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

7.1 Feature 1

7.2 Database Schema (if Applicable)

8. TESTING

8.1 Test Cases

9. RESULTS

9.1 Performance Metrics

10. ADVANTAGES & DISADVANTAGES

11. CONCLUSION

12. FUTURE SCOPE

CHAPTER-1: INTRODUCTION

1.1 Project Overview

The project Inventory Management System is a complete desktop based application designed on .Net technology using Visual Studio Software. The main aim of the project is to develop Inventory Management System Model software in which all the information regarding the stock of the organization will be presented. It is an intranet based desktop application which has admin component to manage the inventory and maintenance of the inventory system. This desktop application is based on the management of stock of an organization. The application contains general organization profile, sales details, Purchase details and the remaining stock that are presented in the organization. There is a provision of updating the inventory also. This application also provides the remaining balance of the stock as well as the details of the balance of transaction. Each new stock is created and entitled with the named and the entry date of that stock and it can also be update any time when required as per the transaction or the sales is returned in case. Here the login page is created in order to protect the management of the stock of organization in order to prevent it from the threads and misuse of the inventory

1.2 Purpose

The primary objectives of the project are mentioned below:

- To fulfill the requirement for achieving the Bachelor's degree of Computer Information System.
- To know the fundamentals of the .Net Technology and Visual Studio with the .Net Framework

The secondary objectives of this project are mentioned below:

- To develop an application that deals with the day to day requirement of any production organization
- To develop the easy management of the inventory
- To handle the inventory details like sales details, purchase details and balance stock details.
- To provide competitive advantage to the organization.
- To provide details information about the stock balance.
- To make the stock manageable and simplify the use of inventory in the organization.

2. LITERATURE SURVEY

2.1 Existing problem

- After analyzing many existing IMS we have now the obvious vision of the project to be developed.
- Before we started to build the application team had many challenges.
- We defined our problem statement as:
- To make desktop based application of IMS for small organization.
- To make the system easily managed and can be secured. To cover all the areas of IMS like purchase details, sales details and stock management.

2.2 References

Software Reference

- Swatik Accounting And Inventory Software High-tech Software, Kalimati
- Inventory Management Software Sagar International, Balkhu

Website

Visual Studio Official Site: https://msdn.microsoft.com/en-us/library/dd492171.aspx

2.3 Problem Statement Definition

A problem statement is a concise description of an issue to be addressed or a condition to be improved upon. It identifies the gap between the current (problem) state and desired (goal) state of a process or product. Focusing on the facts, the problem statement should be designed to address the Five Ws. The first condition of solving a problem is understanding the problem, which can be done by way of a problem statement

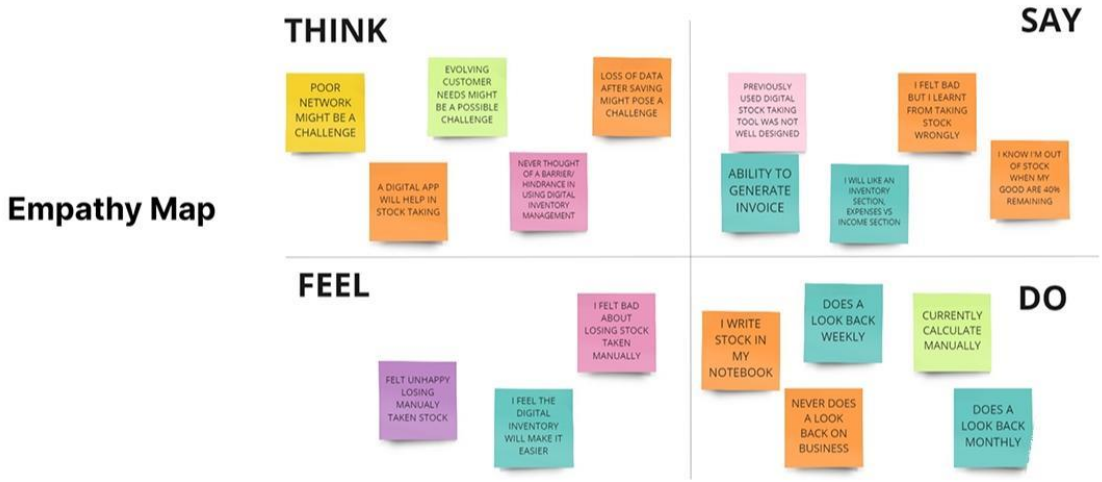
3. IDEATION & PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• the retailers generally Facing issues in recording the stocks and its threshold limit available. • The retailer doesn't know which product is getting expired and when it is being expired. • The retailers couldn't track the availability of all the stocks up-to date. • The customers are not satisfied with the retailers store since it does not have enough supplements and the deliveries were not made on time.
2.	Idea / Solution description	• This proposed system will have a daily update system whenever a product is sold or it is renewed more. • The system will have an alert triggered to indicate both the expired product and soon going to expire products. • The product availability is tracked daily and an alert system in again kept on to indicate those products which falls below the threshold limit.

		• Tracking the order have become easy with this application for both the retailers and the customers
3.	Novelty / Uniqueness	• Certain machine learning algorithms are used to predict the seasonal high selling products which can be made available during that time. • Prediction of the bestselling 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. Exclusive discounts and offers are given for regular customers to keep them engaged with the store regularly.
4.	Social Impact / Customer Satisfaction	• The customers will be highly satisfied since the wasting of time while searching for an unavailable product is reduced. • The work load of the retailers will be minimized if the system is automated every day and during every purchase. • The customer satisfaction will be improved for getting appropriate response from the retailers and that too immediately.
5.	Business Model (Revenue Model)	Hereby we can provide a robust and most reliable inventory management system by using: 1.Can deploy the most appropriate business Advertising models. 2. To establish a loss preventing Strategy. 3. And to ensure the all time, any where availability of products System. 4. Usage of freebies business strategy for dragging the Customer's attention.
6.	Scalability of the Solution	• This system can even work More efficiently with large volume of data. • Daily and Each time purchase

		updation of the stock for preventing inventory shrinkage. • Direct chat system with the retailers and the customers for providing best customer service

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• the retailers generally Facing issues in recording the stocks and its threshold limit available. • The retailer doesn't know which product is getting expired and when it is being expired. • The retailers couldn't track the availability of all the stocks up-to date. • The customers are not satisfied with the retailers store since it does not have enough supplements and the deliveries were not made on time.
2.	Idea / Solution description	• This proposed system will have a daily update system whenever a product is sold or it is renewed more. • The system will have an alert triggered to indicate both the expired product and soon going to expire products. • The product availability is tracked daily and an alert system in again kept on to indicate those products which falls below the threshold limit. • Tracking the order have become easy with this application for both the retailers and the customers
3.	Novelty / Uniqueness	• Certain machine learning algorithms are used to predict the seasonal high selling products which can be made available during that time. • Prediction of the bestselling 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. Exclusive discounts and offers are given for regular customers to keep them engaged with the store regularly.
4.	Social Impact / Customer Satisfaction	• The customers will be highly satisfied since the wasting of time while searching for an unavailable product is reduced. • The work load of the retailers will be minimized if the system is automated every day and during every purchase. • The customer satisfaction will be improved for getting appropriate response from the retailers and that too immediately.
5.	Business Model (Revenue Model)	Hereby we can provide a robust and most reliable inventory management system by using: 1.Can deploy the most appropriate business Advertising models. 2. To establish a loss preventing Strategy. 3. And to ensure the all time, any where availability of products System. 4. Usage of freebies business strategy for dragging the Customer's attention.
6.	Scalability of the Solution	• This system can even work More efficiently with large volume of data. • Daily and Each time purchase updating of the stock for preventing inventory shrinkage. • Direct chat system with the retailers and the customers for providing best customer service

3.4 Problem Solution fit

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

- Customers are retailers, shop owners, business people who are struggling to keep track of their inventory
- Due to their issue, they face many issues like:
  - Loss due to dead products in the inventory availability of fast-moving products.
  - Unnecessary headache due to improper maintenance of inventory.

Define CS, fit into CC

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

- Since most of the software like these will be a subscription model, the consumer must be paying as they use them. This may be against their budget.
- To use this software the customer must be trained or he must hire a person to do that for him
- To deploy this software the customer must have a powerful device which is compatible with the software.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

- Solution: the traditional solution for the inventory management problem is to track the incoming and outgoing goods with a pen and paper
- Pros:
  1. Easy to use
  2. Less cost
- Cons:
  1. Error rate is high
  2. Manual tracking is tedious work

Explore AS, differentiate

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers?
There could be more than one; explore different sides.

- The objective of the software is to make the inventory tracking easier by automating the inventory. Example the initial stocks information is fed to the software and from there it tracks the details of incoming and outgoing products.
- This can generate automatic alerts/notifications to help the user in their work. Example alert for dead stocks in inventory. Alerts for the goods which is to be refilled, notifications for the user defined conditions like if sales go higher than certain limits etc..,
- Graphical representation of sales is also possible

Focus on J&P, tap into BE, understand RC

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

- The primary reason for this problem to exist is the periodic change in demand of the customers.
- This indirectly affects the inventory as change in customers needs is proportional to the sale of a particular products.
- This keeping track of inventory effectively helps n managing the dead and fast moving products

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

- The customer must find a effective tracking software.
- He must implement it in this business to streamline his work and make more profit
- He must volunteer himself to learn and to use the software or be ready to hire a person who can do it for him.

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

- Understanding the fact that using a software to automate inventory system helps him to make more money and also make their work easier. Also seeing other retailers making more money using this software

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

Before: They feel lost due to loses which occur due to improper management of inventory(manual pen and paper tracking)
After: They feel like success after making increased profits, reducing the mistakes that happen in manual process

Define CS, fit into CL

10. YOUR SOLUTION SL

What kind of solution suits Customer scenario the best?
Adjust your solution to fit Customer behaviour, use Triggers, Channels & Emotions for marketing and communication.

- Design a flask based inventory management system application.
- Enable email based alerts for dead and fast moving products using SendGrid framework
- Provide an option for graphical view of sales

8.1 ONLINE CHANNELS CH

What kind of actions do customers take online?
Extract online channels from box #7 Behaviour

Online inventory trackers which come for free may steal personal information of users and it may also contains a lot of ads

8.2 OFFLINE CHANNELS CH

What kind of actions do customers take offline?
Extract offline channels from box #7 Behaviour and use them for customer development.

Manual logs can be maintained. Employees can be hired to maintain the inventory system logs when the business grows

Explore AS, differentiate

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	• Registration through own application. • Form Registration through Gmail. • Registration through LinkedIn. • Registration through Google Docs.
FR-2	User Confirmation	• Confirmation via Email • Confirmation via OTP
FR-3	User Login	• Login through User name and pass-word. • Login through mail I'D and password. • Login through OTP through mail I'd and password. • Login through Phone number.
FR-4	Records of the products	• Product • Name • Product • category • Product I'd • Stock Count Vendor details
FR-5	Login details	• Login Details along with time through E-mail. • Login Details along with time through phone number.
FR-6	Updating inventory Details.	• Update through E-mail • Update through User account
FR-7	Unavailability Alert	• Alert Message through mail or phone number.
FR-8	Monitoring of stock	• Audit monitoring through incoming and outgoing stock
FR-9	Database	• Usage of standard database for storing the data.

4.2 Non-Functional requirements

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

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	• 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. • It can use by wide variety of client as it is very simple to learn and not complex to proceed • Easy to use, User-friendly and Responsive.
NFR-2	**Security**	• Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. With Registered Mail id only, retailers can log into the application. So, it provides authentication. • We are using login for the user and the information will be hashed so that it will be very secure to use.
NFR-3	**Reliability**	• It will be reliable that it can update with very time period so that the accuracy will be good.
NFR-4	**Performance**	• User can track the record of goods available using the application. Inventory tracking helps to improve inventory management and ensures that having optimal stock available to fulfill orders. Reduces manpower , cost and saves time. Emails will be sent automatically While stocks are not available. Makes the business process more efficient. Improves organizations performance. • It will be performed fast and secure even at the lower bandwidth

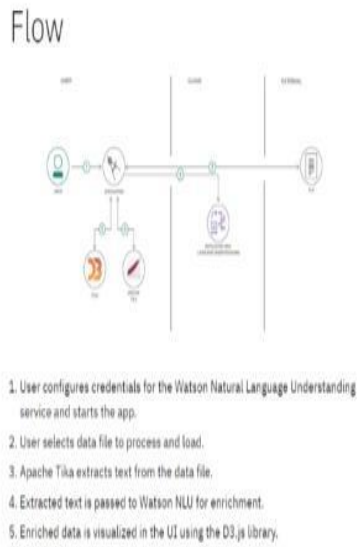
NFR-5	**Availability**	• The availability of product is just one way in which an inventory management system creates customer satisfaction. Inventory management systems are designed to monitor product availability, determine purchasing schedules for better customer interaction. • Prediction will be available for every user but only for premium user news database and price alert will be alert
NFR-6	**Scalability**	• Scalability is an aspect or rather a functional quality of a system, software or solution. This proposed system for inventory management system can accommodate expansion without restricting the existing workflow and ensure an increase in the output or efficiency of the process • It is scalable that we are going to use data in kilobytes so that the quite amount of storage is satisfied

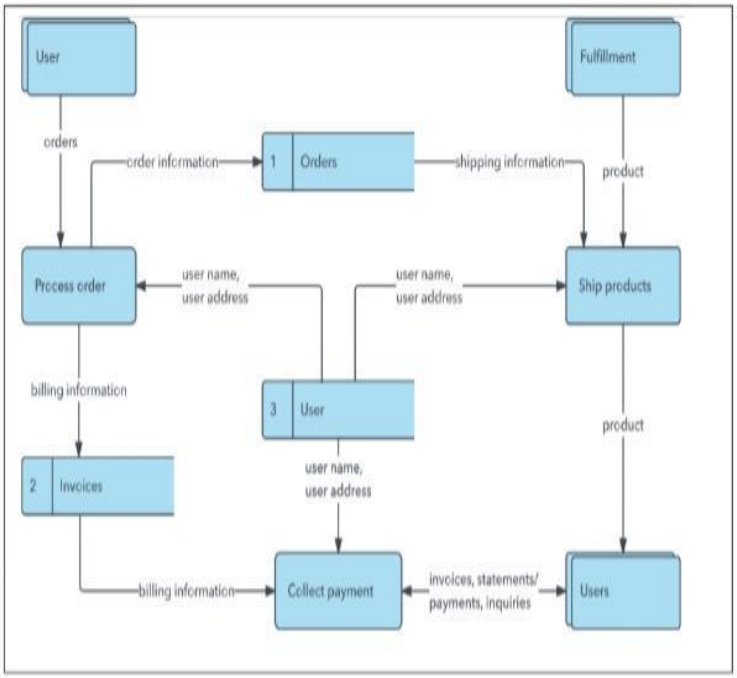
5. PROJECT DESIGN

5.1 Data Flow Diagrams

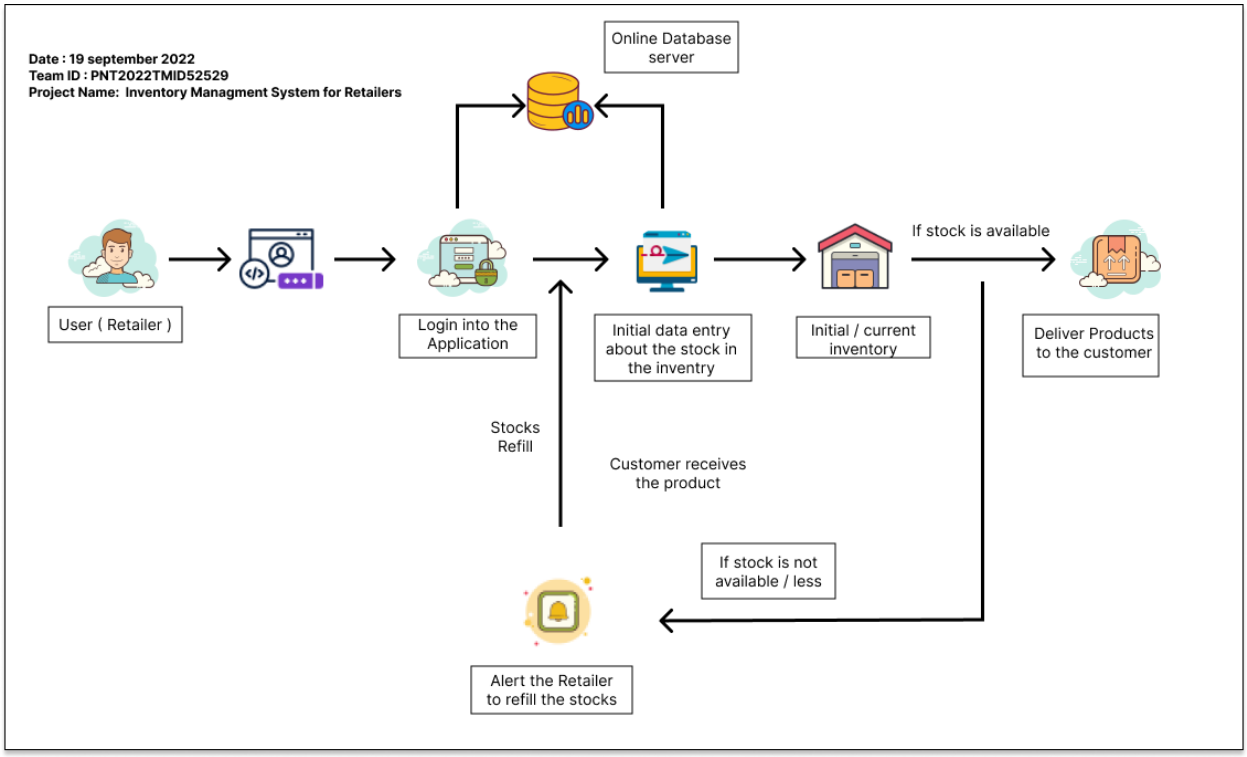
Data Flow Diagrams:

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

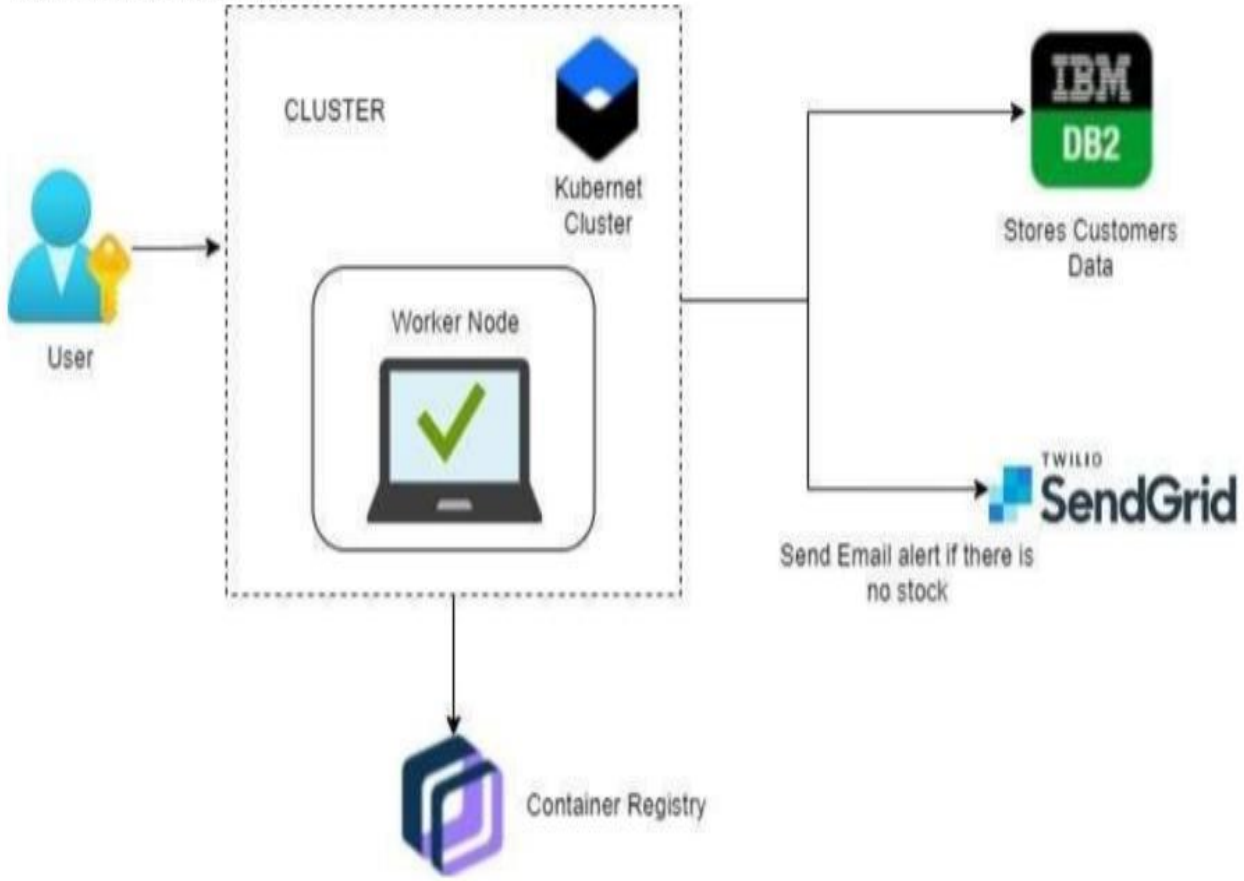




5.2 Solution & Technical Architecture



Technical Architecture:



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can create my account & access my dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation through email & click to confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register with Facebook Login	Low	Sprint-3
		USN-4	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-2
	Login	USN-5	As a user, I can log into the application by entering email & password	I can log in by entering Gmail & password	High	Sprint-1
	Dashboard	USN-6	As a user, I can track data of sales of products and inventory levels	I can track data of inventory levels & sales.	High	Sprint-1
Customer (Web user)	Registration	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-8	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-9	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-3
		USN-10	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-2
	Login	USN-11	As a user, I can log into the application by entering email & password	I can log in by entering Gmail & password	High	Sprint-1
	Dashboard	USN-12	As a user, I can track data of sales of products and inventory levels	I can track data of sales of products and inventory levels.	High	Sprint-1
Customer Care	Support	USN-13	As a Executive, I Provide answers for the queries asked by users.	I provide the answers for the queries asked by	High	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Executive				the users.		

PROJECT PLANNING & SCHEDULING

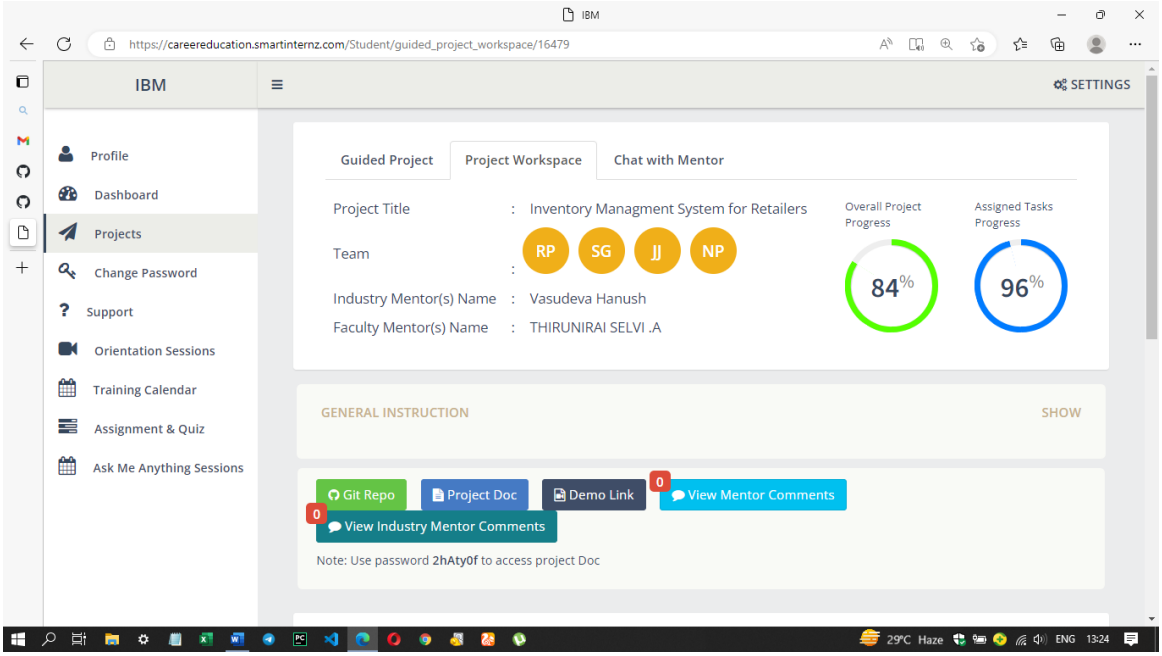
Sprint Planning & Estimation

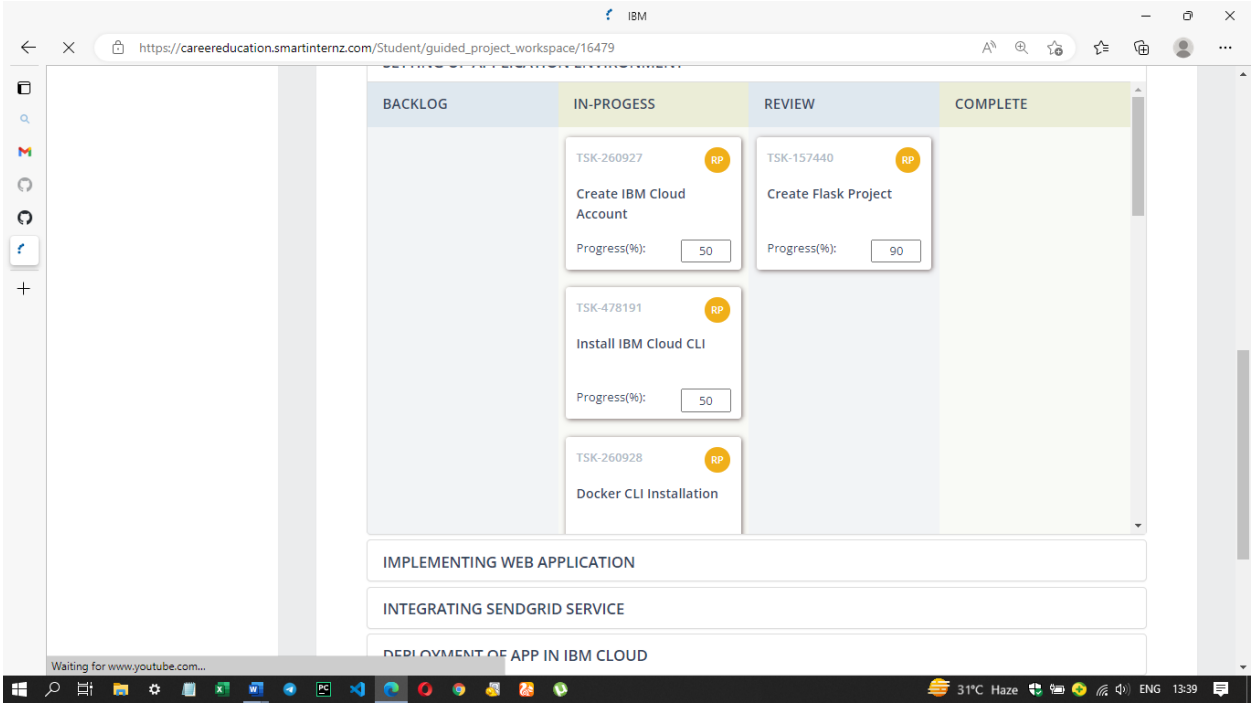
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login & Registration UI Design	USN-1	As a user, I want login and registration page design	2	High	4
Sprint-1	Landing Page UI Design	USN-2	As a user, I want to view the application overview and available functionalities	1	High	4
Sprint-2	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	2	Low	4
Sprint-2	Login	USN-4	As a user, I can log into the application by entering email & password	2	Medium	4
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	1	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 a report	5	Medium	4

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	7	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	9	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	5	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	10	19 Nov 2022

6.3 Reports from JIRA





7. CODING & SOLUTIONING

App.py

```
from flask import Flask, render_template, flash, redirect,
url_for, session, request, logging

from wtforms import Form, StringField, TextAreaField,
PasswordField, validators, SelectField, IntegerField

import ibm_db

from passlib.hash import sha256_crypt

from functools import wraps


from sendgrid import *


#creating an app instance

app = Flask(__name__)


app.secret_key='aSGSGFSGsd'


conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=IBM_HO
ST;PORT=IBM_PORT;SECURITY=SSL;SSLServerCertificate=DigiCe
rtGlobalRootCA.crt;UID=USERNAME;PWD=PASSWORD;","","")


#Index
```

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

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

    products=[]
    row = ibm_db.fetch_assoc(stmt)
    while(row):
        products.append(row)
        row = ibm_db.fetch_assoc(stmt)
    products=tuple(products)
    #print(products)

    if result>0:
```

```
        return render_template('products.html', products =
products)
    else:
        msg='No products found'
        return render_template('products.html', msg=msg)

#Locations
@app.route('/locations')
def locations():

    sql = "SELECT * FROM locations"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)

    locations=[]
    row = ibm_db.fetch_assoc(stmt)
    while(row):
        locations.append(row)
        row = ibm_db.fetch_assoc(stmt)
    locations=tuple(locations)
    #print(locations)
```

```
    if result>0:
        return render_template('locations.html', locations =
locations)
    else:
        msg='No locations found'
        return render_template('locations.html', msg=msg)

#Product Movements
@app.route('/product_movements')
def product_movements():

    sql = "SELECT * FROM productmovements"
    stmt = ibm_db.prepare(conn, sql)
    result=ibm_db.execute(stmt)

    movements=[]
    row = ibm_db.fetch_assoc(stmt)
    while(row):
        movements.append(row)
        row = ibm_db.fetch_assoc(stmt)
```

```
    movements=tuple(movements)
    #print(movements)


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


#Register Form Class
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1,
max=50)])
    username = StringField('Username',
[validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6,
max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
```

```
        validators.EqualTo('confirm', message='Passwords do not
match')
    ])
    confirm = PasswordField('Confirm Password')


#user register
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password =
sha256_crypt.encrypt(str(form.password.data))


        sql1="INSERT INTO users(name, email, username,
password) VALUES(?,?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,name)
        ibm_db.bind_param(stmt1,2,email)
```

```
        ibm_db.bind_param(stmt1,3,username)
        ibm_db.bind_param(stmt1,4,password)
        ibm_db.execute(stmt1)
        #for flash messages taking parameter and the category of
message to be flashed
        flash("You are now registered and can log in", "success")

        #when registration is successful redirect to home
        return redirect(url_for('login'))
    return render_template('register.html', form = form)



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

        sql1="Select * from users where username = ?"
```

```
stmt1 = ibm_db.prepare(conn, sql1)
ibm_db.bind_param(stmt1,1,username)
result=ibm_db.execute(stmt1)
d=ibm_db.fetch_assoc(stmt1)
if result > 0:
    #Get the stored hash
    data = d
    password = data['PASSWORD']

    #compare passwords
    if sha256_crypt.verify(password_candidate, password):
        #Passed
        session['logged_in'] = True
        session['username'] = username

        flash("you are now logged in","success")
        return redirect(url_for('dashboard'))
    else:
        error = 'Invalid Login'
        return render_template('login.html', error=error)
    #Close connection
```

```
            cur.close()
        else:
            error = 'Username not found'
            return render_template('login.html', error=error)
    return render_template('login.html')

#check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login','danger')
            return redirect(url_for('login'))
    return wrap

#Logout
@app.route('/logout')
@is_logged_in
def logout():
```

```
    session.clear()
    flash("You are now logged out", "success")
    return redirect(url_for('login'))

#Dashboard
@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql2="SELECT product_id, location_id, qty FROM
product_balance"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)



    products=[]
    row = ibm_db.fetch_assoc(stmt2)
    while(row):
```

```
        products.append(row)
        row = ibm_db.fetch_assoc(stmt2)
    products=tuple(products)

    locations=[]
    row2 = ibm_db.fetch_assoc(stmt3)
    while(row2):
        locations.append(row2)
        row2 = ibm_db.fetch_assoc(stmt3)
    locations=tuple(locations)

    locs = []
    for i in locations:
        locs.append(list(i.values())[0])

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

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

#Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
    form = ProductForm(request.form)
    if request.method == 'POST' and form.validate():
        product_id = form.product_id.data
        product_cost = form.product_cost.data
        product_num = form.product_num.data

        sql1="INSERT INTO products(product_id, product_cost,
product_num) VALUES(?,?,?)"
```

```
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,product_id)
        ibm_db.bind_param(stmt1,2,product_cost)
        ibm_db.bind_param(stmt1,3,product_num)

        ibm_db.execute(stmt1)

        flash("Product Added", "success")

        return redirect(url_for('products'))

    return render_template('add_product.html', form=form)

#Edit Product
@app.route('/edit_product/<string:id>', methods=['GET',
'POST'])
@is_logged_in
def edit_product(id):
    sql1="Select * from products where product_id = ?"
    stmt1 = ibm_db.prepare(conn, sql1)
    ibm_db.bind_param(stmt1,1,id)
```

```
    result=ibm_db.execute(stmt1)
    product=ibm_db.fetch_assoc(stmt1)

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

    #populate product form fields
    form.product_id.data = product['PRODUCT_ID']
    form.product_cost.data = str(product['PRODUCT_COST'])
    form.product_num.data = str(product['PRODUCT_NUM'])

    if request.method == 'POST' and form.validate():
        product_id = request.form['product_id']
        product_cost = request.form['product_cost']
        product_num = request.form['product_num']

        sql2="UPDATE products SET
product_id=?,product_cost=?,product_num=? WHERE
product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
```

```
        ibm_db.bind_param(stmt2,1,product_id)
        ibm_db.bind_param(stmt2,2,product_cost)
        ibm_db.bind_param(stmt2,3,product_num)
        ibm_db.bind_param(stmt2,4,id)
        ibm_db.execute(stmt2)

        flash("Product Updated", "success")

        return redirect(url_for('products'))

    return render_template('edit_product.html', form=form)

#Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in
def delete_product(id):

    sql2="DELETE FROM products WHERE product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)
```

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

    return redirect(url_for('products'))

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

#Add Location
@app.route('/add_location', methods=['GET', 'POST'])
@is_logged_in
def add_location():
    form = LocationForm(request.form)
    if request.method == 'POST' and form.validate():
        location_id = form.location_id.data

        sql2="INSERT into locations VALUES(?)"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
```

```
        ibm_db.execute(stmt2)

        flash("Location Added", "success")

        return redirect(url_for('locations'))

    return render_template('add_location.html', form=form)

#Edit Location
@app.route('/edit_location/<string:id>', methods=['GET',
'POST'])
@is_logged_in
def edit_location(id):

    sql2="SELECT * FROM locations where location_id = ?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    result=ibm_db.execute(stmt2)
    location=ibm_db.fetch_assoc(stmt2)
    #Get form
    form = LocationForm(request.form)
```

```
    print(location)

    #populate article form fields
    form.location_id.data = location['LOCATION_ID']

    if request.method == 'POST' and form.validate():
        location_id = request.form['location_id']

        sql2="UPDATE locations SET location_id=? WHERE
location_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,location_id)
        ibm_db.bind_param(stmt2,2,id)
        ibm_db.execute(stmt2)

        flash("Location Updated", "success")

        return redirect(url_for('locations'))

    return render_template('edit_location.html', form=form)
```

```
#Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in
def delete_location(id):
    sql2="DELETE FROM locations WHERE location_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

    flash("Location Deleted", "success")

    return redirect(url_for('locations'))

#Product Movement Form Class
class ProductMovementForm(Form):
    from_location = SelectField('From Location', choices=[])
    to_location = SelectField('To Location', choices=[])
    product_id = SelectField('Product ID', choices=[])
    qty = IntegerField('Quantity')

class CustomError(Exception):
```

```
    pass

#Add Product Movement
@app.route('/add_product_movements', methods=['GET',
'POST'])
@is_logged_in
def add_product_movements():
    form = ProductMovementForm(request.form)

    sql2="SELECT product_id FROM products"
    sql3="SELECT location_id FROM locations"
    stmt2 = ibm_db.prepare(conn, sql2)
    stmt3 = ibm_db.prepare(conn, sql3)

    result=ibm_db.execute(stmt2)
    ibm_db.execute(stmt3)



    products=[]
    row = ibm_db.fetch_assoc(stmt2)
    while(row):
```

```
    products.append(row)
    row = ibm_db.fetch_assoc(stmt2)
products=tuple(products)

locations=[]
row2 = ibm_db.fetch_assoc(stmt3)
while(row2):
    locations.append(row2)
    row2 = ibm_db.fetch_assoc(stmt3)
locations=tuple(locations)

prods = []
for p in products:
    prods.append(list(p.values())[0])

locs = []
for i in locations:
    locs.append(list(i.values())[0])

form.from_location.choices = [(l,l) for l in locs]
```

```
    form.from_location.choices.append(("Main Inventory","Main
Inventory"))
    form.to_location.choices = [(l,l) for l in locs]
    form.to_location.choices.append(("Main Inventory","Main
Inventory"))
    form.product_id.choices = [(p,p) for p in prods]

    if request.method == 'POST' and form.validate():
        from_location = form.from_location.data
        to_location = form.to_location.data
        product_id = form.product_id.data
        qty = form.qty.data



        if from_location==to_location:
            raise CustomError("Please Give different From and To
Locations!!")



        elif from_location=="Main Inventory":
            sql2="SELECT * from product_balance where
location_id=? and product_id=?"
```

```
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,to_location)
        ibm_db.bind_param(stmt2,2,product_id)
        result=ibm_db.execute(stmt2)
        result=ibm_db.fetch_assoc(stmt2)
        print("-----------------")
        print(result)
        print("-----------------")
        app.logger.info(result)
        if result!=False:
            if(len(result))>0:
                Quantity = result["QTY"]
                q = Quantity + qty

                sql2="UPDATE product_balance set qty=? where
location_id=? and product_id=?"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,q)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.execute(stmt2)
```

```
            sql2="INSERT into
productmovements(from_location, to_location, product_id,
qty) VALUES(?, ?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.bind_param(stmt2,4,qty)
            ibm_db.execute(stmt2)
        else:

            sql2="INSERT into product_balance(product_id,
location_id, qty) values(?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,product_id)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,qty)
            ibm_db.execute(stmt2)

            sql2="INSERT into productmovements(from_location,
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
```

```
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,from_location)
        ibm_db.bind_param(stmt2,2,to_location)
        ibm_db.bind_param(stmt2,3,product_id)
        ibm_db.bind_param(stmt2,4,qty)
        ibm_db.execute(stmt2)



    sql = "select product_num from products where
product_id=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,product_id)
    current_num=ibm_db.execute(stmt)
    current_num = ibm_db.fetch_assoc(stmt)

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

ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-
qty)
```

```
        ibm_db.bind_param(stmt2,2,product_id)
        ibm_db.execute(stmt2)

        alert_num=current_num['PRODUCT_NUM']-qty

        if(alert_num<=0):
            alert("Please update the quantity of the product {},
Atleast {} number of pieces must be added to finish the pending
Product Movements!".format(product_id,-alert_num))

    elif to_location=="Main Inventory":
        sql2="SELECT * from product_balance where
location_id=? and product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,from_location)
        ibm_db.bind_param(stmt2,2,product_id)
        result=ibm_db.execute(stmt2)
        result=ibm_db.fetch_assoc(stmt2)

        app.logger.info(result)
        if result!=False:
```

```
        if(len(result))>0:
            Quantity = result["QTY"]
            q = Quantity - qty

            sql2="UPDATE product_balance set qty=? where
location_id=? and product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,q)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.execute(stmt2)

            sql2="INSERT into
productmovements(from_location, to_location, product_id,
qty) VALUES(?, ?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.bind_param(stmt2,4,qty)
            ibm_db.execute(stmt2)
```

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



        sql = "select product_num from products where
product_id=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,product_id)
        current_num=ibm_db.execute(stmt)
        current_num = ibm_db.fetch_assoc(stmt)

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

ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']+
qty)
        ibm_db.bind_param(stmt2,2,product_id)
        ibm_db.execute(stmt2)

        alert_num=q
        if(alert_num<=0):
```

```
                alert("Please Add {} number of {} to {}
warehouse!".format(-q,product_id,from_location))
        else:
            raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))




    else: #will be executed if both from_location and
to_location are specified
        f=0
        sql = "SELECT * from product_balance where
location_id=? and product_id=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,from_location)
        ibm_db.bind_param(stmt,2,product_id)
        result=ibm_db.execute(stmt)
        result = ibm_db.fetch_assoc(stmt)

        if result!=False:
            if(len(result))>0:
                Quantity = result["QTY"]
```

```
            q = Quantity - qty

            sql2="UPDATE product_balance set qty=? where
location_id=? and product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,q)
            ibm_db.bind_param(stmt2,2,from_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.execute(stmt2)
            f=1

            alert_num=q
            if(alert_num<=0):
              alert("Please Add {} number of {} to {}
warehouse!".format(-q,product_id,from_location))

        else:
          raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))

        if(f==1):
```

```
        sql = "SELECT * from product_balance where
location_id=? and product_id=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,to_location)
        ibm_db.bind_param(stmt,2,product_id)
        result=ibm_db.execute(stmt)
        result = ibm_db.fetch_assoc(stmt)

        if result!=False:
            if(len(result))>0:
                Quantity = result["QTY"]
                q = Quantity + qty

                sql2="UPDATE product_balance set qty=? where
location_id=? and product_id=?"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,q)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,product_id)
                ibm_db.execute(stmt2)
```

```
            else:

                sql2="INSERT into product_balance(product_id,
location_id, qty) values(?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
                ibm_db.bind_param(stmt2,1,product_id)
                ibm_db.bind_param(stmt2,2,to_location)
                ibm_db.bind_param(stmt2,3,qty)
                ibm_db.execute(stmt2)
            sql2="INSERT into productmovements(from_location,
to_location, product_id, qty) VALUES(?, ?, ?, ?)"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,from_location)
            ibm_db.bind_param(stmt2,2,to_location)
            ibm_db.bind_param(stmt2,3,product_id)
            ibm_db.bind_param(stmt2,4,qty)
            ibm_db.execute(stmt2)

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

        render_template('products.html',form=form)
```

```
        return redirect(url_for('product_movements'))

    return render_template('add_product_movements.html',
form=form)


#Delete Product Movements
@app.route('/delete_product_movements/<string:id>',
methods=['POST'])
@is_logged_in
def delete_product_movements(id):

    sql2="DELETE FROM productmovements WHERE
movement_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,id)
    ibm_db.execute(stmt2)

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

```
    return redirect(url_for('product_movements'))


if __name__ == '__main__':
    app.secret_key = "secret123"
    #when the debug mode is on, we do not need to restart the
server again and again
    app.run(debug=True)
```

Login.html

```
@import
url("https://fonts.googleapis.com/css2?family=Inter:wght@300
;600&display=swap");
* {
  box-sizing: border-box;
}


body {
  padding: 0;
  margin: 0;
  font-family: "Inter", sans-serif;
  background: linear-gradient(45deg, #131086, #b621f3);
```

```
  padding: 40px;
}

.container {
  min-height: calc(100vh - 80px);
  display: grid;
  grid-template-columns: 50% 50%;
  border-radius: 10px;
  overflow: hidden;
}

.login-left {
  display: flex;
  flex-direction: column;
  justify-content: center;
  padding-left: 150px;
  background-color: white;
}

.login-right {
  background-color: #eeeeee;
```

```
  display: flex;
  justify-content: center;
}
.login-right img {
  width: 80%;
}

.login-header {
  margin-bottom: 50px;
}
.login-header h1 {
  font-size: 40px;
  margin-bottom: 10px;
}
.login-header p {
  opacity: 0.7;
}

.login-form {
  width: 450px;
}
```

```
.login-form-content {
  display: flex;
  flex-direction: column;
  gap: 35px;
}
.login-form-footer {
  display: flex;
  gap: 30px;
  margin-top: 70px;
}
.login-form-footer a {
  flex: 6;
  gap: 15px;
  border-radius: 100px;
  padding: 0.6rem;
  justify-content: center;
  border: 1px solid black;
  display: flex;
  align-items: center;
  color: black;
  text-decoration: none;
```

```
}
.login-form-footer a:hover {
  background-color: #eeeeee;
}

.form-item label:not(.checkboxLabel) {
  display: inline-block;
  background-color: white;
  margin-bottom: 10px;
  position: absolute;
  padding: 0 10px;
  transform: translate(30px, -10px);
  font-size: 14px;
}

input[type=text],
input[type=password],
input[type=email] {
  border: 1px solid black;
  outline: none;
  height: 55px;
```

```
  padding: 0 2rem;
  border-radius: 100px;
  width: 100%;
  transition: background 0.5s;
  font-size: 18px;
}
input[type=text]:focus,
input[type=password]:focus,
input[type=email]:focus {
  border-color: #131086;
}

.checkbox {
  display: flex;
  align-items: center;
  gap: 7px;
}

input[type=checkbox] {
  width: 20px;
  height: 20px;
```

```
  accent-color: #131086;
}

button {
  border: none;
  background: linear-gradient(45deg, #131086, #b621f3);
  color: white;
  padding: 1rem;
  border-radius: 100px;
  text-align: center;
  text-transform: uppercase;
  letter-spacing: 2px;
  font-size: 18px;
  height: 55px;
  cursor: pointer;
}

/* Responsive */
@media (max-width: 1350px) {
  .login-left {
    padding: 50px !important;
```

```
  }
  .login-form {
    width: 100%;
  }
  .login-form-footer {
    flex-direction: column;
    gap: 15px;
  }
}
@media (max-width: 768px) {
  body {
    padding: 20px;
  }
  .container {
    grid-template-columns: auto;
  }
  .login-right {
    display: none;
  }
}
```

Signup.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="static/css/login.css" />
    <link rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Material+Sym
bols+Rounded:opsz,wght,FILL,GRAD@48,600,0,0" />
    <title>Register</title>
</head>

<body>

    <div class="container">
```

```
<div class="login-left">
    <div class="login-header">
        <h1>Welcome to Go Product</h1>
        <p>Please register to use the platform</p>
    </div>
    <form class="login-form" autocomplete="off">
        <div class="login-form-content">
            <div class="form-item">
                <label for="emailForm">Enter Email</label>
                <input type="text" id="emailForm">
            </div>

            <div class="form-item">
                <label for="usernameForm">Enter User
Name</label>
                <input type="text" id="usernameForm">
            </div>

            <div class="form-item">
                <label for="passwordForm">Enter
Password</label>
```

```
                <input type="password" id="passwordForm">
            </div>

            <button type="submit">Sign In</button>
        </div>

        </form>
    </div>
    <div class="login-right">
        <img src="static/img/login.svg" alt="image">
    </div>
  </div>

</body>

</html>
```

Footer

Register.html

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


{% block body %}
<h1>Login</h1>
<form method="POST" action="">
   <div class="form-group">
      <label>Username</label>
      <input type="text" name="username" class="form-control"
value={{request.form.username}}>
   </div>
   <div class="form-group">
         <label>Password</label>
         <input type="password" name="password" class="form-control"
value={{request.form.password}}>
      </div>
   <p><button type="submit" class="btn btn-primary"
value="Submit">Submit</button></p>
</form>
{% endblock %}
```

Home.html

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

{% block body%}

<style>

   body {

     background-image: url('https://softwareauggest-blogimages.s3.ca-central-1.amazonaws.com/blog/wp-content/uploads/2016/02/14191055/9-Top-Retail-Inventory-Management-Software-for-SMEs-in-India-1068x578.png');

   }

   </style><br><br>

<div class="jumbotron mt-4">

   <h7 class="display-4">Inventory Management System for Retailers</h7><br><br>

   {% if session.logged_in == NULL %}

       <center><a href="/register" class="btn btn-primary btn-lg">Register</a>

       <a href="/login" class="btn btn-success btn-lg">Login</a></center>

   {% endif %}<br>

   <center><h5>Created By: Rajeshwaran, Joffin Joel, Sivakumar, Nambu sai Prakash</h5><center>

</div>
```

```
{% endblock %}
```

Layout.html

```
<html>
  <head>
    <meta charset="utf-8">
    <title>MyFlaskApp</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
  </head>
  <body>
    {% include 'include/_navbar.html' %}
    <div class="container mt-4">
      {% include 'include/_messages.html' %}
      {% block body %}{% endblock %}
    </div>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"></script>
  </body>
</html>
```

Products.html

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



{% block body %}
     <h1>Products</h1>
     <a class="btn btn-success"
href="/add_product">Add Product</a>
     <hr>
     <table class="table table-striped">
         <thead>
             <tr>
                 <th>Product ID</th>
                 <th>Product Cost</th>
                 <th>Product Quantity</th>
                 <th></th>
                 <th></th>
             </tr>
         </thead>
         <tbody>
                 {% for product in products %}
                <tr>

<td>{{product.PRODUCT_ID}}</td>
```

```
<td>{{product.PRODUCT_COST}}</td>

<td>{{product.PRODUCT_NUM}}</td>
                <td><a
href="edit_product/{{product.PRODUCT_ID}}"
class="btn btn-primary pull-
right">Edit</a></td>
                <td>
                    <form
action="{{url_for('delete_product',
id=product.PRODUCT_ID)}}" method="POST">
                        <input type="hidden"
name="method" value="DELETE">
                        <input type="submit"
value="Delete" class="btn btn-danger">
                    </form>
                </td>
               </tr>
               {% endfor %}
         </tbody>
     </table>
{% endblock %}
```

Dashboard.html:

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


{% block body %}
    <h1>Dashboard <small>Welcome
{{session.username}}</small></h1>
    <hr>
      {% for location in locations %}
      <div>
       <h3 class="mt-4 text-primary" >{{location}}</h3>
       <table class="table table-striped">
          <thead>
             <tr>
                <th>Product</th>
                <th>Warehouse</th>
                <th>Qty</th>
             </tr>
          </thead>
          <tbody>
                {% for product in products %}
                  {% if product.LOCATION_ID == location %}
               <tr>
                <td>{{product.PRODUCT_ID}}</td>
```

```
                <td>{{product.LOCATION_ID}}</td>
                <td>{{product.QTY}}</td>
               </tr>
                  {% endif %}
               {% endfor %}
          </tbody>
       </table>
       <hr>
       </div>
    {% endfor %}
{% endblock %}
```

Sendgrid.py

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase

def alert(main_msg):
   mail_from = 'admin@a.com'
   mail_to = 'admin@a.com'
   msg = MIMEMultipart()
   msg['From'] = mail_from
```

```
msg['To'] = mail_to
msg['Subject'] = '!Alert Mail On Product Shortage! - Regards'
mail_body = main_msg
msg.attach(MIMEText(mail_body))

try:
  server = smtplib.SMTP_SSL('smtp.sendgrid.net', 465)
  server.ehlo()
  server.login('apikey', 'SENDGRID_APIKEY')
  server.sendmail(mail_from, mail_to, msg.as_string())
  server.close()
  print("Mail sent successfully!")
except:
  print("Some Issue, Mail not Sent :(")
```

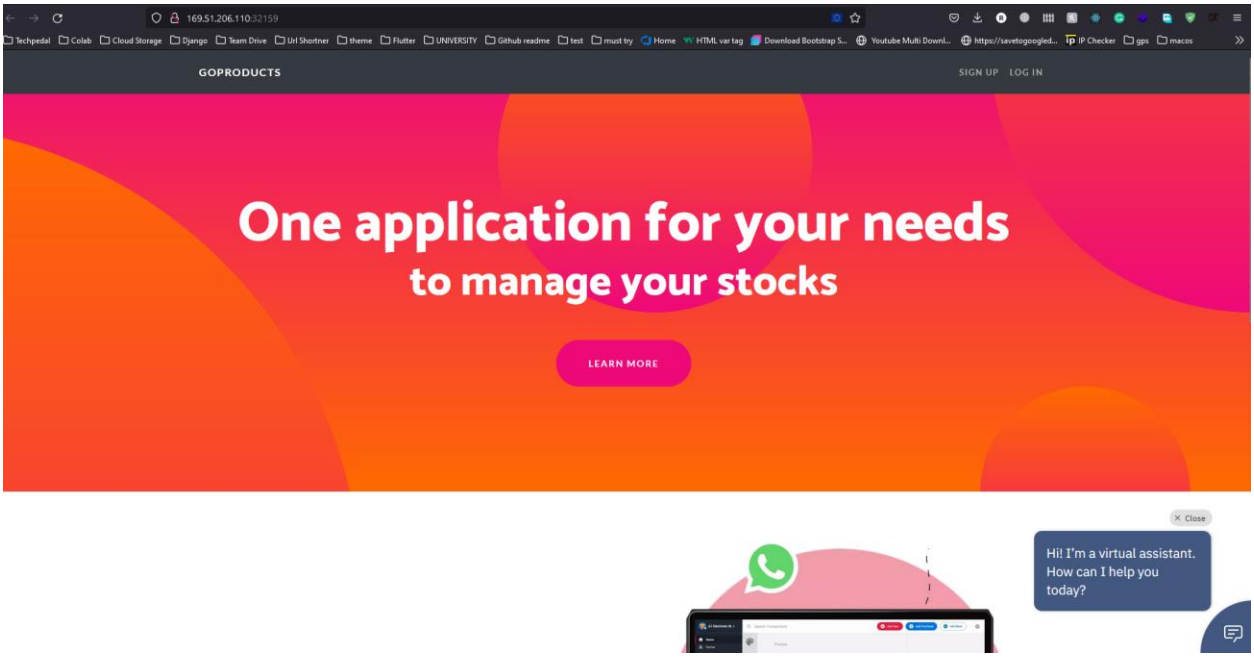
8. TESTING

8.1 Test Cases

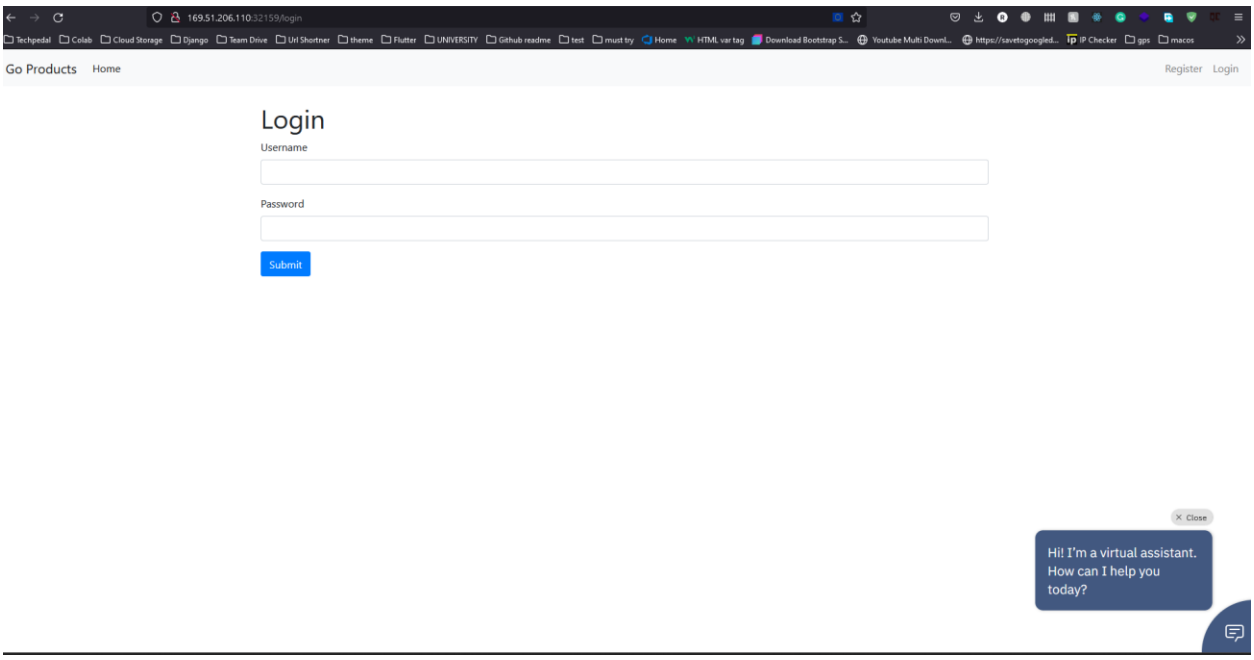
Sr no	Test cases	Action	Steps	Input data	Expected result	Actual result	Status
1	TC-1	Invoice no	Enter invoice no	Input *1021*	It should accepted invoice no.	Invoice no .is accepted	Pass
2	TC-2	Bill date	Enter bill date	Input*27/11/2021*	It should accepted bill date	Bill date is accepted	Pass
3	TC-3	Item name	Select item name	-	Item name should be automatically reflected	Item name is reflecting automatically	Pass
4	TC-4	Available item stock	Click on textbox	-	It should reflect automatically item stock	Item stock is reflecting automatically	Pass
5	TC-5	Quantity	Enter item quantity	Input*5000*	Item quantity should be accepted	Item quantity is accepting	Pass
6	TC-6	Price	Click on textbox	-	Price should be reflected automatically	Price is reflecting automatically	Pass
7	TC-7	Total	Click on textbox	-	Total should be reflected automatically	Total is reflecting automatically	Pass
8	TC-8	Receive bill date	Enter receive bill date	Input*29/1/2021*	Receive bill date should be accepting	Receive bill date is accepting	Pass
9	TC-9	Add item	Click on add item	-	It should be add item reflecting in database	Add item In reflecting database s	Pass

9. RESULTS

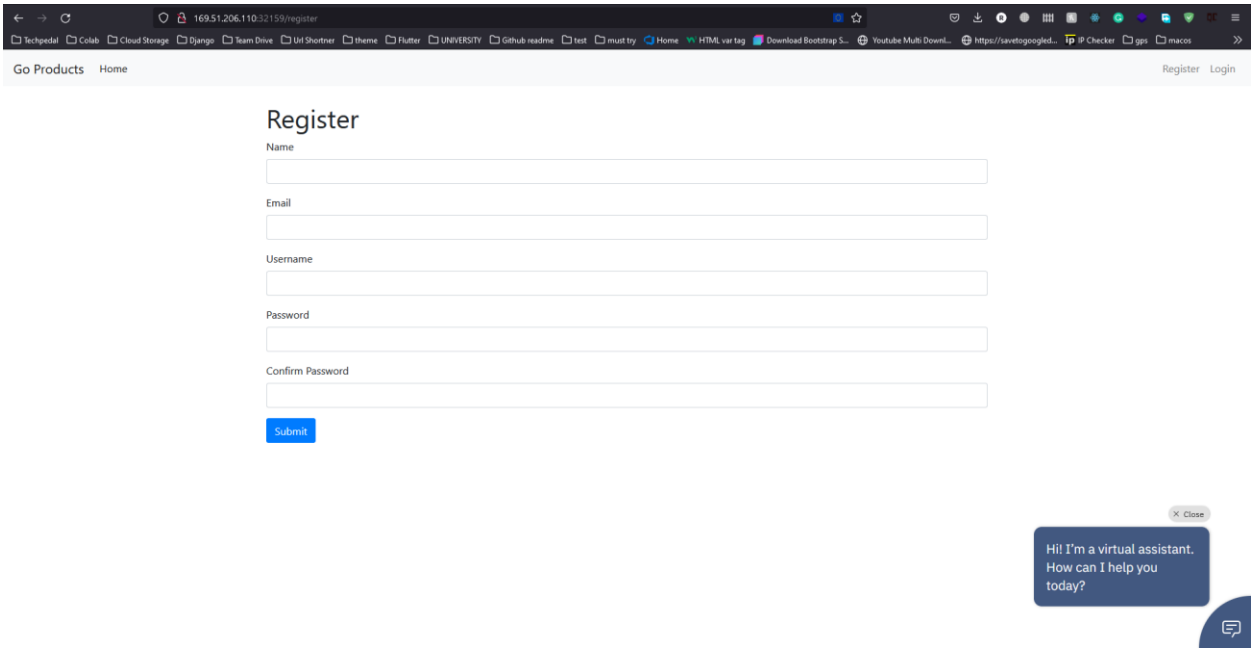
HomePage



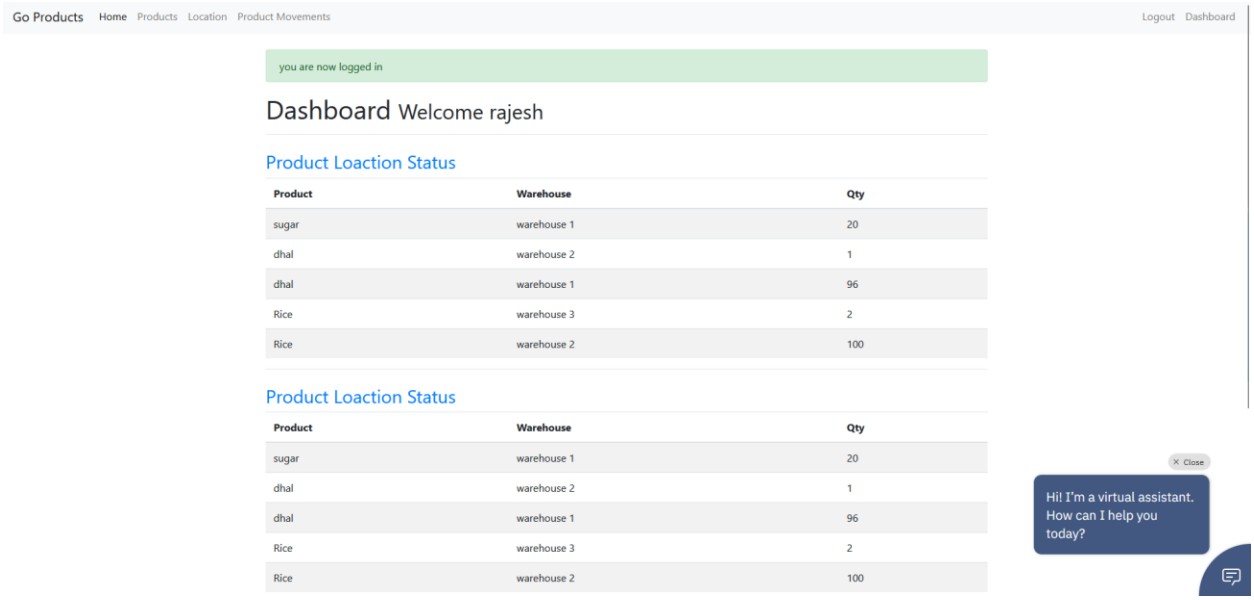
Login



Register

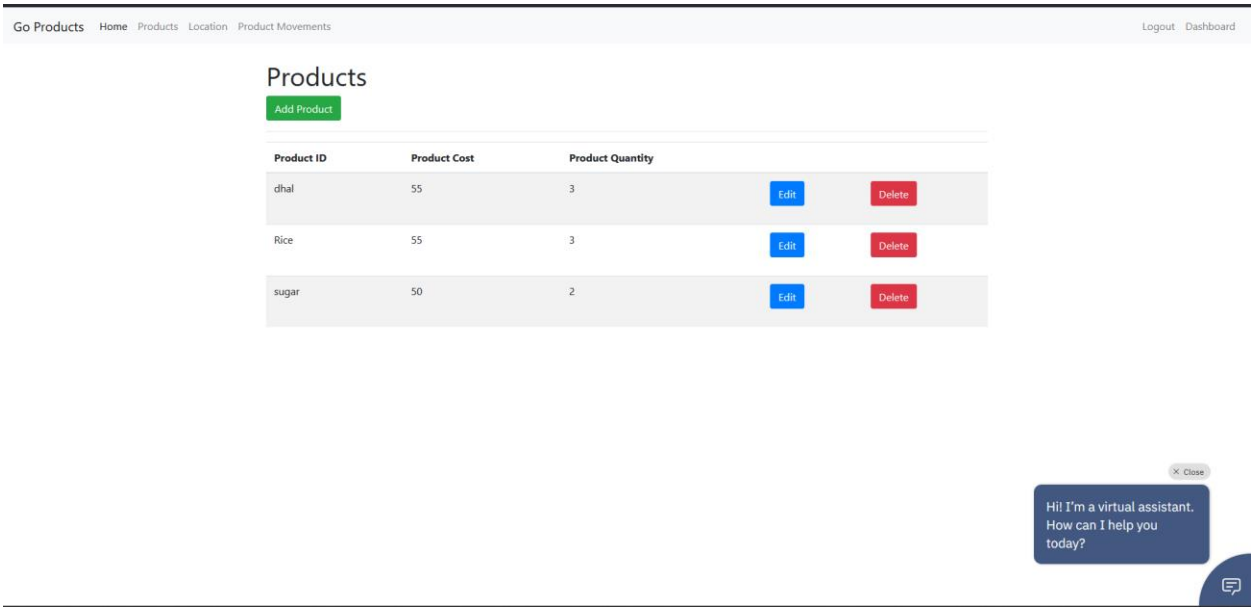


DashBoard

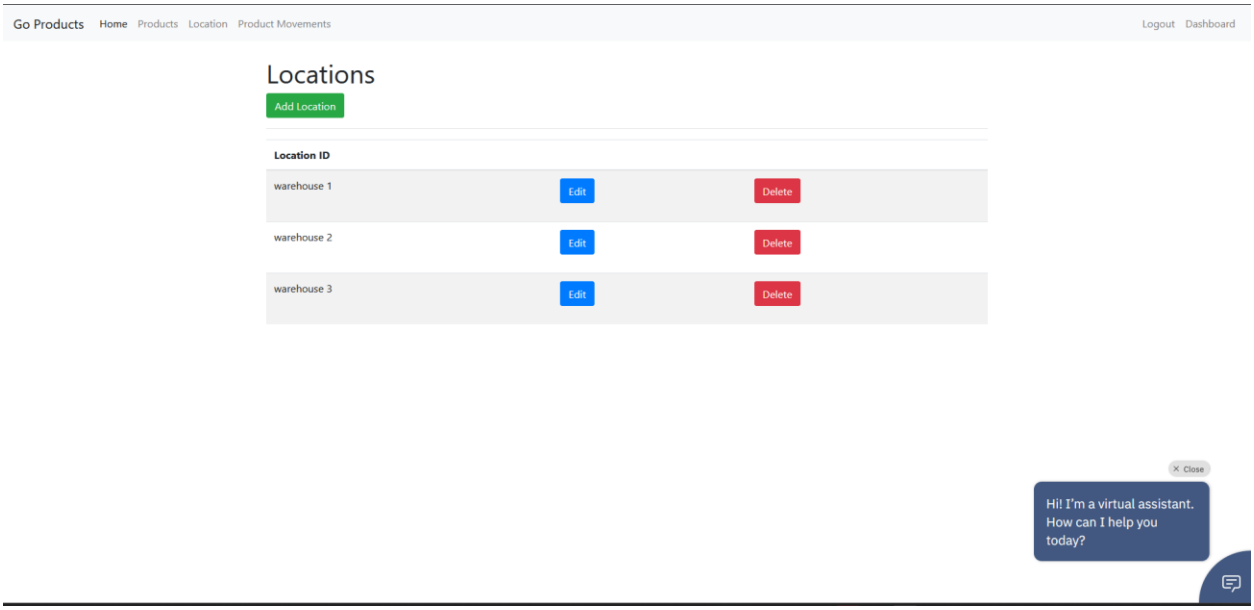


Add Product

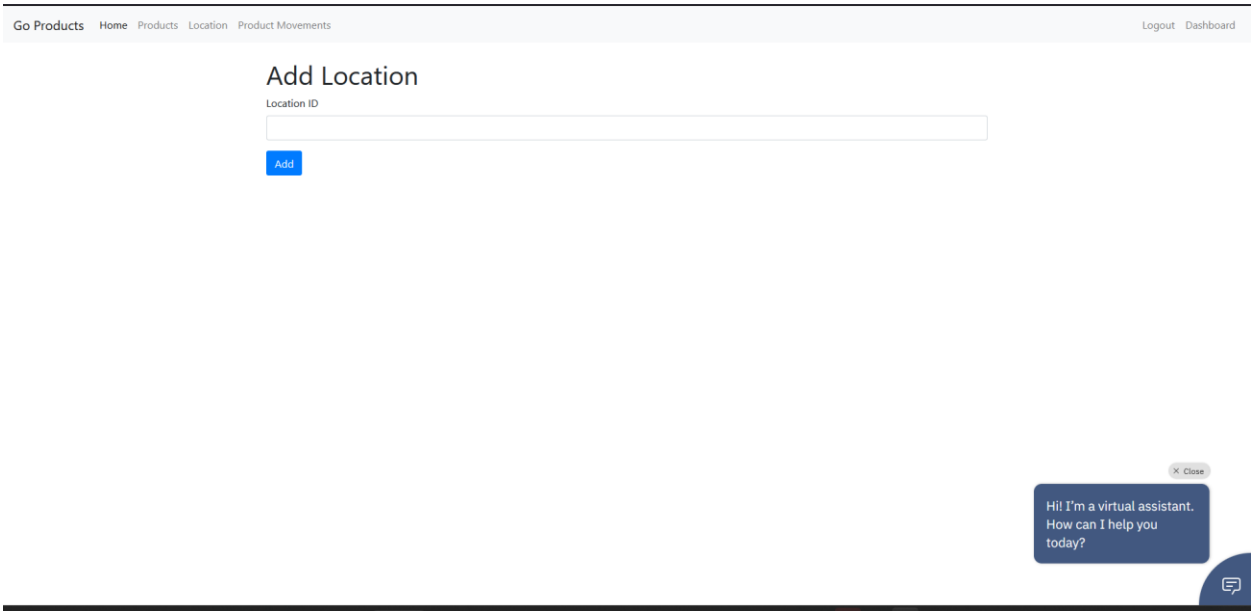
Products



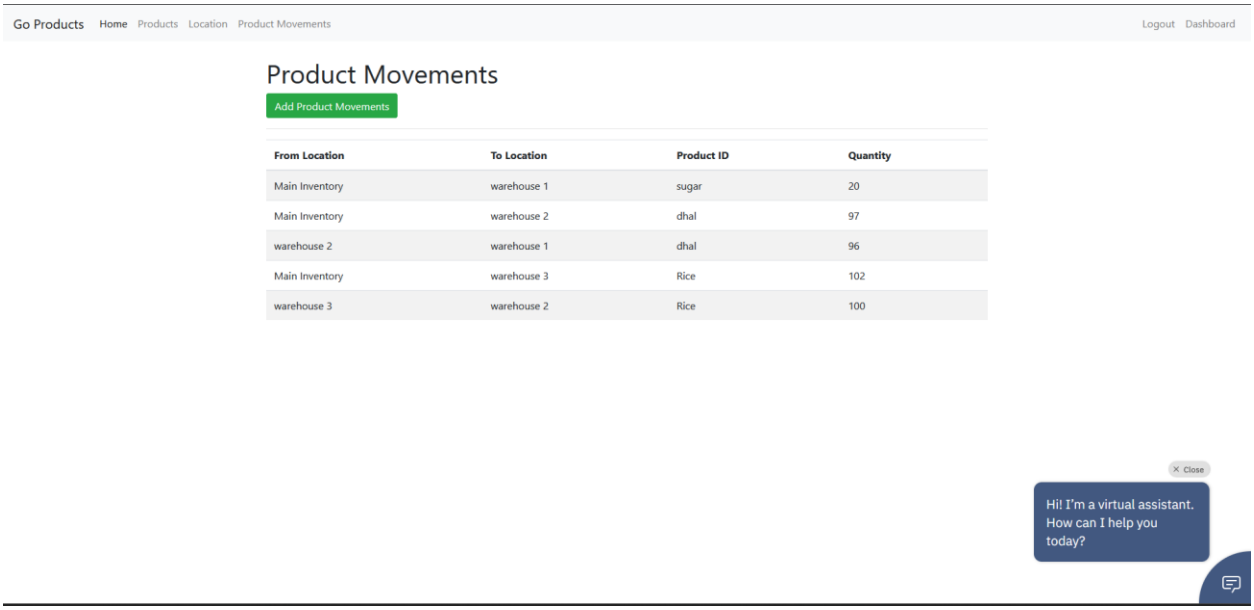
Location



Add Location

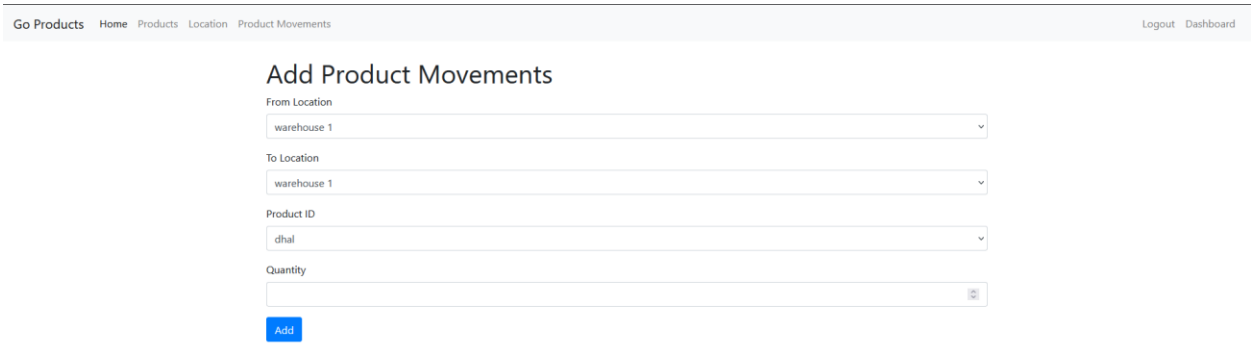


Product Movement

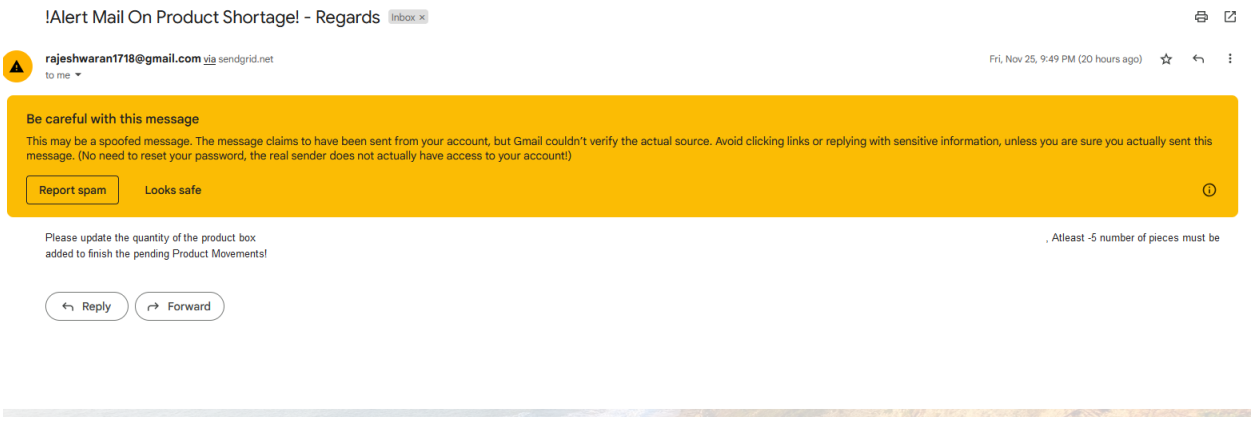


From Location	To Location	Product ID	Quantity
Main Inventory	warehouse 1	sugar	20
Main Inventory	warehouse 2	dhal	97
warehouse 2	warehouse 1	dhal	96
Main Inventory	warehouse 3	Rice	102
warehouse 3	warehouse 2	Rice	100

Add Product Movement



Alert Mail



!Alert Mail On Product Shortage! - Regards Inbox ×

rajeshwaran1718@gmail.com via sendgrid.net
to me

Fri, Nov 25, 9:49 PM (20 hours ago)

Be careful with this message

This may be a spoofed message. The message claims to have been sent from your account, but Gmail couldn't verify the actual source. Avoid clicking links or replying with sensitive information, unless you are sure you actually sent this message. (No need to reset your password, the real sender does not actually have access to your account!)

Report spam Looks safe

Please update the quantity of the product box added to finish the pending Product Movements!

, Atleast -5 number of pieces must be

Reply Forward

10. ADVANTAGES & DISADVANTAGES

Advantages:

- Automated inventory management
- Prevent stock outs and overselling
- Reduce ecommerce business costs
- Better inventory planning and forecasting
- Improving supply chain operations
- Add new selling channels easily

Disadvantages:

- This application is not suitable for those organization where there is large quantity of product and different level of warehouses
- This software application is able to generate only simple reports.
- Single admin panel is only made.
- It is not suitable for large organization.

11. CONCLUSION

To conclude, Inventory Management System is a simple desktop based application basically suitable for small organization. It has every basic items which are used for the small organization. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also provides a simple report on daily basis to know the daily sales and purchase details. This application matches for small organization where there small limited if godwoms.Through it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organization.

12. FUTURE SCOPE

Since this project was started with very little knowledge about the Inventory Management System, we came to know about the enhancement capability during the process of building it. Some of the scope we can increase for the betterment and effectiveness oar listed below:

- Interactive user interface design.
- Manage Stock Godown wise.
- Use of Oracle as its database.
- Online payment system can be added.
- Making the system flexible in any type.
- Sales and purchase return system will be added in order to make return of products.
- Lost and breakage