TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Our project is a cloud based web application that is specifically implemented to make the lives of warehouse workers much easier.It is an inventory management system for all the retailers out there in the market where they can manage,add,delete and track their goods that are being imported and exported through all locations.By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.This results in lower costs and gives them a better understanding on sales patterns.

1.2 PURPOSE

The purpose is to help retailers track and manage stocks related to their own products. The system will ask the retailers to create their accounts by providing essential details. Once retailers login succesfully into the application they can update their inventory details, also users will be bale to add new stock by submitting essential details related to the stock. They can view their inventory whenever they wish
And we have used SendGrid email service which sends an alert to retailers through email
If there is no stock found in their accounts.And they can order new stock at that time.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Warehouses of a single organization can be in different locations. It makes it really hard for the admin to keep track of all the goods across all the warehouses. Management of these information is really essential for purchasing goods on the proper time. Also these data can be used to get an insight on the recent trends for efficient purchase of goods. Also, manual tracking leads to a lot of human errors. There also exists some communication gaps between the workers and the admin which makes it even harder to keep track of the products across the warehouses.

2.2 REFERENCES

[1] Sambasiva Rao K., Singh, Sukhdev. (2006). Inventory control practices in IFFCO. The Management Accountant.

[2] Pradeep Singh(2008),” Inventory and Working Capital Management- An Empirical Analysis”, The ICFAI Journal of Accounting and Research.

[3] Capkun, Vedran, Hameri, Ari-Pekka & Weiss, Lawrence A. (2009). On the relationship between inventory and financial performance in manufacturing. International Journal of Operations & Production Management.

[4] Gaur, Jighyasu & Bhattacharya, Sourabh. (2011). The relationship of financial and inventory performance of manufacturing firms in Indian context. California Journal of Operations Management.

[5] Krishnankutty, Raveesh. (2011). Panel data analysis on retail inventory productivity. The Economic Research Guardian.

[6] Eneje, B. C., Nweze, A .U. & Udeh, A. (2012). Effect of Efficient Inventory Management on Profitability: Evidence from Selected Brewery Firms in Nigeria. International Journal of Current Research.

[7] Nyabwanga, Robert Nyamao & Ojera, Patrick. (2012). Inventory management practices and business performance for small scale enterprises in Kenya. KCA Journal of Business Management.

2.3 PROBLEM STATEMENT DEFINITION

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.
In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.
Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

3.2 IDEATION AND BRAINSTORMING

Fig 3.1: Problem Definition

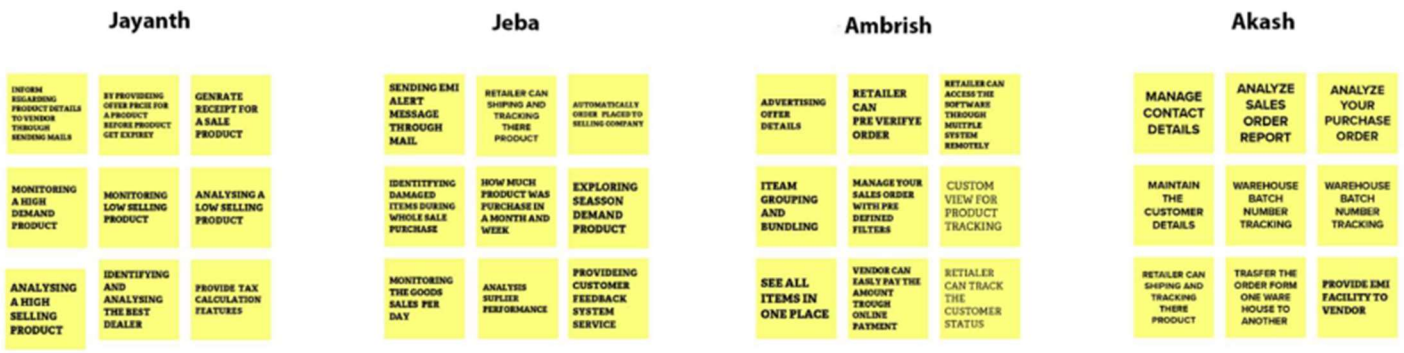


Fig 3.2: Brainstorm

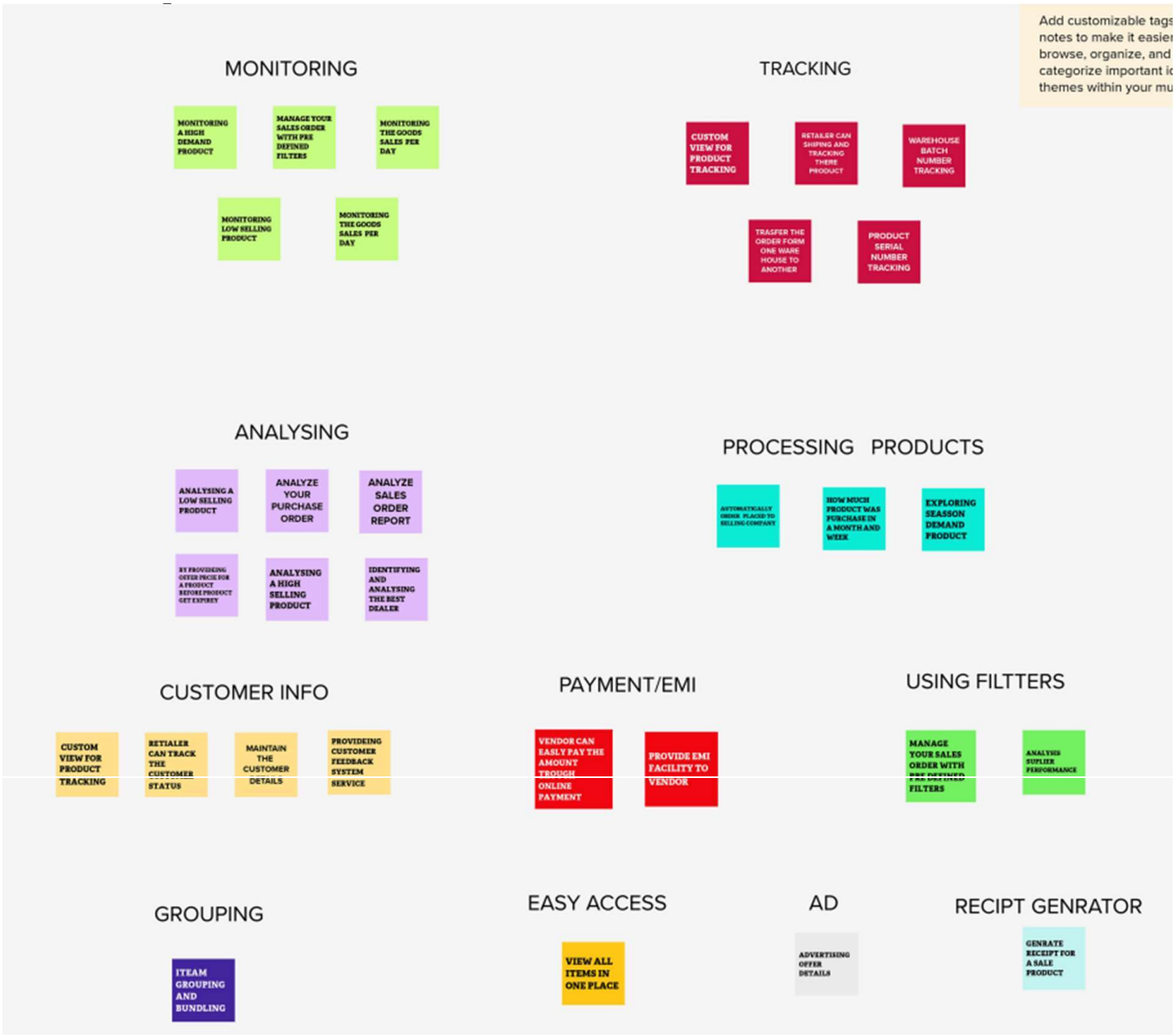


Fig 3.3: Group ideas

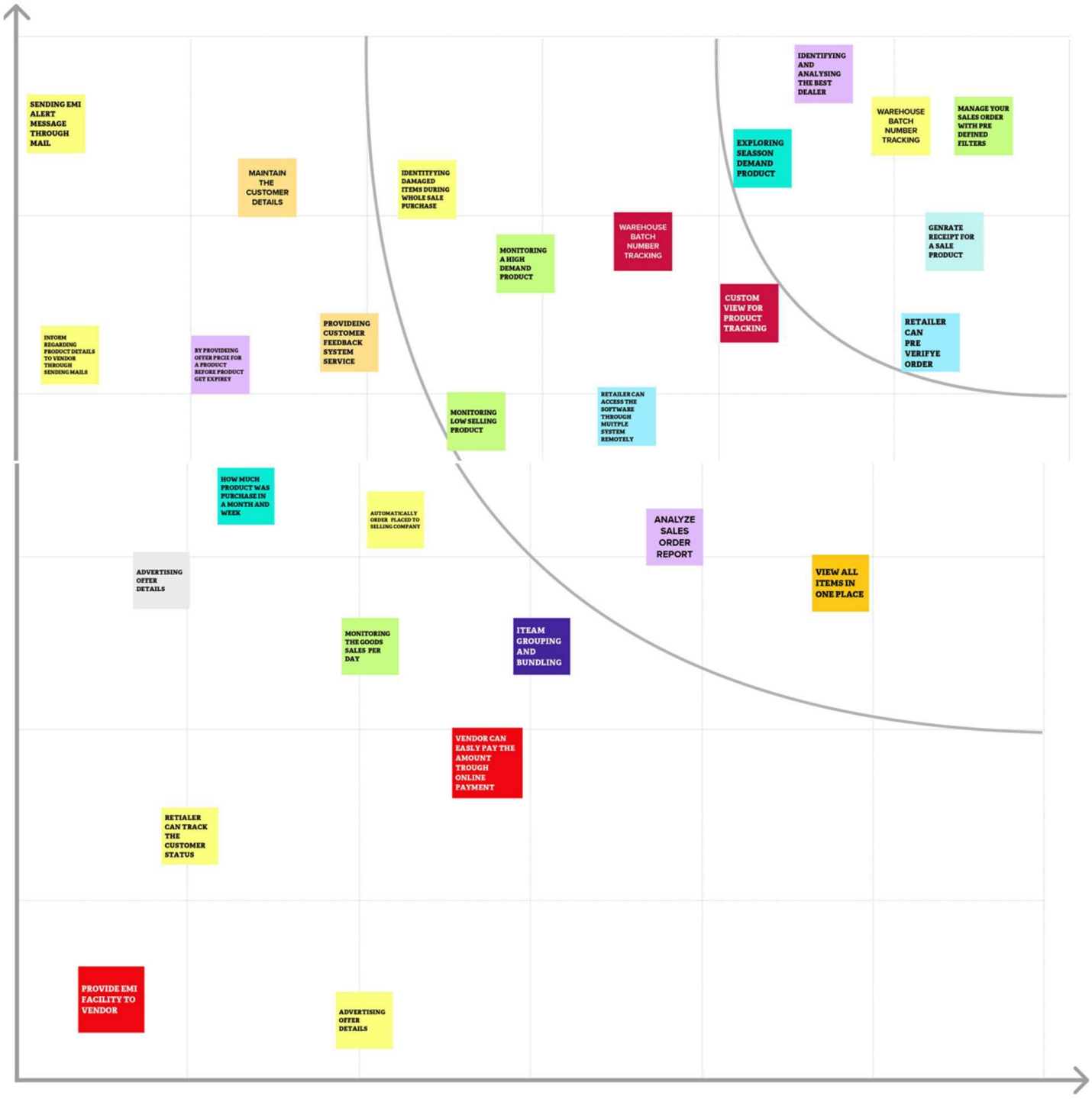
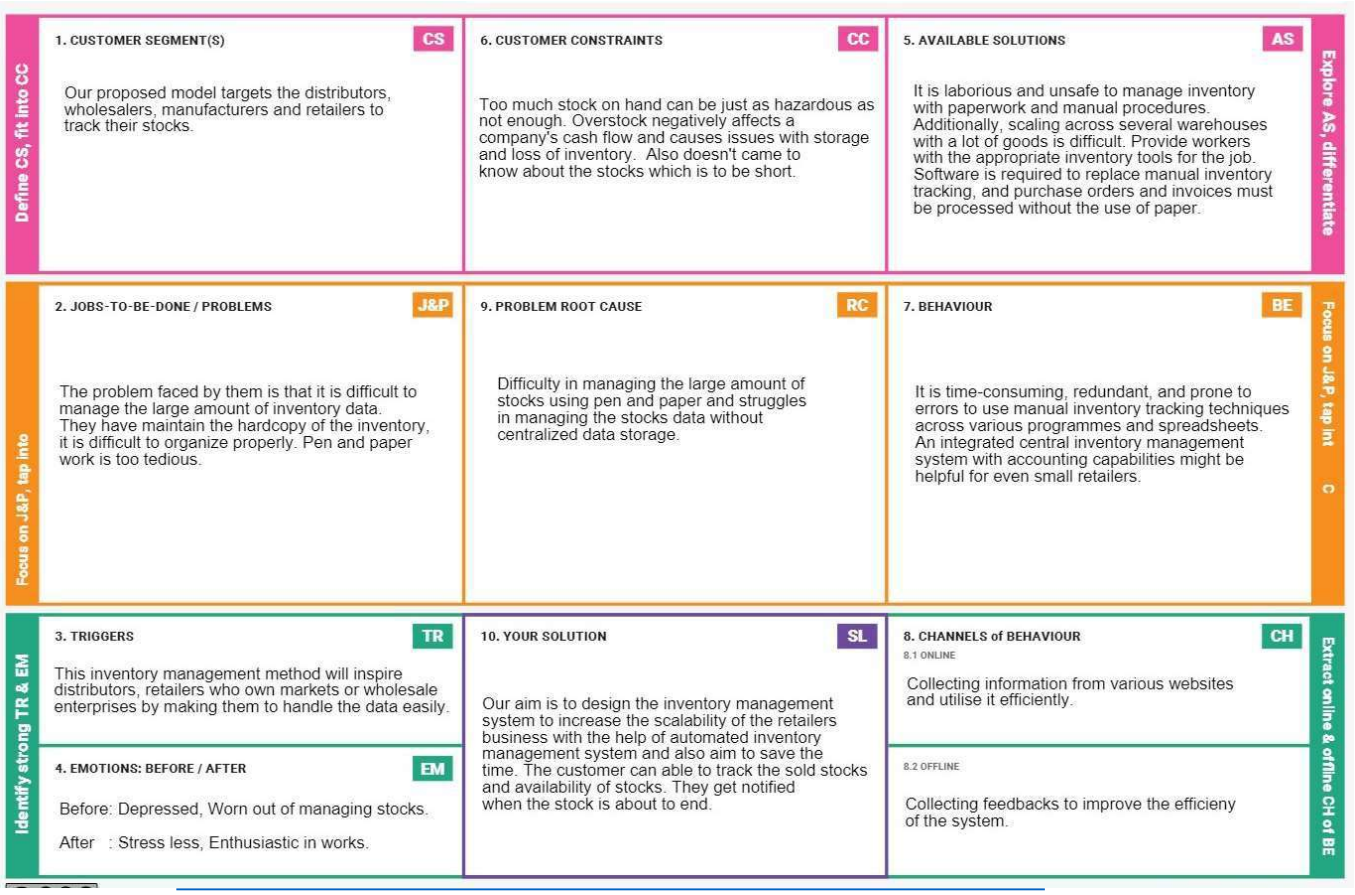


Fig 3.4: Prioritize

3.3 PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Inventory systems, demand is usually uncertain, and the lead-time can also vary. To avoid shortages, managers often maintain a safety stock. In suchsituations, it is not clear what order quantities and reorder points will minimize expected total inventory cost.
2.	Idea / Solution description	To develop an end-to-end web application which in default shows the amount of stock present in the inventory at that time. Users can add or reduce the number of goods based on purchase and sales.
3.	Novelty / Uniqueness	Track inventory across multiple locations and automatically notify when products count reaches a certain limit. This helps in saving time.
4.	Social Impact / Customer Satisfaction	It makes the life of retailers easier as it helps them keeping track of items that are stored in their warehouse.
5.	Business Model (Revenue Model)	We can charge users based on the number of warehouses they add
6.	Scalability of the Solution	Inventory data can be scaled up and scaled down based on the number of available inventory in the warehouse.

3.4 PROPOSED SOLUTION FIT



CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form
FR-2	User Login	Login with username Login with password
FR-3	Product record	Product ID Product name Product Count Minimum count to trigger reorder notification Maximum count Product category Vendor details
FR-4	Email Notification	Email through SendGridReduced stock quantity Email to both retailer and seller
FR-5	Audit Monitoring	Monitor incoming and outgoing stock

4.2 NON FUNCTIONAL REQUIREMENTS

NFR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Highly portable, User-friendly and highlyresponsive UI for easy access
NFR-2	**Security**	Access Control, User privileges, Passwordmanagement features, Hashed Password Storage
NFR-3	**Reliability**	Secure server for reliable and fault tolerant connection
NFR-4	**Performance**	The System shall be able to handle multiple requests at any given point in time and generate an appropriate response.
NFR-5	**Availability**	It is a cloud-based web application so user can access without any platform limitations, just using a browser with an internet connection is enough for use the application
NFR-6	**Scalability**	As the business grows, the users can keep track of stocks in multiple warehouses located at various locations without any hustle

CHAPTER 5

PROJECT DESIGN

5.1 DATA FLOW DIAGRAM

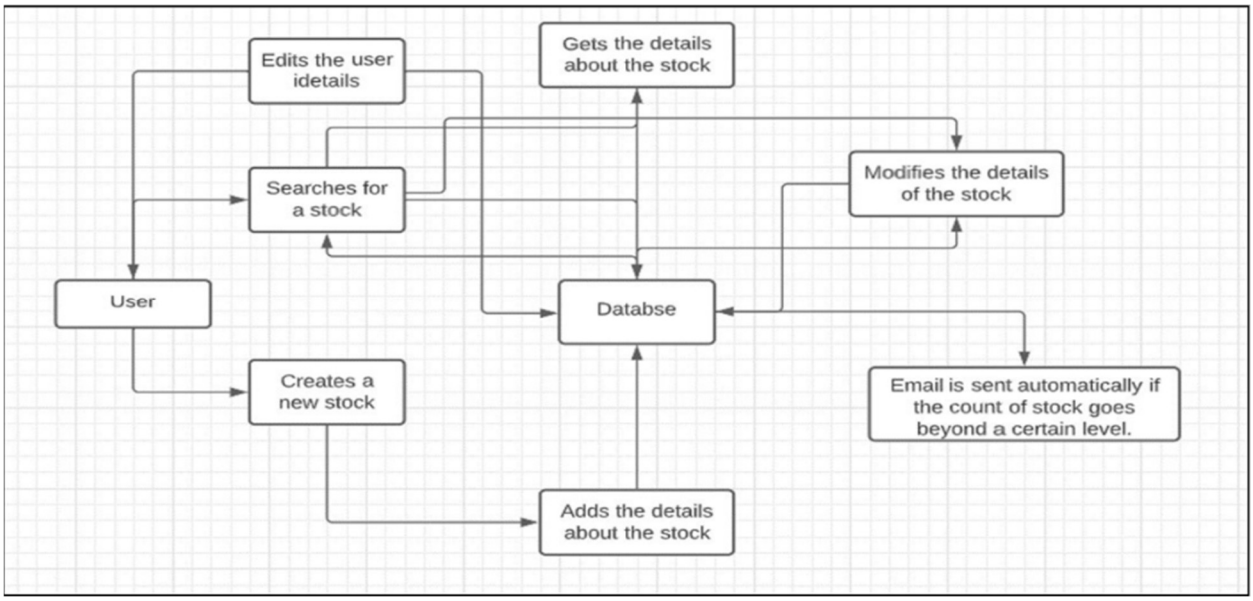


Fig 5.1: Data Flow Diagram Of Inventory Management

5.2 SOLUTION AND TECHNICAL ARCHITECTURE

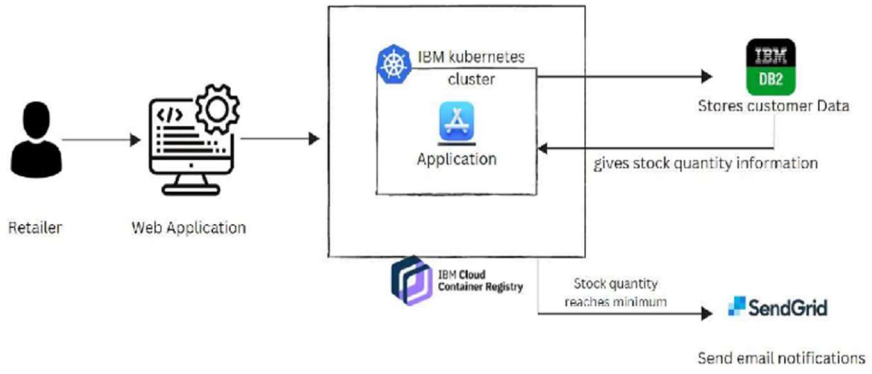


Fig 5.2: Solution Architecture Diagram

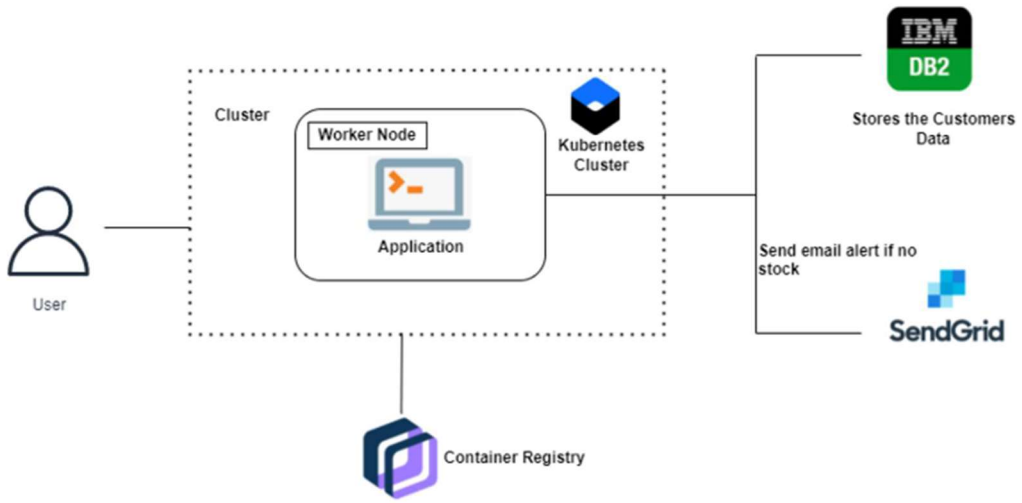


Fig 5.3: Technical Architecture

5.3 USER STORIES

User Type	Functional Requirement(Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Normal User and Admin User	Registration	USN-1	As a normal user, I can register for the application by entering my email and password and confirming my password and giving the inventory ID	I can access my account /dashboard	High	Sprint-1
		USN-2	As an admin user, I can register for the application by entering my email and password and confirming my password and giving the inventory Name	I can access my account /dashboard	High	Sprint-1
	Login and Authentication	USN-3	As a normal user and admin user, I can log into the application by entering email & password	I can Sign In	High	Sprint-1
	Dashboard	USN-4	As a normal user and admin user, I can log into my account and access the Dashboard	I can access the Dashboard	High	Sprint - 2
	Edit Details	USN-5	As a normal user, I can edit my details	I can edit my details	High	Sprint - 2

	USN-6	As an admin user, I can edit my details and change my Inventory name	I can edit my details and change inventory name	High	Sprint-2
Management	USN-7	As an admin user, I can add warehouses and add/remove products to them	I can add warehouses and add/ remove products	High	Sprint-3
	USN-8	As a normal user, I can add warehouses and remove products to them	I can remove products	High	Sprint-3
Notification	USN-9	As a user, I should get mail if certain products count goes below the threshold count specified by me As an admin user, I should get mail if certainproducts count goes below the threshold count specified by me	I should receive notification mail	Medium	Sprint-4

CHAPTER 6

PROJECT PLANNING AND SCHEDULING

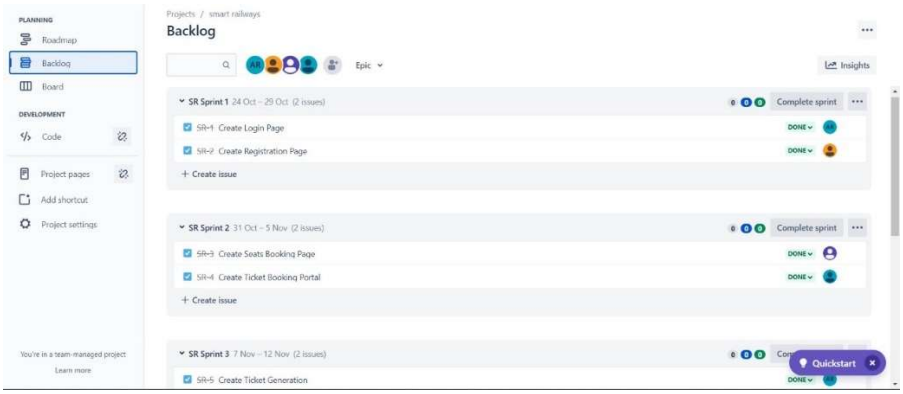
6.1 Sprint Planning and Estimation:

Sprint	Functional Requirement(Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a normal user, I can register for the application by entering my email and password and confirming my password and giving the inventory ID	10	High	Ambarish, Akash
Sprint-1		USN-2	As an admin user, I can register for the application by entering my email and password and confirming my password and giving the inventory Name	10	High	Jeba Regan Raj, Jayanth
Sprint-1	Login and Authentication	USN-3	As a normal user and admin user, I can log into the application by entering email & password	10	High	Jayanth
Sprint-2	Dashboard	USN-4	As a normal user, I can log into my account and access the Dashboard	10	High	Jeba ReganRaj
Sprint-2	Edit Details	USN-5	As a normal user, I can edit my details	10	High	Akash, Ambarish
Sprint-2		USN-6	As an admin user, I can edit my details and change my Inventory name	10	High	Ambarish, Akash
Sprint-3	Management	USN-7	As an admin user, I can add warehouses and add/remove products to them	10	High	Jeba Regan Raj, Jayanth
Sprint-3		USN-8	As a normal user, I can remove products to them	10	High	Jayanth
Sprint-4	Notification	USN-9	As a normal user, I should get mail if certainproducts count goes below the threshold count specified by me As an admin user, I should get mail if certainproducts count goes below the threshold count specified by me	10	Medium	Jeba Regan Raj

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	30	6 Days	24 Oct 2022	29 Oct 2022	30	29 Oct 2022
Sprint-2	30	6 Days	31 Oct 2022	05 Nov 2022	30	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	10	6 Days	14 Nov 2022	19 Nov 2022	10	19 Nov 2022

6.3 Reports from JIRA:



CHAPTER 7

CODING & SOLUTIONING

7.1 FEATURE 1:

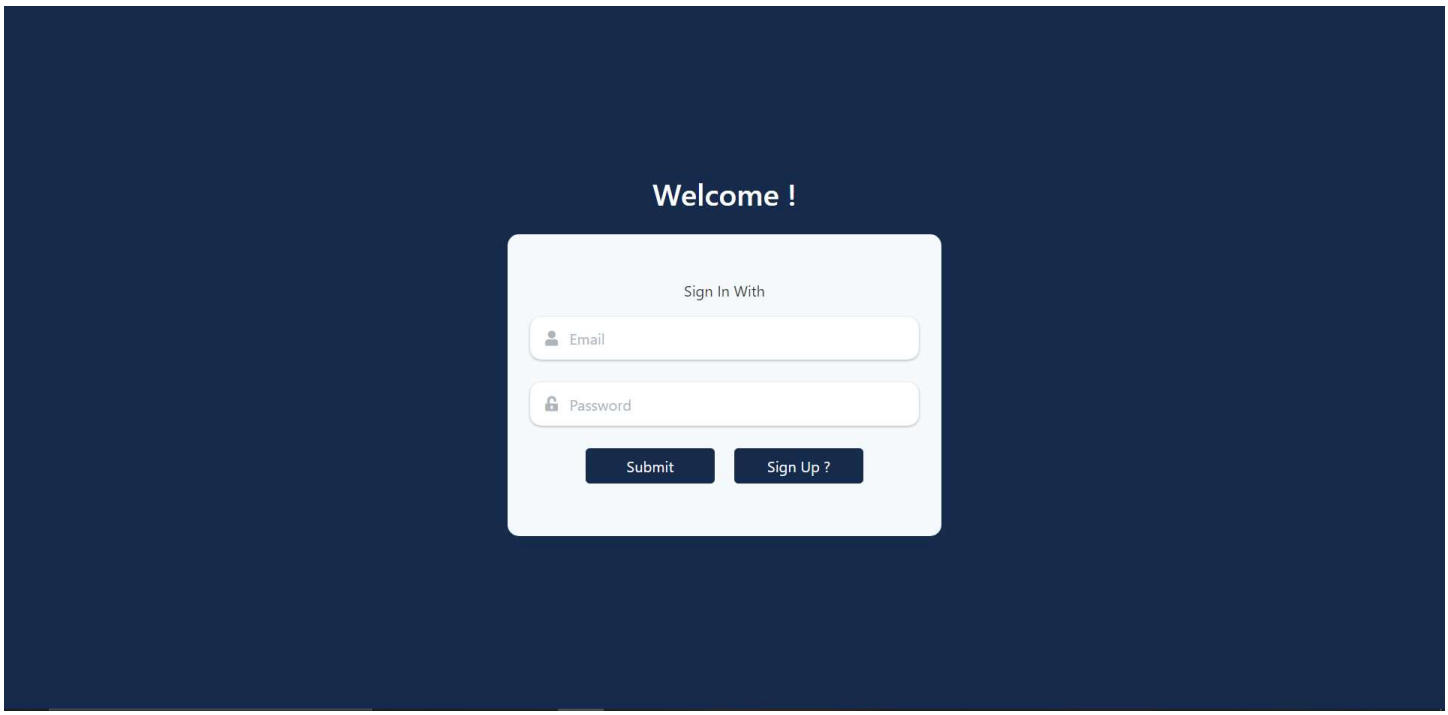


Figure 7.1 : Sign in Page

7.2 FEATURE 2:

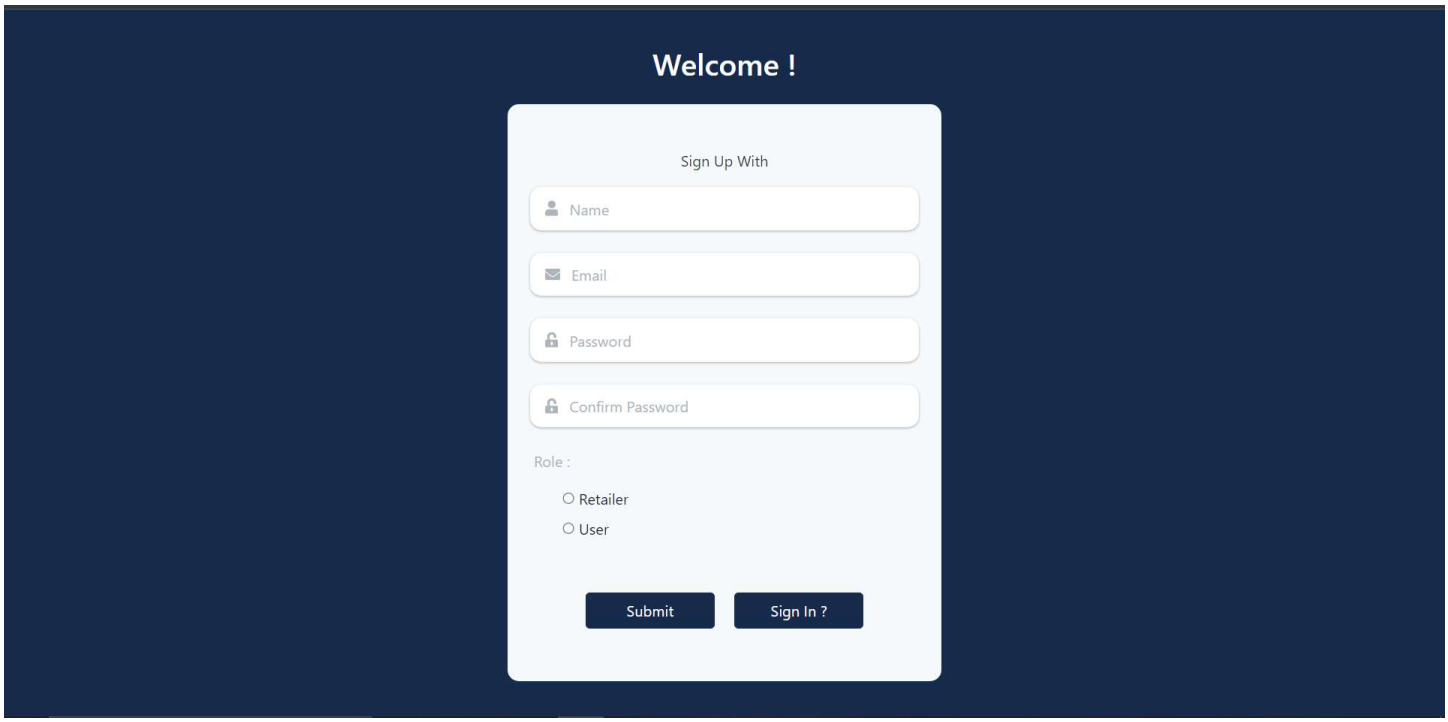


Figure 7.2 : Sign up Page

7.3 FEATURE 3:

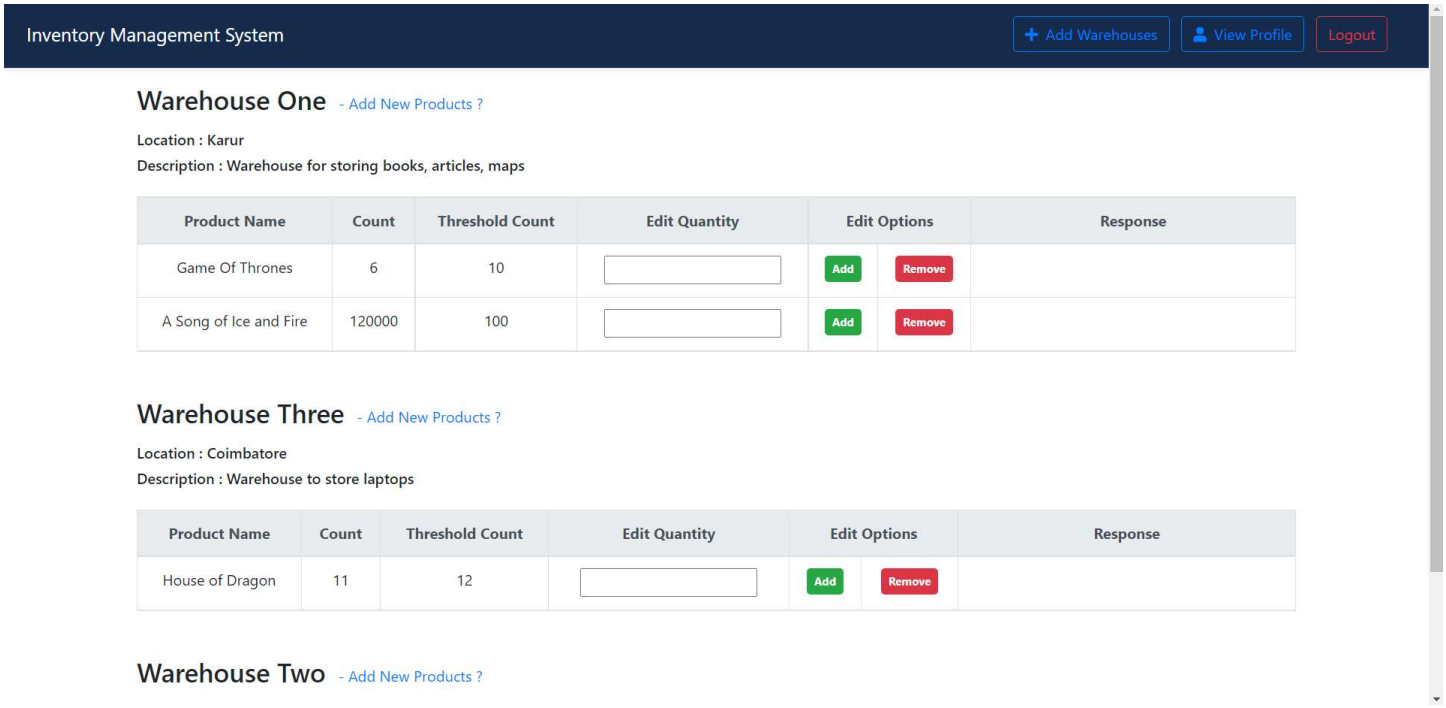


Figure 7.3 : Dashboard Page

7.4 FEATURE 4:

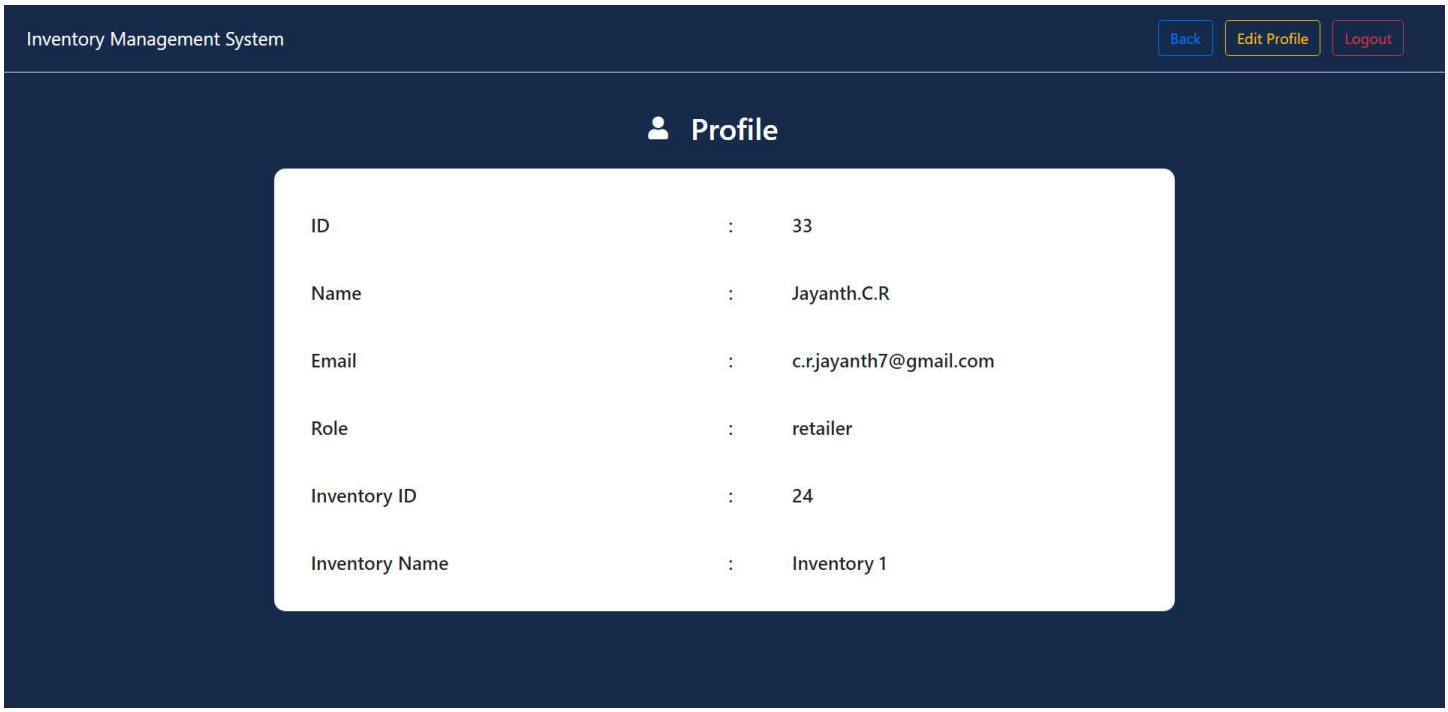


Figure 7.4 : Profile Page

7.5 FEATURE 5:

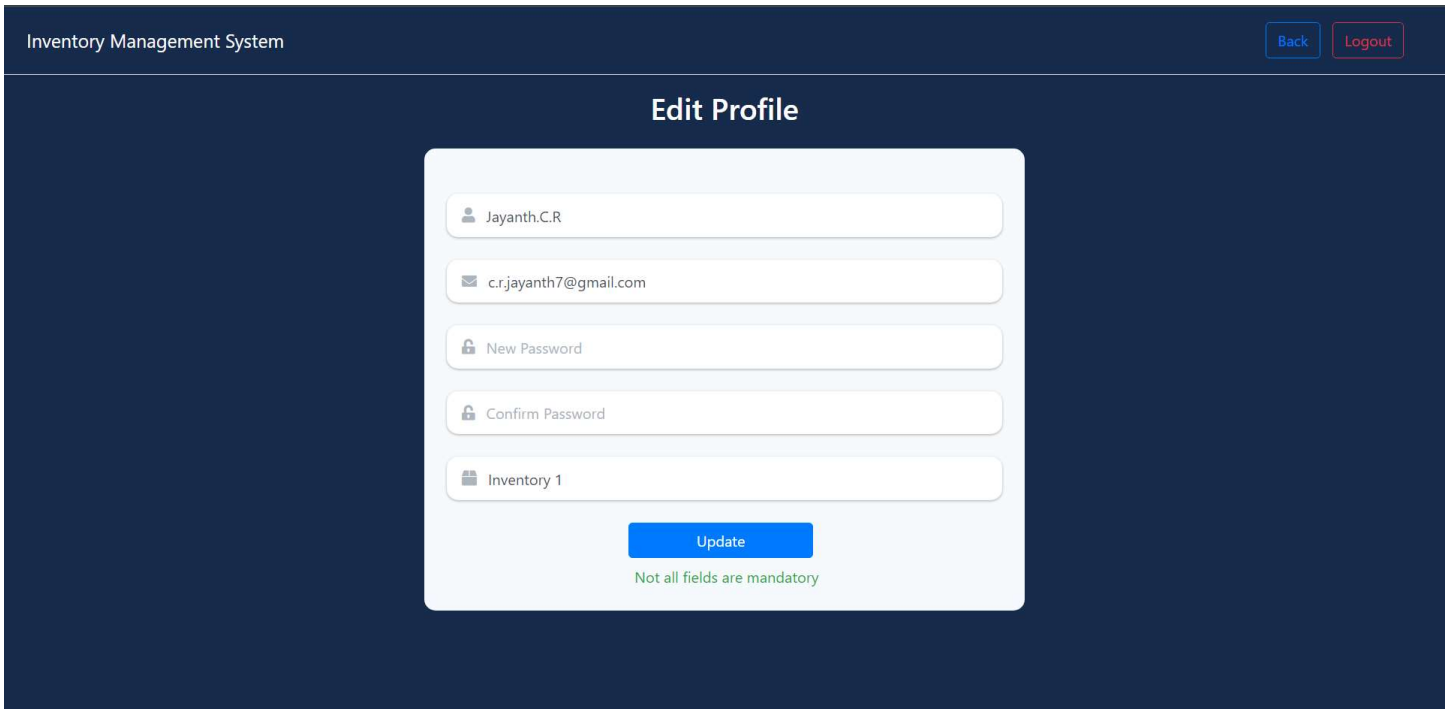


Figure 7.5 : Edit Profile Page

7.6 FEATURE 6:

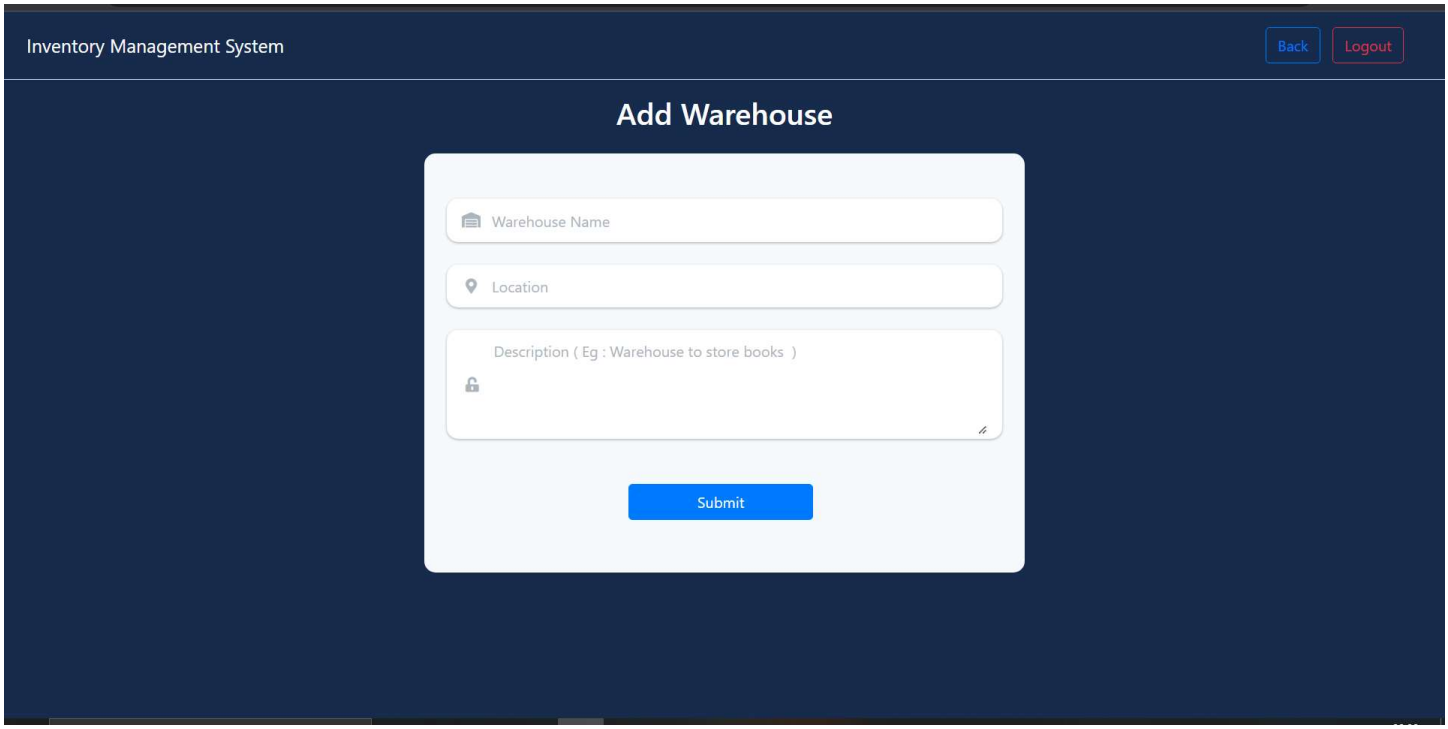
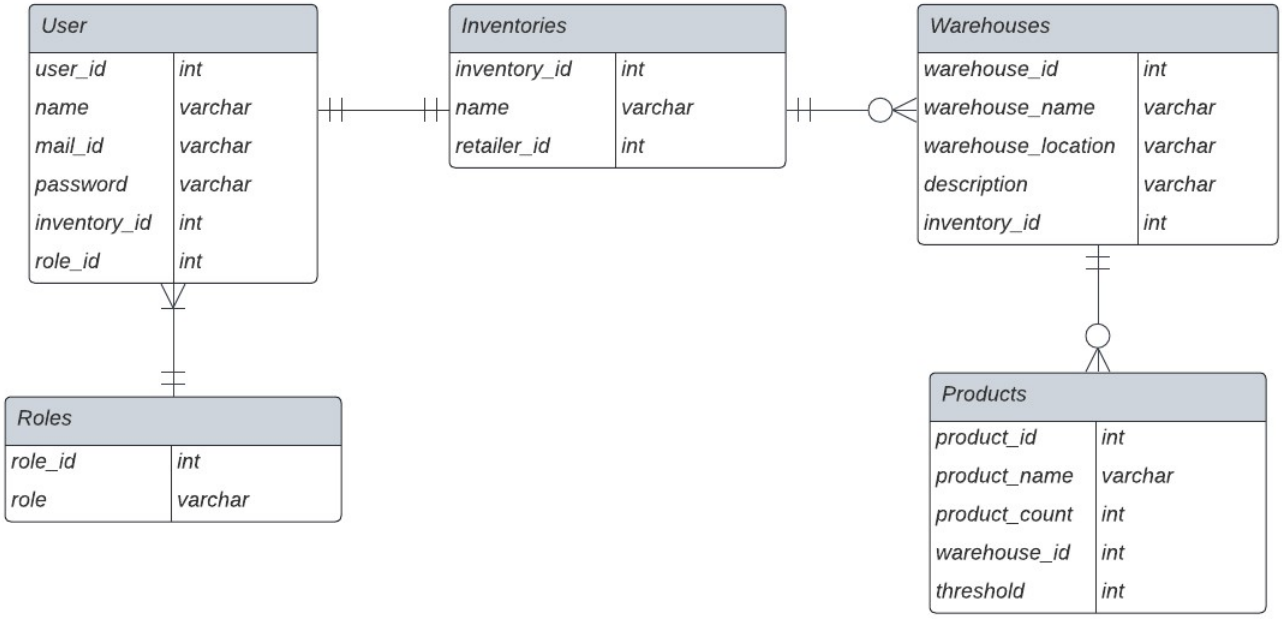


Figure 7.6 : Add Warehouse Page

7.7 DATABASE SCHEMA:



CHAPTER 8

TESTING

8.1 TEST CASES :

Test cases	Result
Verify whether the user is able to see the sign in page	Positive
Verify whether the user is able to go to the sign up page	Positive
Verify whether the user is able to create a new account	Positive
Verify whether the user is able to able choose their preferred role when creation	Positive
Verify whether the user is able to login using email id and password	Positive
Verify whether the user is able to see the dashboard after login	Positive
Verify whether the user is able to view their profile after clicking view profile button	Positive
Verify whether the user is able to edit their profile info after clicking edit profile	Positive
Verify whether the user is able to create a new warehouse by choosing create warehouse	Positive
Verify whether the user is able to add new products to the warehouse	Positive
Verify whether the user is able to view the list of products in the dashboard	Positive
Verify whether the user is able to add/remove the product count	Positive
Verify whether the user receives notification mail when the product count reaches threshold	Postive
Verify whether the user is able to logout	Positive

8.2 USER ACCEPTANCE TESTING :

Test case ID	**Feature Type**	**Component**	**Test Scenario**	**Steps to Execute**
SignUpPage_TC_001	Functional	Sign Up page	Verify the user is able to see the Sign up page when the user clicks the signup button in navigation bar	1. Enter the url and go 2.Click the sign up link in the navigation bar. 3.Verify the sign up page is visible or not.
SignUpPage_TC_002	UI	Sign Up page	Verify the UI elements in the Sign up page	1. Enter the url and go 2.Click the sign up link in the navigation bar. 3.Verify the below mentioned ui elements: a.name text box b. email text box. c. password text box. d. repeat password text box. e. sign up button f. role type radio button

SignUpPage_TC_003	Functional	Sign Up page	Verify the user is able to register into the application by providing valid details	1. Enter the url and go 2.Click the sign up link in the navigation bar. 3.Enter valid details in the text boxes. 4. Verify the confirmation message.
SignInPage_TC_001	Functional	Sign In page	Verify the user is able to see the sign in page when the user clicks the signin button in navigation bar	1. Enter the url and go 2.Click the sign in link in the navigation bar. 3.Verify the sign in page is visible or not.
SignInPage_TC_002	UI	Sign In page	Verify the UI elements in the Sign in page	1. Enter the url and go 2.Click the sign in link in the navigation bar. 3.Verify the below mentioned ui elements: a. email text box. b. password text box. c. sign in button
SignInPage_TC_003	Functional	Sign In page	Verify the user is able to login into the application by providing valid details	1. Enter the url and go 2.Click the sign in link in the navigation bar. 3.Enter valid details in the text boxes. 4. Verify the user is able to login.
DashboardPage_TC_001	Functional	Dashboard	Verify whether the user is able to see the list of products stored in the warehouse	1. Enter the url and go 2.Verify the whether products are visible or not.
DashboardPage_TC_002	UI	Dashboard	Verify the UI elements in the dashboard page	. Enter the url and go 2.Verify the below mentioned ui elements: a. A navbar b. list of table each representing a warehouse location c. view profile button d. add products button e. logout button f. add warehouses button
EditProfilePage_TC_001	Functional	Edit profile page	Verify the user is able to change user details by providing valid details	1. Enter the url and go 2.Enter valid details in the text boxes.

				3. Click the update button. 4. Verify whether the user information is updated sucessfully.
EditProfilePage_TC_002	UI	Edit profile page	Verify the UI elements in the edit profile page	1. Enter the url and go 2.Click the edit profile button in the navigation bar. 3.Verify the below mentioned ui elements: a. name text box b. email text box. c. password text box. d. inventory name text box. e. an update button
AddProductForm_TC_001	Functional	Add Product page	Verify the user is able to add a product to the warehouse	1. Enter the url and go 2.Click the request link near the warehouse name. 3.Enter valid details in the text boxes. 4. Click the add button. 5. Verify whether the product is added sucessfully.
AddWarehouse_TC_001	Functional	Add warehouse page	Verify the user is able to add a warehouse	1. Enter the url and go 2. Go to add awrehouse page. 3. Enter the details and click add button.
Notication_TC_001	Functional	Dashboard	Verify whether the user gets email notification when the product count reached threshold	1. Enter the url and go 2. Go to the dashboard. 3. Remove products so that the product count reaches below threshold level.
Logout_TC_001	Functional	Dashboard	Verify the user is able to logout	1. Enter the url and go 2.Click the logout button

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS:

1. Hours worked : 50 hours
2. Stick to Timelines : 100%
3. Consistency of the product : 75%
4. Efficiency of the product : 80%
5. Quality of the product : 85%

CHAPTER 10

ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

- Easier accesible from anywhere
- Can add more than one warehouse
- Measured pay per use
- Effective management

10.2 DISADVANTAGES

- Works only when internet is on
- Latency may be observed based on the client side machine
- Needs maintenance to ensure scability

CHAPTER 11

CONCLUSION

So, the purpose of this project ergo the main objective was to make a convenient management system software for retailers so that they can keep track of their goods without any hassle and here we have come to the end of our project.
We got to learn many new technologies whilst implementing and this project and it was a great experience.

CHAPTER 12

FUTURE SCOPE

12.1 FUTURE SCOPE :

- Successful companies will view inventory as a strategic asset, rather than an aggravating expense or an evil to be tolerated.
- Collaboration with supply chain partners, coupled with a holistic approach to supply chain management, will be key to effective inventory management.The nature of globalization will change, impacting inventory deployment decisions dramatically.
- Increased focus on supply chain security, and concerns about the quality of inventory itself, will be primary motivators to changing supply chain and inventory strategy.
- The scope of an inventory system can cover many needs, including valuing the inventory, measuring the change in inventory and planning for future inventory levels. The value of the inventory at the end of each period provides

CHAPTER 13

APPENDIX

13.1 SOURCE CODE :

main.py :

```
from flask import *
import re
import os
from dbactions.signup import create_retailer_account,create_user_account
from dbactions.signin import validate_user
from dbactions.profile import get_user_profile_details, update_profile
from dbactions.addwarehouse import add_new_warehouse
from dbactions.dashboard import get_dashboard_details
from dbactions.products import add_product,edit_product_count
from flask_mail import Mail
from dotenv import load_dotenv

app = Flask(__name__)
mail = Mail(app)

load_dotenv()
app.config['MAIL_SERVER']= os.getenv('MAIL_SERVER')
app.config['MAIL_PORT'] = os.getenv('MAIL_PORT')
app.config['MAIL_USERNAME'] = os.getenv('MAIL_USERNAME')
app.config['MAIL_PASSWORD'] = os.getenv('MAIL_PASSWORD')
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)

@app.route('/signin',methods = ['POST', 'GET'])
def sign_in():
    if request.method == 'GET':
        mail_id = request.cookies.get('mail_id')
        if mail_id != None:
            return redirect("http://127.0.0.1:5000/dashboard",code=302)
        else:
            return render_template('signin.html')
    elif  request.method == 'POST':
        response = validate_user(request.json['mail_id'],request.json['password'])
        resp = make_response(response)
        if response['status']:
            resp.set_cookie("mail_id", request.json['mail_id'])
            resp.set_cookie("role",'retailer' if response['role_id'] == 1 else 'user')
        return(resp)
```

```
@app.route('/signup',methods = ['POST', 'GET'])
def sign_up():
    if request.method == 'GET':
        mail_id = request.cookies.get('mail_id')
        if mail_id != None:
            return redirect("http://127.0.0.1:5000/dashboard",code=302)
        else:
            return render_template('signup.html')
    elif  request.method == 'POST':
        if request.json['role'] == 'retailer':
            response =
create_retailer_account(request.json['name'],request.json['mail_id'],request.json['password'],request.json['invent
ory_id_or_name'])
        else:
            response =
create_user_account(request.json['name'],request.json['mail_id'],request.json['password'],request.json['inventory
_id_or_name'])
        resp = make_response(response)
        if response['status']:
            resp.set_cookie("mail_id", request.json['mail_id'])
            resp.set_cookie("role",request.json['role'])
        return(resp)


@app.route('/logout',methods = ['GET'])
def logout():
    if request.method == 'GET':
        resp = make_response(redirect("http://127.0.0.1:5000/signin",code=302))
        resp.set_cookie('mail_id', '', expires=0)
        return(resp)

@app.route('/dashboard',methods = ['GET'])
def dashboard():
    if request.method == 'GET':
        mail_id = request.cookies.get('mail_id')
        if mail_id != None:
            user_info = {}
            warehouses_info = {}
            user_info_response = get_user_profile_details(mail_id)
            warehouses_info_response = {}
            if user_info_response.get('status'):
                user_info = user_info_response['user_info']
                warehouses_info_response = get_dashboard_details(user_info['inventory_id'])
            if warehouses_info_response.get('status'):
                warehouses_info = warehouses_info_response['warehouses_info']
            return
render_template("dashboard.html",user_info=user_info,mail_id=user_info['mail_id'],warehouses_info=warehou
ses_info)

        else:
```

```
            return redirect("http://127.0.0.1:5000/signin",code=302)

@app.route('/dashboard/addwarehouse',methods = ['GET','POST'])
def add_new_warehouse():
    mail_id = request.cookies.get('mail_id')
    role = request.cookies.get('role')
    if mail_id != None:
        if role == 'retailer':
            if request.method == 'GET':
                return render_template("addwarehouse.html")
            elif  request.method == 'POST':
                response =
add_new_warehouse(mail_id,request.json['warehouse_name'],request.json['location'],request.json['description'])
                return(response)
        else:
            return redirect("http://127.0.0.1:5000/dashboard",code=302)
    else:
        return redirect("http://127.0.0.1:5000/signin",code=302)


@app.route('/dashboard/addproduct',methods = ['GET','POST'])
def add_new_product():
    mail_id = request.cookies.get('mail_id')
    role = request.cookies.get('role')
    if mail_id != None:
        warehouse_id = request.args.get('warehouse_id')
        if role == 'retailer' :
            if request.method == 'GET':
                warehouse_id_regex = re.compile(r'\d+')
                if  warehouse_id and warehouse_id_regex.search(warehouse_id) :
                    return render_template("addProducts.html",warehouse_id=warehouse_id)
                else:
                    return redirect("http://127.0.0.1:5000/dashboard",code=302)
            elif  request.method == 'POST':
                response =
add_product(mail_id,int(request.json['warehouse_id']),request.json['product_name'],request.json['count'],request
.json['threshold'])
                return(response)
        else:
            return redirect("http://127.0.0.1:5000/dashboard",code=302)
    else:
        return redirect("http://127.0.0.1:5000/signin",code=302)

@app.route('/dashboard/editproductdetails',methods = ['POST'])
def edit_product_details():
    mail_id = request.cookies.get('mail_id')
    if mail_id != None:
        if request.method == 'POST':
            response =
edit_product_count(request.json['inventory_id'],request.json['product_id'],int(request.json['count']),request.json[
```

```
'action'],mail)
                return(response)
            else:
                return redirect("http://127.0.0.1:5000/dashboard",code=302)
        else:
            return redirect("http://127.0.0.1:5000/signin",code=302)


@app.route('/dashboard/profile',methods = ['GET'])
def profile():
    if request.method == 'GET':
        mail_id = request.cookies.get('mail_id')
        if mail_id != None:
            response = get_user_profile_details(mail_id)
            return(render_template('profile.html',response=response['user_info'],reason=response['reason']))
        else:
            return redirect("http://127.0.0.1:5000/signin",code=302)

@app.route('/dashboard/profile/editprofile',methods = ['GET','POST'])
def edit_profile():
    mail_id = request.cookies.get('mail_id')

    if mail_id == None:
        return redirect("http://127.0.0.1:5000/signin",code=302)
    else:
        if request.method == 'GET':
            response = get_user_profile_details(mail_id)
            return render_template('editprofile.html',response=response['user_info'],reason=response['reason'])
        elif request.method == 'POST':
            response = update_profile(request.json['current_mail_id'],request.json)
            resp = make_response(response)
            if request.json.get('mail_id'):
                resp.set_cookie("mail_id", request.json.get('mail_id'))
            return resp


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

connection.py :

```
import ibm_db
import os
from dotenv import load_dotenv

def getConnection():
   conn = False
   try:
      conn = ibm_db.connect(os.getenv('DB2_CREDENTIALS'), '', '')
   except Exception as e:
```

```
        print("Exception while opening connection :",e)
        print(ibm_db.conn_errormsg())
    return(conn)

def closeConnection(conn):
    try:
        ibm_db.close(conn)
        return(True)
    except Exception as e:
        print("Exception while closing connection :",e)
        return(False)
```

dashboard.py :

```
from dbactions.connection import getConnection,closeConnection
import ibm_db

def get_dashboard_details(inventory_id):
    response =  get_warehouses_info(inventory_id)
    return(response)


def get_warehouses_info(inventory_id):
    conn = getConnection()
    warehouses_info = {}
    if conn:
        try:
            query =  '''SELECT
                        INVENTORIES.INVENTORY_ID,
                        WAREHOUSES.WAREHOUSE_ID,
                        WAREHOUSES.WAREHOUSE_NAME,
                        WAREHOUSES.WAREHOUSE_LOCATION,
                        WAREHOUSES.DESCRIPTION,
                        PRODUCTS.PRODUCT_ID,
                        PRODUCTS.PRODUCT_NAME,
                        PRODUCTS.PRODUCT_COUNT,
                        PRODUCTS.THRESHOLD_COUNT
                    FROM
                        INVENTORIES INNER JOIN WAREHOUSES
                            ON INVENTORIES.INVENTORY_ID = WAREHOUSES.INVENTORY_ID
                                LEFT JOIN PRODUCTS
                                    ON WAREHOUSES.WAREHOUSE_ID = PRODUCTS.WAREHOUSE_ID
                    WHERE INVENTORIES.INVENTORY_ID = ?
                    '''
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,inventory_id)
            ibm_db.execute(statement)
            result = ibm_db.fetch_both(statement)
            while result:
```

```
            if warehouses_info.get(result[1]):
                warehouses_info[result[1]]['products'].update({
                    result[5]:{
                            'product_id': result[5],
                            'product_name': result[6],
                            'product_count': result[7],
                            'threshold_count': result[8],
                    }
                })
            else:
                warehouses_info[result[1]] = {
                    'warehouse_id':result[1],
                    'warehouse_name' : result[2],
                    'location' : result[3],
                    'description' : result[4],
                    'products':{

                        }
                    }
                if result[5]:
                    warehouses_info[result[1]]['products'] = {
                        result[5]:{
                            'product_id': result[5],
                            'product_name': result[6],
                            'product_count': result[7],
                            'threshold_count': result[8],
                        }
                    }

            result = ibm_db.fetch_both(statement)
        except Exception as e:
            print("Exception while getting dashboard details : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong" }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" ,'warehouses_info':warehouses_info}
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB"}
```

addwarehouse.py :

```
from dbactions.connection import getConnection,closeConnection
from dbactions.profile import get_user_info
import ibm_db

def add_new_warehouse(mail_id,warehouse_name,location,description):
    response = get_user_info(mail_id)
    if response['status']:
        inventory_id = response['user_info']['inventory_id']
```

```
    conn = getConnection()
    if conn:
        try:
            query = "INSERT INTO WAREHOUSES
(WAREHOUSE_NAME,WAREHOUSE_LOCATION,DESCRIPTION,INVENTORY_ID) VALUES(?,?,?,?)"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,warehouse_name)
            ibm_db.bind_param(statement, 2,location)
            ibm_db.bind_param(statement, 3,description)
            ibm_db.bind_param(statement, 4,inventory_id)
            ibm_db.execute(statement)
            if ibm_db.num_rows(statement) != 1:
                return{ 'status' : False, 'reason' : "Something went wrong" }
        except Exception as e:
            print("Exception while creating warehouse : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong"}
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" }
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB" }
  return response
```

products.py :

```
from dbactions.connection import getConnection,closeConnection
import ibm_db
import os
from dotenv import load_dotenv
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from flask_mail import Message

def add_product(mail_id,warehouse_id,product_name,count,threshold):
  response = get_warehouse_ids_and_product_names(mail_id)
  if response['status']:
    warehouse_ids = response['warehouse_ids']
    product_names = response['product_names']
    if warehouse_id in warehouse_ids:
      if product_name.lower() in product_names:
        return { 'status' : False, 'reason' : 'Product name already exists' }
      else:
        response = add_new_product(warehouse_id,product_name,count,threshold)
        return(response)
    else:
      return { 'status' : False, 'reason' : "Warehouse ID doesn't exists"  }
  return response

def edit_product_count(inventory_id,product_id,count,action,flask_mail_object):
```

```
    response = get_product_count_and_threshold_count(product_id)
    if response['status']:
        count = count if action == 'add' else -1*count
        threshold_count = response['threshold_count']
        new_count = response['product_count'] + count
        if new_count >= 0:
            response = change_product_count(product_id,new_count)
            if response['status']:
                if new_count < threshold_count:
                    send_alert_mail(inventory_id,new_count,threshold_count,product_id,flask_mail_object)
                return({ 'status':True,'reason':'','new_count':new_count })
            return(response)
        else:
            return { 'status':False,'reason':"Invalid quantity" }
    return(response)

def get_warehouse_ids_and_product_names(mail_id):
    conn = getConnection()
    warehouse_ids = []
    product_names = []
    if conn:
        try:
            query = "SELECT
WAREHOUSES.WAREHOUSE_ID,PRODUCTS.PRODUCT_NAME,USERS.MAIL_ID FROM  USERS
INNER JOIN INVENTORIES ON USERS.USER_ID = INVENTORIES.RETAILER_ID INNER JOIN
WAREHOUSES ON INVENTORIES.INVENTORY_ID = WAREHOUSES.INVENTORY_ID LEFT JOIN
PRODUCTS ON WAREHOUSES.WAREHOUSE_ID = PRODUCTS.WAREHOUSE_ID WHERE
USERS.MAIL_ID = ?"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement,1,mail_id)
            ibm_db.execute(statement)
            result = ibm_db.fetch_both(statement)
            while(result):
                warehouse_ids.append(result[0])
                if result[1]:
                    product_names.append(result[1].lower())
                result = ibm_db.fetch_both(statement)
        except Exception as e:
            print("Exception while getting warehouse ids and product names from DB : ",e)
            return{ 'status' : False, 'reason' : "Something went
wrong",'warehouse_ids':warehouse_ids,"product_names":product_names }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "",'warehouse_ids':warehouse_ids,"product_names":product_names }
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to
DB",'warehouse_ids':warehouse_ids,"product_names":product_names}

def add_new_product(warehouse_id,product_name,count,threshold):
    conn = getConnection()
```

```
    if conn:
        try:
            query = "INSERT INTO PRODUCTS
(WAREHOUSE_ID,PRODUCT_NAME,PRODUCT_COUNT,THRESHOLD_COUNT) VALUES(?,?,?,?)"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,warehouse_id)
            ibm_db.bind_param(statement, 2,product_name)
            ibm_db.bind_param(statement, 3,count)
            ibm_db.bind_param(statement, 4,threshold)
            ibm_db.execute(statement)
            if ibm_db.num_rows(statement) != 1:
                return{ 'status' : False, 'reason' : "Something went wrong" }
        except Exception as e:
            print("Exception while adding new products : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong" }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" }
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB"}


def get_product_count_and_threshold_count(product_id):
    conn = getConnection()
    product_count = ''
    threshold_count = ''
    if conn:
        try:
            query = "SELECT PRODUCT_COUNT,THRESHOLD_COUNT FROM PRODUCTS WHERE
PRODUCT_ID = ?"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,product_id)
            ibm_db.execute(statement)
            result = ibm_db.fetch_both(statement)
            while result:
                product_count = result[0]
                threshold_count = result[1]
                result = ibm_db.fetch_both(statement)
        except Exception as e:
            print("Exception while getting products count : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong" , 'product_count':product_count,
'threshold_count':threshold_count }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" ,'product_count':product_count, 'threshold_count':threshold_count}
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB" , 'product_count':product_count ,
'threshold_count':threshold_count}

def change_product_count(product_id,new_count):
```

```
    conn = getConnection()
    if conn:
        try:
            query = "UPDATE PRODUCTS SET PRODUCT_COUNT = ? WHERE PRODUCT_ID = ?"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,new_count)
            ibm_db.bind_param(statement, 2,product_id)
            ibm_db.execute(statement)
        except Exception as e:
            print("Exception while getting products count : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong"  }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" }
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB"  }

def send_alert_mail(inventory_id,current_count,threshold_count,product_id,flask_mail_object):
    response = get_receiver_mail_ids(inventory_id)
    if response['status']:
        receiver_mail_ids = response['receiver_mail_ids']
        response = get_product_details(product_id)
        if response['status']:

send_notification_via_flaskmail(receiver_mail_ids,current_count,threshold_count,response['product_name'],res
ponse['warehouse_name'],flask_mail_object)

send_notification_via_sendgrid(receiver_mail_ids,current_count,threshold_count,response['product_name'],res
ponse['warehouse_name'])
    return(response)


def get_receiver_mail_ids(inventory_id):
    conn = getConnection()
    receiver_mail_ids = []
    if conn:
        try:
            query = "SELECT MAIL_ID FROM USERS WHERE INVENTORY_ID = ?"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,inventory_id)
            ibm_db.execute(statement)
            result = ibm_db.fetch_both(statement)
            while result:
                receiver_mail_ids.append(result[0])
                result = ibm_db.fetch_both(statement)
        except Exception as e:
            print("Exception while getting receiver mail ids for notification : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong" , 'receiver_mail_ids':receiver_mail_ids }
        finally:
            closeConnection(conn)
```

```
        return{ 'status' : True, 'reason' : "" ,'receiver_mail_ids':receiver_mail_ids}
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB" , 'receiver_mail_ids':receiver_mail_ids}

def get_product_details(product_id):
    conn = getConnection()
    product_name = ""
    warehouse_name = ""
    if conn:
        try:
            query = "SELECT WAREHOUSES.WAREHOUSE_NAME,PRODUCTS.PRODUCT_NAME FROM
PRODUCTS INNER JOIN WAREHOUSES ON WAREHOUSES.WAREHOUSE_ID =
PRODUCTS.WAREHOUSE_ID WHERE PRODUCT_ID = ?"
            statement = ibm_db.prepare(conn,query)
            ibm_db.bind_param(statement, 1,product_id)
            ibm_db.execute(statement)
            result = ibm_db.fetch_both(statement)
            while result:
                warehouse_name = result[0]
                product_name = result[1]
                result = ibm_db.fetch_both(statement)
        except Exception as e:
            print("Exception while getting product details for sending mail : ",e)
            return{ 'status' : False, 'reason' : "Something went wrong" ,
'product_name':product_name,'warehouse_name':warehouse_name }
        finally:
            closeConnection(conn)
        return{ 'status' : True, 'reason' : "" , 'product_name':product_name,'warehouse_name':warehouse_name}
    else:
        return{ 'status' : False, 'reason' : "Couldn't connect to DB" ,
'product_name':product_name,'warehouse_name':warehouse_name}


def
send_notification_via_sendgrid(receiver_mail_ids,current_count,threshold_count,product_name,warehouse_na
me):
    message = Mail(
    from_email=os.getenv('SENDER_MAIL_ID'), # sender mail ID
    #to_emails=[receiver_mail_ids],
    to_emails=receiver_mail_ids,
    subject='Test Mail sendgrid',
    html_content='<strong>Test Content</strong>')
    try:
        sg = SendGridAPIClient(os.getenv('SENDGRID_API_KEY'))
        sg.send(message)
    except Exception as e:
        print("Exception while sending alert mail via sendgrid : ",e)

def
send_notification_via_flaskmail(receiver_mail_ids,current_count,threshold_count,product_name,warehouse_na
```

```
me,flask_mail_object):
    msg = Message(
                'Inventory Management - Notification Mail',
                sender =os.getenv('SENDER_MAIL_ID'),
                recipients = receiver_mail_ids
               )
    msg.body = "Hello , This is to notify that a product count has decreased below the threshold limit . Kindly note it down.\n1. Warehouse Name : {}\n2. Product Name : {}\n3. Product's Current Count : {}\n4. Threshold Count : {}".format(warehouse_name,product_name,current_count,threshold_count)
    flask_mail_object.send(msg)
    return 'Sent'
```

13.2 GITHUB LINK :

https://github.com/IBM-EPBL/IBM-Project-24139-1659938595