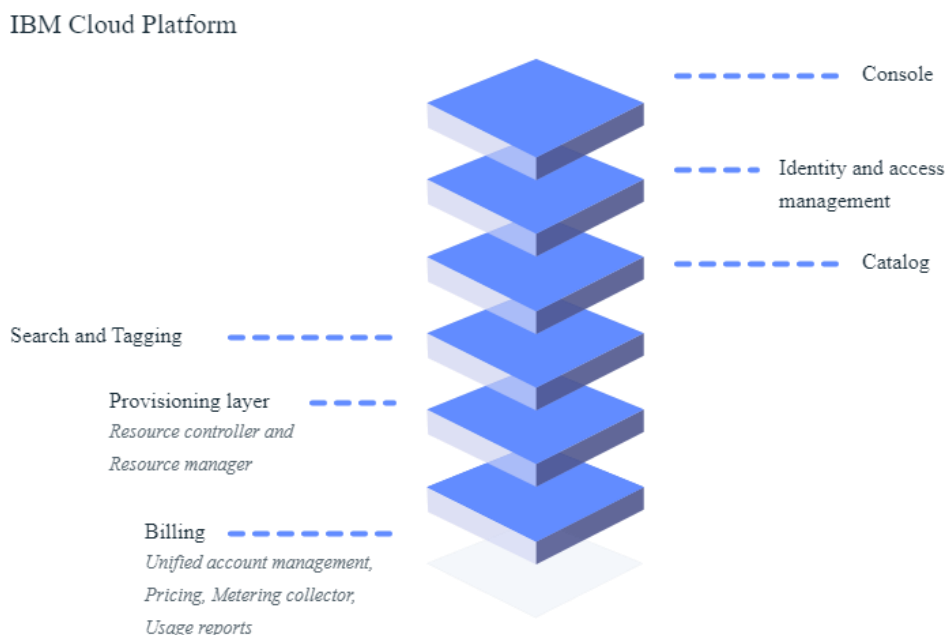
## **CHAPTER 7**

### **CODING & SOLUTIONING**

#### **7.1 IBM Cloud**

The IBM Cloud platform combines platform as a service (PaaS) with infrastructure as a service (IaaS) to provide an integrated experience. 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.



**Fig 7.1: IBM cloud platform**

#### **7.2 Flask framework**

Flask is a micro web framework written in Python. It is classified as a micro framework 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 frameworks related tools.

### 7.3 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.

## **7.4 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

## **7.5 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.

## **7.6 SendGrid API**

Send Grid'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.

## **7.7 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 the results of Inventory Management System for Retailers. The below Fig 8.1 shows the Sign-Up page for Inventory Management System for Retailers

**Inventory Management**

**Welcome User!**

To keep connected with us please login with your personal info

**SIGN IN**

**Create Account**

or use your email for registration

ramco

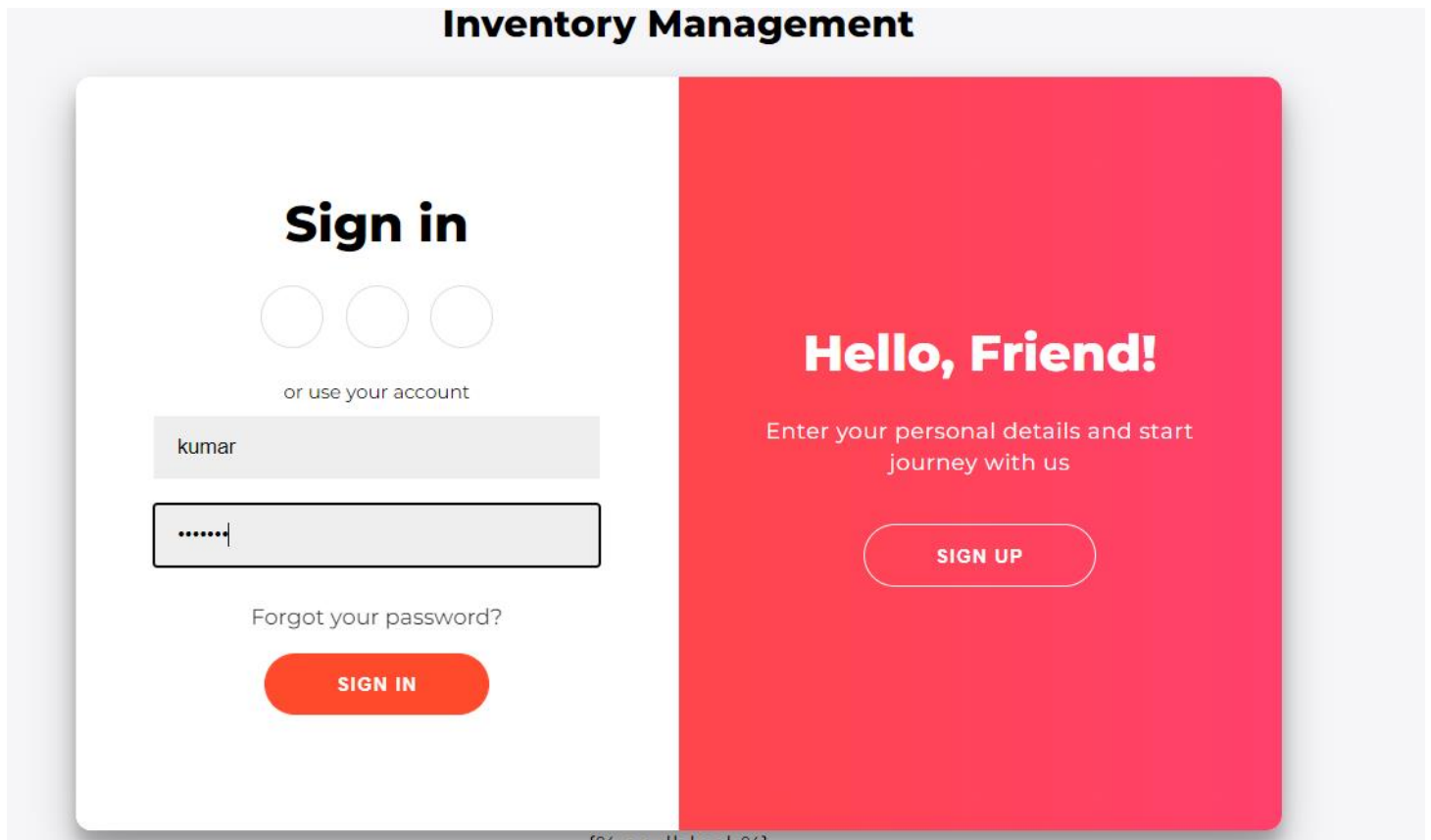
ramco@ritrjpm.ac.in

.....

**SIGN UP**

**Fig 8.1: Sign Up page for Inventory Management System for Retailers**

The below Fig 8.2 shows the Login Page for Inventory Management System for Retailers



The image displays a login page for an 'Inventory Management' system. The page is split into two main sections. The left section, with a white background, is titled 'Sign in' in bold black text. Below the title are three empty circular profile icons. Underneath these icons is the text 'or use your account'. There are two input fields: the first contains the text 'kumar', and the second contains seven dots for a password. Below the password field is a link that says 'Forgot your password?'. At the bottom of this section is a red rounded button labeled 'SIGN IN'. The right section has a solid red background. It features the text 'Hello, Friend!' in large white bold font. Below this is the instruction 'Enter your personal details and start journey with us' in a smaller white font. At the bottom of this section is a white rounded button labeled 'SIGN UP'.

## Inventory Management

### Sign in

or use your account

kumar

.....

[Forgot your password?](#)

**SIGN IN**

### Hello, Friend!

Enter your personal details and start journey with us

**SIGN UP**

**Fig 8.2: Login Page for Inventory Management System for Retailers**

The below Fig 8.3 shows the Dashboard page for Inventory Management system for Retailers

Inventory

Dashboard

Orders

Suppliers

Profile

logout

Dashboard

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

ID	NAME	QUANTITY	PRICE_PER_QUANTITY	TOTAL_PRICE
1	Book	100	10.0	1000.0
2	Laptop	100	10.0	1000.0
3	Table	100	20.0	2000.0
10	dkds	100	10.0	1000.0
5	Pencil	300	5.0	1500.0

Update Stock

Enter Item

Choose a field

Enter Value

Update

Add New Stock

Enter the item

Enter quantity

Enter price

Add Stock

Remove stocks

Enter the item

Remove

29°C  
Haze

ENG  
IN15:58  
17-11-2022

**Fig 8.3: Dashboard page for Inventory Management system for Retailers**

The below Fig 8.4 shows the Orders Page for Inventory Management system for Retailers

**Inventory**

Dashboard

Orders

Suppliers

Profile

Logout

### Orders

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

ID	STOCKS_ID	QUANTITY	DATE	DELIVERY_DATE	PRICE
1	10	10	2022-11-13	2022-11-20	1000.0
8	10	555	2022-11-13	2022-11-20	275.0
3	3	10	2022-11-13	2022-11-20	1000.0

#### Create Order

Enter Stock ID

Enter Quantity

#### Update Order

Enter Order ID

Choose a field

Enter Value

#### Cancel Order

Enter Order ID

**Fig 8.4: Orders Page for Inventory Management system for Retailers**

The below Fig 8.5 shows the Profile Page for Inventory Management system for Retailers

**Inventory**

Dashboard

Orders

Suppliers

Profile

Logout

### Profile

#### User Details

USERNAME : ram  
FIRSTNAME : firstname  
LASTNAME : lastname  
EMAIL : ram@gmail.com

#### Update user details

Choose a field

Enter Value

#### Update Password

Enter Old Password

Enter New Password

Enter Confirm Password

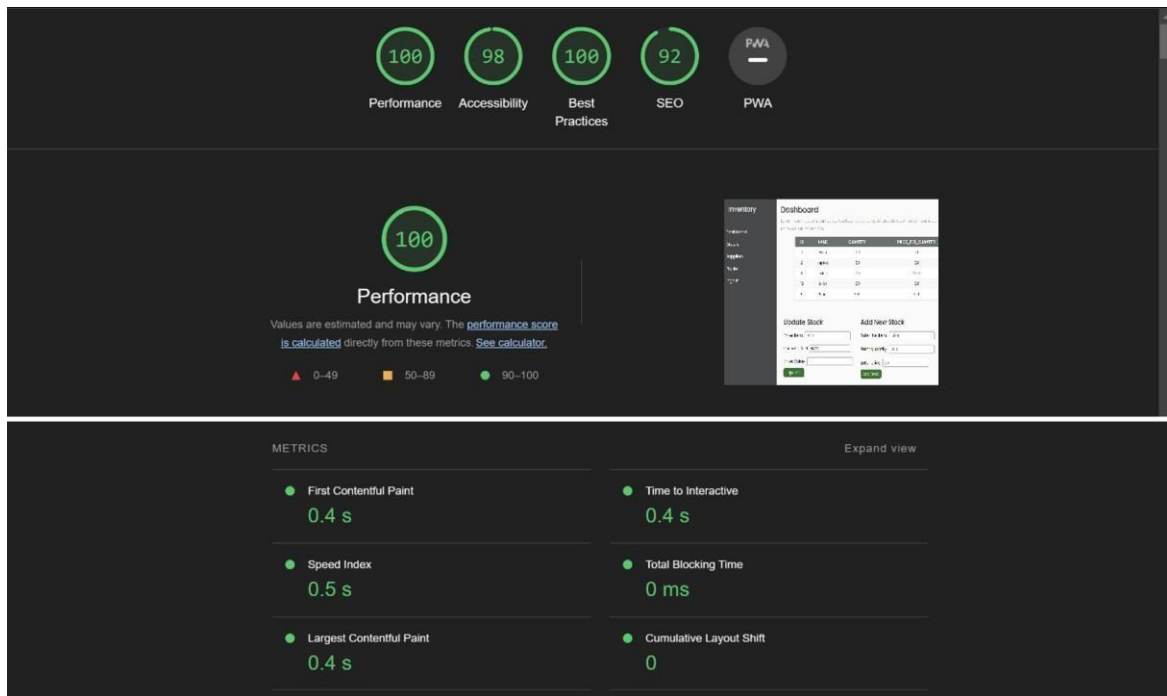
**Fig 8.5: Profile Page for Inventory Management system for Retailers**

## CHAPTER 9

### PERFORMANCE RESULTS

#### 9.1 PERFORMANCE METRICS

Fig 9.1 shows the performance metrics of the flask application using Google Developer Tools.



**Fig 9.1: Performance metrics**

## **CHAPTER 10**

### **ADVANTAGES AND DISADVANTAGES**

#### **10.1 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

#### **10.2 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 11**

### **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 12**

### **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, make_response
from functools import wraps
import re
import ibm_db
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=764264db-9824-4b7c-82df-
40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=32536;SECURITY=SSL
;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=qwq87197;PWD=7TN1X5zgnKSTn9uc",
,)

app = Flask(__name__)
app.secret_key = 'ramcoinstitute'

def rewrite(url):
    view_func, view_args = app.create_url_adapter(request).match(url)
    return app.view_functions[view_func](**view_args)

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if "id" not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/')
def root():
    return render_template('login.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ""

    if request.method == 'POST':
        un = request.form['username']
        pd = request.form['password_1']
```

```

print(un, pd)
sql = "SELECT * FROM Client WHERE username =? AND password=?"
stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt, 1, un)
ibm_db.bind_param(stmt, 2, pd)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)
print(account)
if account:
    session['loggedin'] = True
    session['id'] = account['EMAIL']
    userid = account['EMAIL']
    session['username'] = account['USERNAME']
    msg = 'Logged in successfully !'

    return rewrite('/dashboard')
else:
    msg = 'Incorrect username / password !'
return render_template('login.html', msg=msg)

```

```

@app.route('/signup', methods=['POST', 'GET'])
def signup():
    mg = ""
    if request.method == "POST":
        username = request.form['username']
        email = request.form['email']
        pw = request.form['password']
        sql = 'SELECT * FROM Client WHERE email =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        acnt = ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg = 'Account already exists!!'

        elif not re.match(r'^[^\s@]+@[^\s@]+\.[^\s@]+', email):
            mg = 'Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms = 'name must contain only character and number'
        else:
            insert_sql = 'INSERT INTO Client (USERNAME,EMAIL,PASSWORD) VALUES (?,?,:)?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, username)
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.bind_param(pstmt, 3, pw)
            print(pstmt)
            ibm_db.execute(pstmt)
            mg = 'You have successfully registered click login!'

```

```

        message = Mail(
            from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
            to_emails=email,
            subject='New SignUp',
            html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for
choosing us.</p>')

        sg = SendGridAPIClient(
            api_key=os.environ.get('SENDGRID_API_KEY'))

        response = sg.send(message)
        print(response.status_code, response.body)
        return render_template("login.html", meg=mg)

    elif request.method == 'POST':
        msg = "fill out the form first!"
        return render_template("signup.html", meg=mg)

@app.route('/dashboard', methods=['POST', 'GET'])
@login_required
def dashBoard():
    sql = "SELECT * FROM stocks"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    stocks = []
    headings = [*dictionary]
    while dictionary != False:
        stocks.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)

    return render_template("dashboard.html", headings=headings, data=stocks)

@app.route('/addstocks', methods=['POST'])
@login_required
def addStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            quantity = request.form['quantity']
            price = request.form['price']
            total = int(price) * int(quantity)
            insert_sql = 'INSERT INTO stocks
(NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES (?,?=?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, price)
            ibm_db.bind_param(pstmt, 4, total)

```

```

        ibm_db.execute(pstmt)

except Exception as e:
    msg = e

finally:
    # print(msg)
    return redirect(url_for('dashBoard'))

@app.route('/updatestocks', methods=['POST'])
@login_required
def UpdateStocks():
    if request.method == "POST":
        try:
            item = request.form['item']
            print("hello")
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE stocks SET ' + field + " = ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
            if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
                insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, item)
                ibm_db.execute(pstmt)
                dictionary = ibm_db.fetch_assoc(pstmt)
                print(dictionary)
                total = dictionary['QUANTITY'] * dictionary['PRICE_PER_QUANTITY']
                insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, total)
                ibm_db.bind_param(pstmt, 2, item)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

    finally:
        # print(msg)
        return redirect(url_for('dashBoard'))

@app.route('/deletestocks', methods=['POST'])
@login_required
def deleteStocks():

```

```

if request.method == "POST":
    print(request.form['item'])
    try:
        item = request.form['item']
        insert_sql = 'DELETE FROM stocks WHERE NAME=?'
        pstmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(pstmt, 1, item)
        ibm_db.execute(pstmt)
    except Exception as e:
        msg = e

    finally:
        # print(msg)
        return redirect(url_for('dashBoard'))

@app.route('/update-user', methods=['POST', 'GET'])
@login_required
def updateUser():
    if request.method == "POST":
        try:
            email = session['id']
            field = request.form['input-field']
            value = request.form['input-value']
            insert_sql = 'UPDATE Client SET ' + field + ' = ? WHERE EMAIL=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('profile'))

@app.route('/update-password', methods=['POST', 'GET'])
@login_required
def updatePassword():
    if request.method == "POST":
        try:
            email = session['id']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM Client WHERE EMAIL=? AND PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)

```

```

dictionary = ibm_db.fetch_assoc(pstmt)
print(dictionary)
if curPassword == confirmPassword:
    insert_sql = 'UPDATE Client SET PASSWORD=? WHERE EMAIL=?'
    pstmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(pstmt, 1, confirmPassword)
    ibm_db.bind_param(pstmt, 2, email)
    ibm_db.execute(pstmt)
except Exception as e:
    msg = e
finally:
    # print(msg)
    return render_template('result.html')

```

```

@app.route('/orders', methods=['POST', 'GET'])
@login_required
def orders():
    query = "SELECT * FROM orders"
    stmt = ibm_db.exec_immediate(conn, query)
    dictionary = ibm_db.fetch_assoc(stmt)
    orders = []
    headings = [*dictionary]
    while dictionary != False:
        orders.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template("orders.html", headings=headings, data=orders)

```

```

@app.route('/createOrder', methods=['POST'])
@login_required
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']
            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['PRICE_PER_QUANTITY'])
                query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?,?,,?,?)'

```

```

        pstmt = ibm_db.prepare(conn, query)
        ibm_db.bind_param(pstmt, 1, stock_id)
        ibm_db.bind_param(pstmt, 2, quantity)
        ibm_db.bind_param(pstmt, 3, date)
        ibm_db.bind_param(pstmt, 4, delivery_date)
        ibm_db.bind_param(pstmt, 5, price)
        ibm_db.execute(pstmt)
    except Exception as e:
        print(e)

```

```

    finally:
        return redirect(url_for('orders'))

```

```

@app.route('/updateOrder', methods=['POST'])
@login_required
def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + " = '" + value + "' WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

    finally:
        return redirect(url_for('orders'))

```

```

@app.route('/cancelOrder', methods=['POST'])
@login_required
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

    finally:
        return redirect(url_for('orders'))

```

```

@app.route('/suppliers', methods=['POST', 'GET'])
@login_required
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

# get order ids from orders table and identify unassigned order ids
    sql = "SELECT ID FROM orders"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    order_ids = []
    while dictionary != False:
        order_ids.append(dictionary['ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

    unassigned_order_ids = set(order_ids) - set(orders_assigned)
    return
render_template("suppliers.html",headings=headings,data=suppliers,order_ids=unassigned_order_ids)

@app.route('/updatesupplier', methods=['POST'])
@login_required
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + " = ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/addsupplier', methods=['POST'])
@login_required

```



```

def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']
            insert_sql = 'INSERT INTO suppliers (NAME,ORDER_ID,LOCATION) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:
            email = session['id']
            insert_sql = 'SELECT * FROM Client WHERE EMAIL=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
        except Exception as e:
            msg = e
        finally:

```

```

        # print(msg)
        return render_template("profile.html", data=dictionary)

@app.route('/logout', methods=['GET'])
@login_required
def logout():
    print(request)
    resp = make_response(render_template("login.html"))
    session.clear()
    return resp

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

```

## Login.html

```

{% extends 'base.html' %} {% block head %}
<title>Login page</title>
<link rel="stylesheet" href="../static/css/login.css" />
{% endblock %} {% block body %}

<h2>Inventory Management</h2>
<div class="container" id="container">
    <div class="form-container sign-up-container">
        <form action="{{ url_for('signup') }}" method="POST">
            <h1>Create Account</h1>
            <div class="social-container">
                <a href="#" class="social"><i class="fab fa-facebook-f"></i></a>
                <a href="#" class="social"><i class="fab fa-google-plus-g"></i></a>
                <a href="#" class="social"><i class="fab fa-linkedin-in"></i></a>
            </div>
            <span>or use your email for registration</span>
            <input type="text" placeholder="Name" name="username" />
            <input type="email" placeholder="Email" name="email" />
            <input type="password" placeholder="Password" name="password" />
            <button>Sign Up</button>
        </form>
    </div>
    <div class="form-container sign-in-container">
        <form action="{{ url_for('login') }}" method="POST">
            <h1>Sign in</h1>
            <div class="social-container">
                <a href="#" class="social"><i class="fab fa-facebook-f"></i></a>
                <a href="#" class="social"><i class="fab fa-google-plus-g"></i></a>
                <a href="#" class="social"><i class="fab fa-linkedin-in"></i></a>
            </div>
            <span>or use your account</span>
            <input type="username" placeholder="username" name="username" />

```

```

        <input type="password" placeholder="password" name="password_1" />
        <a href="#">Forgot your password?</a>
        <button>Sign In</button>
    </form>
</div>
<div class="overlay-container">
    <div class="overlay">
        <div class="overlay-panel overlay-left">
            <h1>Welcome User!</h1>
            <p>To keep connected with us please login with your personal info</p>
            <button class="ghost" id="signIn">Sign In</button>
        </div>
        <div class="overlay-panel overlay-right">
            <h1>Hello, Friend!</h1>
            <p>Enter your personal details and start journey with us</p>
            <button class="ghost" id="signUp">Sign Up</button>
        </div>
    </div>
</div>
</div>
</div>

<script src="../../static/js/login.js"></script>

{ % endblock% }

```

## Dashboard.html

```

{ % extends 'base2.html' % } { % block head % }
<title>Dashboard</title>
{ % endblock% } { %block body% }
<h2>Dashboard</h2>
<p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
</p>
{ % include 'table.html' % }
<div class="forms-wrapper">
    <form action="{{ url_for('UpdateStocks') }}" method="post">
        <h3>Update Stock</h3>
        <div class="field">
            <label class="custom-label" for="item"> Enter Item</label>
            <input class="text-inputs" type="text" name="item" placeholder="milk" />
        </div>
        <div class="field">
            <label for="input-field">Choose a field :</label>
            <select name="input-field" id="field">
                <option value="NAME">NAME</option>
                <option value="PRICE_PER_QUANTITY">PRICE_PER_QUANTITY</option>
                <option value="QUANTITY">QUANTITY</option>
            </select>
        </div>
    </form>

```

```

<div class="field">
  <label class="custom-label" for="input-value"> Enter Value</label>
  <input
    class="text-inputs"
    type="text"
    name="input-value"
    placeholder=" "
  />
</div>
<button class="submit-button">Update</button>
</form>

```

```

<form action="{ { url_for('addStocks') } }" method="post">
  <h3>Add New Stock</h3>
  <div class="field">
    <label class="custom-label" for="item"> Enter the item</label>
    <input class="text-inputs" name="item" type="text" placeholder="juice" />
  </div>
  <div class="field">
    <label class="custom-label" for="quantity"> Enter quantity</label>
    <input
      class="text-inputs"
      type="number"
      name="quantity"
      placeholder="200"
    />
  </div>
  <div class="field">
    <label class="custom-label" for="price"> Enter price</label>
    <input class="text-inputs" type="number" name="price" placeholder="25" />
  </div>
  <button class="submit-button">Add Stock</button>
</form>

```

```

<form action="{ { url_for('deleteStocks') } }" method="post">
  <h3>Remove stocks</h3>
  <div class="field">
    <label class="custom-label" for="item"> Enter the item</label>
    <input class="text-inputs" name="item" type="text" placeholder="juice" />
  </div>
  <button class="submit-button red-button">Remove</button>
</form>
</div>

```

```
{ % endblock% }
```

GitHub: <https://github.com/IBM-EPBL/IBM-Project-54059-1661588263>

## **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, DOI:10.1109/ICI2ST51859.2021.00018, Conference: 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)