Inventory Management System for Retailers

Submitted By

YUVARAJ N (812019205060)

ABU RIJAS S (812019205060)

MOHAMMED UNISH(812019205060)

SYED APSAR (812019205060)

TEAM ID : PNT2022TMID45253

In partial fulfillment of the requirement for the degree of

Bachelor of Engineering

(Information Technology)

Table of Contents

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

a. Purpose

Products are considered as the business resources for the organization. This includes managing the product with appropriate way to review any time as per the requirement. Therefore it is important to have a computer based IMS which has the ability to generate reports, maintain the balance of the stock, details about the purchase andsales in the organization. Before developing this application we came up with 2 Inventory Management System existing in the market, which helps to give the knowledge for the development of our project. These application software are only used by the large organization but so we came up with the application which can be used by the small company for the management of their stock in the production houses.

After analyzing the other inventory management system we decided to include some of common and key features that should be included in every inventory management system. So we decided to include those things that help the small organization in a way or other.

1. LiteratureSurvey

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:

1. To make desktop based application of IMSR for small organization.
2. To make the system easily managed and can be secured.
3. To cover all the areas of IMSR like purchase details, sales details and stock management.
4. To fulfill the requirement for achieving the Bachelor’s degree of Computer Information System.
5. To know the fundamentals of the .Net Technology and Visual Studio with the Net Framework.

2.1

a. Existing Problem

Monitor and Categorize Products: Production monitoring gives you a complete picture of the manufacturing process, alerts you to any faults early on, and helps you avoid delivery delays. And when you understand and control your production process from beginning to end, only two things can happen higher product quality and happier consumers.

Manage Product Pricing: Managing Product Price is a set of tools, procedures, methods, and cultures used by companies that create and manufacture products to ensure that their profit (or cost) targets are met.

Manage Reservation and Orders: They must manage reservations and maintain correct records to ensure that all rooms are filled as soon as feasible. This job frequently collaborates with the managers of the revenue and rooms divisions.

Revenue and Expenses Management: Revenue and management aid in the prediction of consumer demand in order to maximize revenue growth by optimizing inventory and price availability. The goal of revenue management isn't to sell a product for a cheap price today in order to offer it for a greater price tomorrow.

2.2

b. 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

2.3 Problem Statement Definition

Problem Statement: Demand is frequently unpredictable in inventory systems, and lead times can often vary. Managers frequently keep a safety supply to minimize shortages. In such cases, it's difficult to say what order amounts and reorder points will result in the lowest total inventory cost.

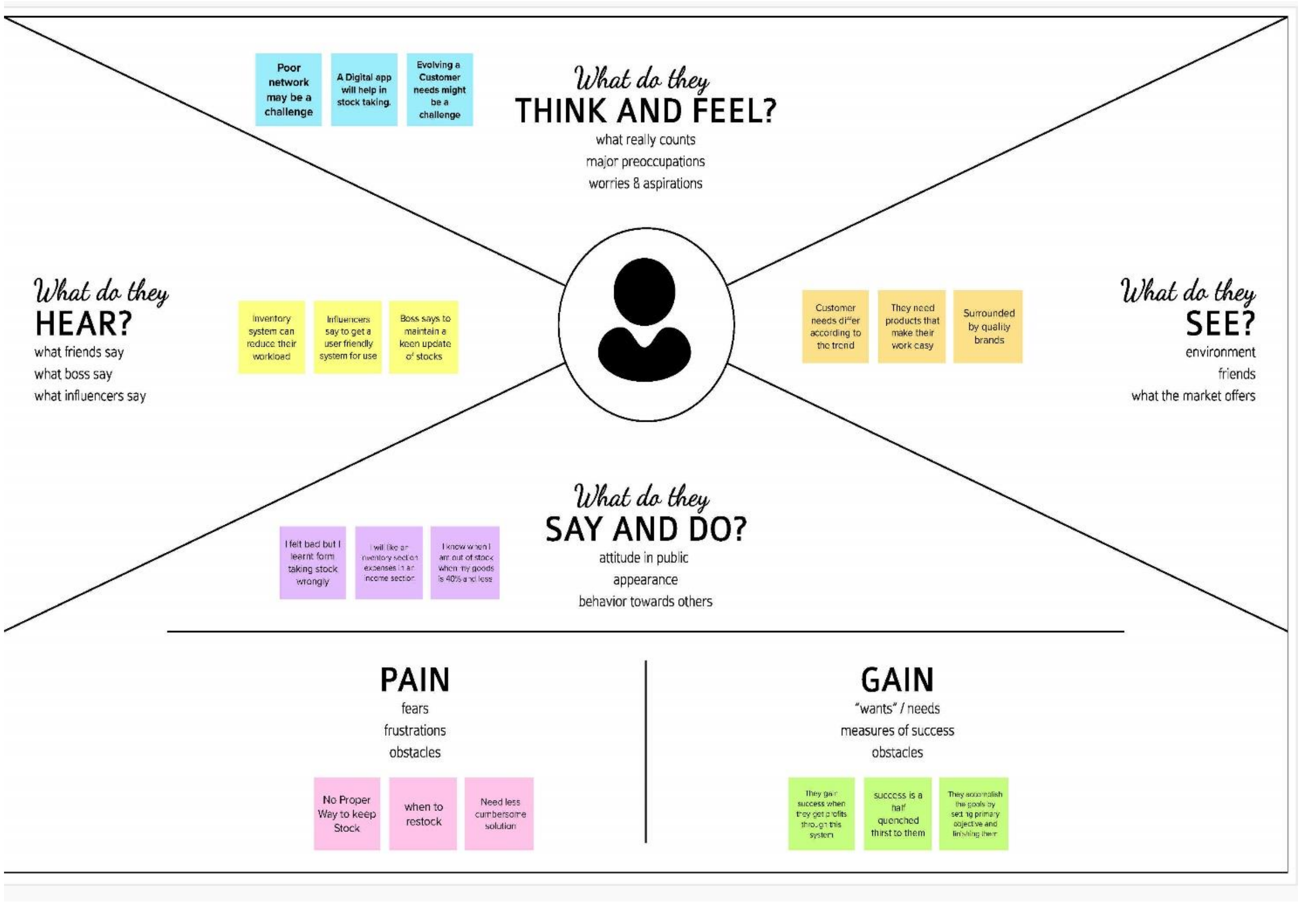
The inventory issue refers to the general issue of deciding how much inventory to keep on hand in expectation of possible demand. Loss occurs when a business is unable to meet demand (for example, when a store loses sales or when soldiers in a war run out of ammunition) or when commodities are stocked for which there is no demand.

Solution: Inventory management's major goal is to make ordering, stocking, storing, and using inventory as simple and efficient as possible for firms. You'll always know what things are in stock, how many there are, and where they are if you manage your inventory properly. An inventory system's main function is to keep track of your products and supplies.

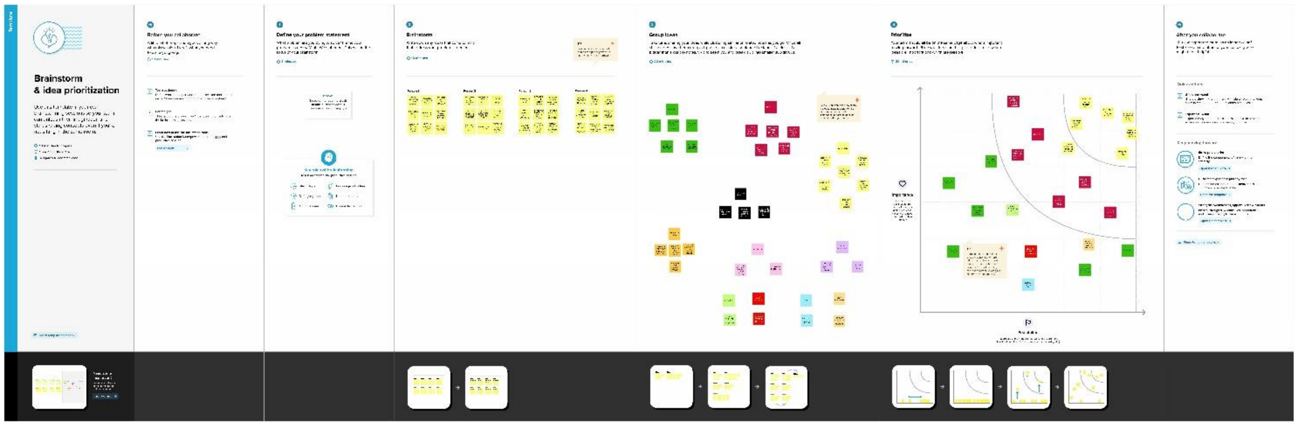
When you buy inventory, you need to keep track of when you bought it, when you sold it, and how much you have on hand. It also tells you where your inventory is.

1. Ideation & Proposed Solution

a. Empathy map canvas


Poor network may be a challenge
A Digital app will help in stock taking.
Evolving a Customer needs might be a challenge
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
What do they HEAR?
what friends say
what boss say
what influencers say
Inventory system can reduce their workload
Influencers say to get a user friendly system for use
Boss says to maintain a keen update of stocks
Customer needs differ according to the trend
They need products that make their work easy
Surrounded by quality brands
What do they SEE?
environment
friends
what the market offers
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
I felt bad but I learnt form taking stock wrongly
I will like an inventory section expenses in an income section
I know when I am out of stock when my goods is 40% and less
PAIN
fears
frustrations
obstacles
No Proper Way to keep Stock
when to restock
Need less cumbersome solution
GAIN
"wants" / needs
measures of success
obstacles
They gain success when they get profits through this system
success is a half quenched thirst to them
They accomplish the goals by setting primary objective and finishing them

3.2 Ideation brainstorming



3.3 a. Proposed solution

1· The customers are not satisfied with the retailers store since it doesn't have enough supplements and the deliveries were not made on time

2. Idea / Solution description

The product availability is tracked daily and an alert system in again kept on to indicate those products which falls below the threshold limit.

· All the customers can register their accounts after which they will be given a login credentials which they can use whenever they feel like buying the stocks.

. The application allows the customers to know all the present time available stocks and also when the new stock will be available on the store for them to buy

3. Novelty / Uniqueness

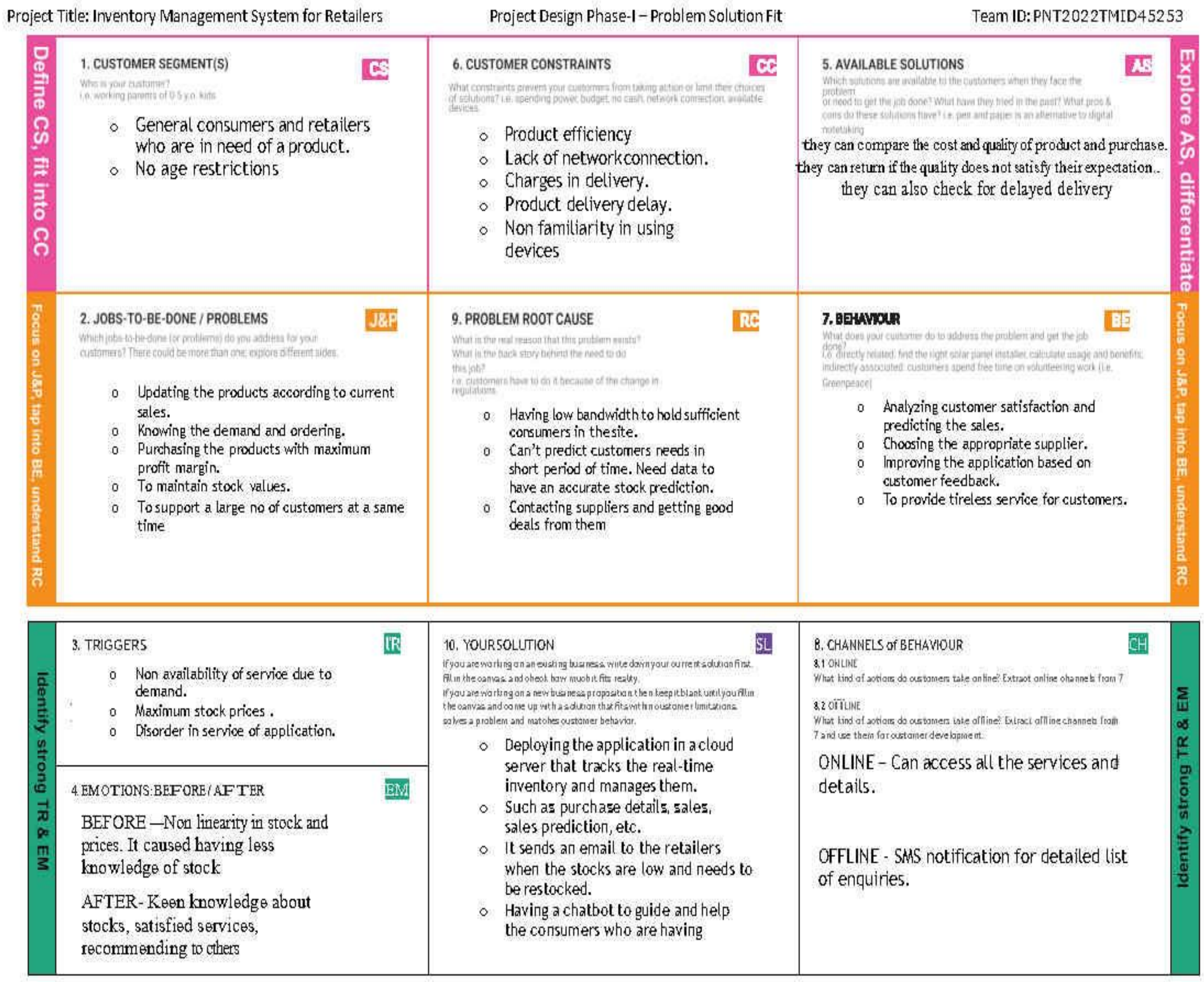
· Prediction of the best selling brand of all certain products based on their popularity, price and customer trust and satisfaction will be implemented.

4. Social Impact / Customer Satisfaction

· The customer satisfaction will be improved for getting appropriate response from the retailers and that too immediately.

5. Business Model (Revenue Model) ML algorithms for all the prediction purposes using all the past dataset since datasets are undoubtedly available in huge amounts. Can deploy the most appropriate business advertising models.

3.4 Problem solution fit



Project Title: Inventory Management System for Retailers

Project Design Phase-I – Problem Solution Fit

Team ID: PNT2022TMID45253

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

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

- General consumers and retailers who are in need of a product.
- No age restrictions

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.

- Product efficiency
- Lack of networkconnection.
- Charges in delivery.
- Product delivery delay.
- Non familiarity in using devices

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

they can compare the cost and quality of product and purchase.
they can return if the quality does not satisfy their expectation..
they can also check for delayed delivery

Explore AS, differentiate

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

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.

- Updating the products according to current sales.
- Knowing the demand and ordering.
- Purchasing the products with maximum profit margin.
- To maintain stock values.
- To support a large no of customers at a same time

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.

- Having low bandwidth to hold sufficient consumers in thesite.
- Can't predict customers needs in short period of time. Need data to have an accurate stock prediction.
- Contacting suppliers and getting good deals from them

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)

- Analyzing customer satisfaction and predicting the sales.
- Choosing the appropriate supplier.
- Improving the application based on customer feedback.
- To provide tireless service for customers.

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

Identify strong TR & EM

3. TRIGGERS TR

- Non availability of service due to demand.
- Maximum stock prices .
- Disorder in service of application.

4. EMOTIONS: BEFORE / AFTER EM

BEFORE —Non linearity in stock and prices. It caused having less knowledge of stock

AFTER- Keen knowledge about stocks, satisfied services, recommending to others

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behavior.

- Deploying the application in a cloud server that tracks the real-time inventory and manages them.
- Such as purchase details, sales, sales prediction, etc.
- It sends an email to the retailers when the stocks are low and needs to be restocked.
- Having a chatbot to guide and help the consumers who are having

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from 7

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

ONLINE – Can access all the services and details.

OFFLINE - SMS notification for detailed list of enquiries.

Identify strong TR & EM

4. Requirement Analysis

4.1. Functional Requirements

The goal for the application is to manage the inventory management function of the organization. Once it is automated all the functions can be effectively managed and the organization can achieve the competitive advantage. Business requirement are discussed in the Scope section, with the following additional details:

1.Helps to search the specific product and remaining stock.

2.Details information about the product sales and purchase.

3.Brief Information of the organization today's status in terms of news, number of present inventory as per the date entered.

4.It helps to identify the total presented inventory in the company

4.2. Non-Functional Requirements

1. Able to edit the entry as per entry.

2.Able to add, modify and delete the stock entry.

3. Able to check the stock available.

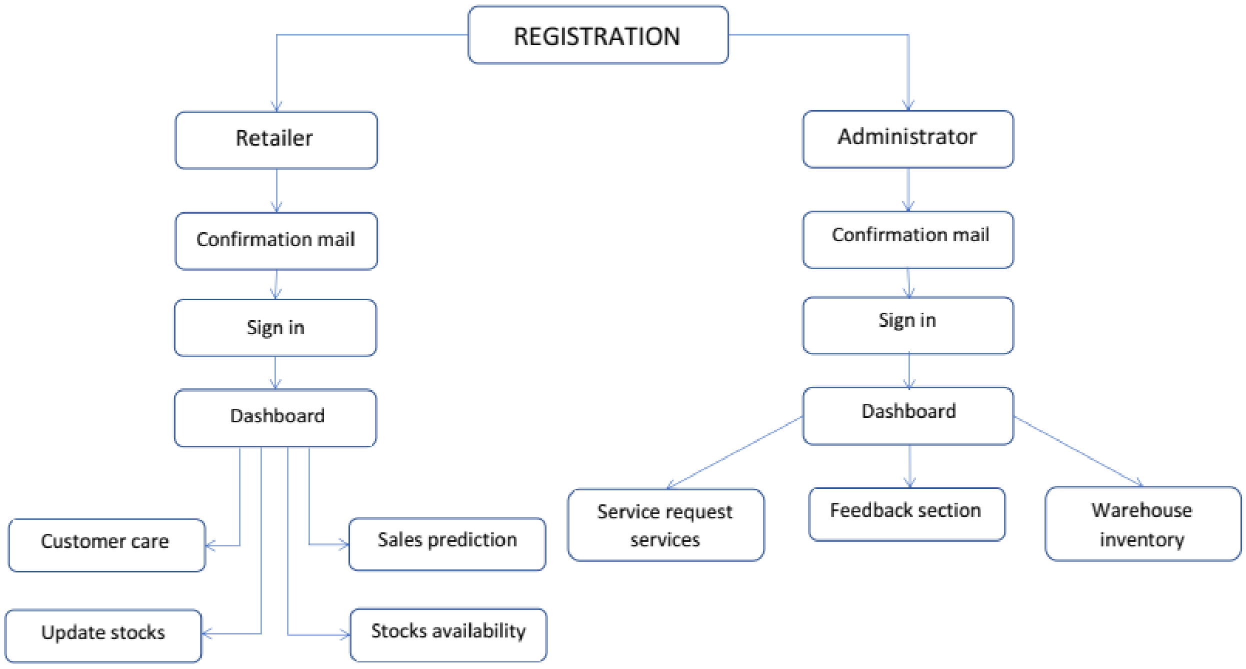
4. Able to check the balance payment.

5.Able to view the remaining sales stock.

5. Project Design

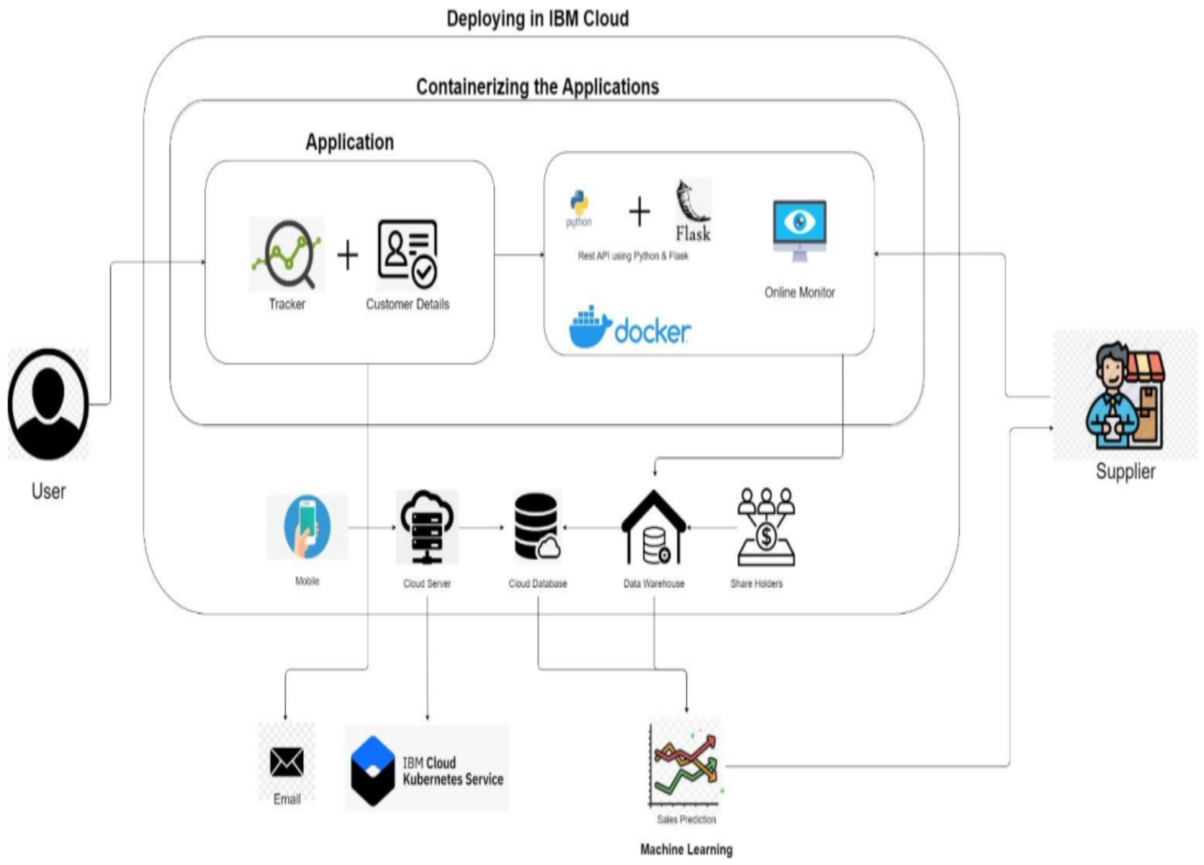
5.1 Data flow Diagram

Process Flow Diagram or Flowchart is a diagram which uses geometric symbols and arrows to define the relationships. It is a diagrammatic representation of the algorithm. The Process flow Diagram of our application is shown below:



5.2 Solution and Technical Architecture

The visual part is implemented using all kinds of swing components, which does not make database calls. The main function of this tier is to display information to the user upon user's request generated by user's inputs such as firing button events. For example, inventory list will display when user click "display" button if he or she wants to know the list of stock remaining in the organization.



Deploying in IBM Cloud
Containerizing the Applications
Application
Tracker
Customer Details
python
Flask
Rest API using Python & Flask
Online Monitor
docker
User
Supplier
Mobile
Cloud Server
Cloud Database
Data Warehouse
Share Holders
Email
IBM Cloud Kubernetes Service
Sales Prediction
Machine Learning

5.3 User Stories

It is important to keep customers happy with what they’re getting. As the number of companies grows, the competition gets more fierce as well, making an inventory management software solution one of the best assets to have in a retail business.

What this does is it lessens the time and effort spent on menial tasks that eat up a substantial amount of time. In growing businesses, time is part of the investment. Once too much time has been taken by a company to ship items or take in orders, customers quickly lose interest. When customers lose interest, they transfer to people who can give them high-quality items in the shortest time possible.

Inventory management software is one of the best methods to use in ensuring customer satisfaction and loyalty to your business. It helps by implementing core features and functionalities such as basic inventory control, barcoding and scanning, demand forecasting, accounting and lot tracking. These features are designed to keep track of your inventory and avoid running out of stock.

6. Project Planning and Scheduling

6.1 Sprint planning & Estimation

Reordering of Inventory:
Inventory reaches a specific threshold; our Inventory Management System can be programmed to tell managers to reorder that product. This helps companies avoid running out of products or tying up too much capital in inventory. This is a very good feature and adds extra advantages to our system. Due to less involvement of human chances of error has been reduced exponentially.

Asset Tracking:

Inventory is a current asset for any company so tracking the asset is mandatory. When an item is in a warehouse, it can be tracked via its barcode or some other ways like serial number, lot number, or revision number. This will help the user and company to track his net worth very clearly. This makes the calculation of net profit and loss more quickly than previous.

Email notification:

Whenever customer books any order in our company an email alert has been sent to him/her as confirmation and tracking id and email notification of their order status send to them periodically.

Service Management:

Companies that are basically service-oriented rather than product-oriented can use an inventory management system to track the cost of the materials they use to deliver facilities, such as cleaning supplies. This way, they can attribute prices to their services that reflect the total cost of executing them.

Barcode:

Due to the use of the barcode process of tracking the product becomes easier. Barcodes are often the means whereby data on products and orders are entered into the inventory management system. A barcode reader is used to read barcodes and look up info on the products they represent. Radio-frequency identification (RFID) tags and wireless methods of product identification are also growing in fame.

6.2 Sprint Delivery Schedule

1. Literature Survey & Information Gathered on28 September

Literature survey on selected project and gathered information by referring the project's related technical papers, research publications, etc.

2. Prepared Empathy Map on 24 September

Prepared empathy map canvas to capture the user's pains & gains and prepare the list of problem statements.

3. Ideation on 25 September

To list by the organizing brainstorm sessions and prioritize the top three ideas based on the feasibility and importance.

4. Proposed Solution on 23 September

To prepare the proposed solution documents, which includes the novelty, feasibility of ideas, business model, social impact, scalability of the solution, etc.

5. Problem Solution Fit on 30 September

Prepared the problem solution fit document.

7. Feature

An easy-to-use interface that doesn’t require advanced training, support or documentation. Automation for eliminating manual processes of business functions related to inventory management. A reliable, secure database that provides accurate, real-time data. Performance that enables fast, actionable inventory monitoring and control. The ability for administrators to easily add software modules with minimal configuration so that the system is scalable. Software integrations and automated features that minimize manual inventory updates or inputs. Testing

The purpose of software testing is to access or evaluate the capabilities or attributes of a software program’s ability to adequately meet the applicable standards and application need. Testing does not ensure quality and the purpose of testing is not to find bugs. Testing can be verification and validation or reliability estimation. The primary objective if testing includes:

To identifying defects in the application.

The most important role of testing is simply to provide information.

To check the proper working of the application while inserting updating and deleting the entry of the products.

7.1 User- Acceptance Testing

This type of testing is the testing of individual software components. It is typically done by the customers and not by the testers. It requires details information and knowledge about the internal program design and code to perform this.

During this testing, we carried out various testing task such as the reflection of the unit data on database and its interface. Various types of bugs associated with the component were identified and fixed. We use various functional keys to test our software.

In our software unit testing is concerned with the stock units, opening stock units and product units validation as well as the validation of product units.

8. Results

Since this is our first project it has some limitation. Due to less knowledge in particular fields and limited time we were not able to fulfill all our expectations that we expected we could do while the project got started. We hope this limitations are considerable. Some of the project limitations are:

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.

Doing something for long time periods always gives good lesson. Some of the things that our team learnt are listed as below:

1. Basically we learnt to work in team.

2. Learnt about the IMS process.

3. Learnt about .NET technology, its components and ways to implement them

4. Learnt to work in pressure and to be patient.

5.Learnt to manage the database under Microsoft SQL server 2008**.**

9. Advantages and Disadvantages

It is important to keep customers happy with what they’re getting. As the number of companies grows, the competition gets more fierce as well, making an inventory management software solution one of the best assets to have in a retail business.

What this does is it lessens the time and effort spent on menial tasks that eat up a substantial amount of time. In growing businesses, time is part of the investment. Once too much time has been taken by a company to ship items or take in orders, customers quickly lose interest. When customers lose interest, they transfer to people who can give them high-quality items in the shortest time possible.

Inventory management software is one of the best methods to use in ensuring customer satisfaction and loyalty to your business. It helps by implementing core features and functionalities such as basic inventory control, barcoding and scanning, demand forecasting, accounting and lot tracking. These features are designed to keep track of your inventory and avoid running out of stock.

Other benefits you can gain from inventory management software include the ability to process and follow work orders directly in the system, create product variations, identify pieces of inventory through serial numbers, bundle individual items to sell as a package, create special orders to meet individual customer needs, as well as track inventory levels across multiple locations.

Inventory management software solutions have features that seek to improve the overall productivity of your business. To achieve maximum customer satisfaction, retail stores and small organizations must apply the most up-to-date technology to assist in daily tasks while also optimizing for the important ones. Even though these tasks might seem like a chore, they help to keep the business running and, more importantly, growing.

10. 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.

11. 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:

1. Interactive user interface design.Manage Stock Godown wise. 3.Use of Oracle as its database.
4.Online payment system can be added.

5.Making the system flexible in any type.

6.Sales and purchase return system will be added in order to make return of

products.

7. Lost and breakage

12. Source code

13. App.py

```python
from flask import Flask, render_template, url_for, request, redirect, session, make_response
import sqlite3 as sql
from functools import wraps
import re
import ibm_db
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-4cd4-bef4-10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32304;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=xqr92747;PWD=ut03NrMp4w3EaZPk", '', '')

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


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


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


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


@app.route('/user/<id>')
@login_required
def user_info(id):
    with sql.connect('inventorymanagement.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM users WHERE email="{id}"')
        user = cur.fetchall()
    return render_template("user_info.html", user=user[0])


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

    if request.method == 'POST':
        un = request.form['username']
        pd = request.form['password_1']
        print(un, pd)
        sql = "SELECT * FROM users WHERE email =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, un)
        ibm_db.bind_param(stmt, 2, pd)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['loggedin'] = True
            session['id'] = account['EMAIL']
            userid = account['EMAIL']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

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


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

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

        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            mg = 'Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms = 'name must contain only character and number'
        else:
            insert_sql = 'INSERT INTO users (USERNAME,EMAIL,PASSWORD) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, username)
            # ibm_db.bind_param(pstmt, 2, "firstname")
            # ibm_db.bind_param(pstmt, 3, "lastname")
            # ibm_db.bind_param(pstmt,4,"123456789")
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.bind_param(pstmt, 3, pw)
            print(pstmt)
            ibm_db.execute(pstmt)
            mg = 'You have successfully registered click login!'
            message = Mail(
                from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
                to_emails=email,
                subject='New SignUp',
                html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for choosing us.</p>')

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

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

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


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

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


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

        except Exception as e:
            msg = e

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


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

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

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

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

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

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

@app.route('/update-password', methods=['POST', 'GET'])
@login_required
def updatePassword():
    if request.method == "POST":
        try:
            email = session['id']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM  users WHERE EMAIL=? AND PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE EMAIL=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
        finally:
            # print(msg)
            return render_template('result.html')

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

```

@app.route('/createOrder', methods=['POST'])
@login_required
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']
            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['PRICE_PER_QUANTITY'])
                query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?,?,?,?,?)'
                pstmt = ibm_db.prepare(conn, query)
                ibm_db.bind_param(pstmt, 1, stock_id)
                ibm_db.bind_param(pstmt, 2, quantity)
                ibm_db.bind_param(pstmt, 3, date)
                ibm_db.bind_param(pstmt, 4, delivery_date)
                ibm_db.bind_param(pstmt, 3, price)
                ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

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


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

```python
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

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


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

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


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

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

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

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

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

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

        except Exception as e:
            msg = e

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

```python

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

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

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

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

if __name__ == '__main__':
    port = int(os.environ.get("PORT",5000))
    app.run(port=port,host='0.0.0.0')
```

```

# ALTER TABLE stocks ALTER COLUMN ID SET GENERATED BY DEFAULT AS IDENTITY

```