PROJECT REPORT

INVENTORY MANAGEMENT SYSTEM

TEAM ID: PNT2022TMID18551

TEAM MEMBERS	
NAME	**ROLLNUMBER**
BALASUBRAMANIAN S	9517201906008
NAVIN G	9517201906031
SRIRAM S	9517201906048
ANBUMANI P	9517201906005

INDEX

INTRODUCTION

Inventory is the supply of raw materials, partially finished goods called work-in-progress and finished goods, an organization maintains to meet its operational needs. It represents a sizeable investment and a potential source of waste that needs to be carefully controlled. Inventory is defined as a stock of goods that is maintained by a business in anticipation of some future demand. The quantity to which inventory must fall to signal that an order must be placed to replenish an item.

Using an extension of a standard inventory-dependent demand model provides a convenient characterization of products that require early replenishment. The optimal cycle time is largely governed by the conventional trade-off between ordering and holding costs, whereas the reorder point relates to a promotions-oriented cost-benefit perspective. The optimal policy yields significantly higher profits than cost-based inventory policies, underscoring the importance of profit-driven inventory management. To work towards perfect order metrics, there must be aggressive inventory management, restructuring supply chain operations, and updating standards to the perfect standard. When updating the metrics, this would include the cases shipped vs. the orders on-time delivery, data synchronization, damages and unusable products, days in supply, the ordering time cycle, and shelf level of service.

Project Overview

The Inventory Management System is an application designed to allow the supermarket staff to create, maintain and view the contents and value of its inventory of items in a categorized way. It also aims to analyze the position of the supermarket in the market and help it know what items to order in what quantity by producing graphs depicting the sale of different items on the different basis such as monthly, yearly, brand type etc.

The Inventory System is to facilitate our customers tracking their products as and when they are transported from the vendor to the warehouse and from the warehouse to the retail location to the customers. It is necessary to keep our resources safe and protected. In order to implement security in the application it would be done by implementing encryption, keeping a secure session base password, implementing two-level authentications, observing system logs and security faults, analyzing network flow using Wireshark, implementing Wireshark, preventing the application validation from unnecessary inputs, session management, session hijacking, hacking, cross-site scripting and implementing code to prevent from SQL injection and many more.

Purpose

The Inventory Management System is a real-time inventory database capable of connecting multiple stores. This can be used to track the inventory of a single store or to manage the delivery of stock between several branches of a larger franchise. However, the system merely records sales and restocking data and provides warning of low stock at any location through email at a specified interval.

The goal is to reduce the stress of tracking rather than to holder all store maintenance. Further features may consist of the ability to create reports of sales, but again the explanation is left to the management. In addition, since theft does occasionally occur, the system provides solutions for confirming the store inventory and for correcting stock quantities.

The inventory management system is used for various purposes, including:

- Maintaining and recording the information between too much and too little inventory in the company.
- Keep track of inventories as it is transported between different locations.
- Recording product information in a warehouse or other location.
- Having a record of Picking, packing, and selling products from a warehouse.
- Reduction of product obsolescence and decay.
- Avoiding out-of-stock situations.

LITERATURE SURVEY

[1] Hind Benfenatki, Catarina Ferreira da Silva, Aïcha-Nabila Benharkat, "Cloud Application Development Methodology" IEEE/WIC/ACM International Conference on Web Intelligence (WI 2014)
This paper describes MADONA methodology, and focuses on the requirements expression phase, by describing RIVAL -a Requirement Vocabulary- based on Linked USDL principles. MADONA allows business stake holders to perform the automatic development of business applications; and combines cloud services discovery and composition with service development using cloud platforms, when the discovery process does not return a service meeting the business stakeholder's requirements. The description of developed services is stored, and the latter are used in the future workflows. MADONA is implemented as "Services Orchestration as a Service." It uses the "Juju" [11] cloud orchestration tool to deploy cloud services in several IaaS. A cloud orchestration tool is available without the underlying physical resources needed for the deployment of services. It allows us to deploy and compose supplied services abstracting from the technical details, i.e. (i) the management of the dependencies between services, (ii) the deployment of selected services, (iii) the scalability of the deployed services. RIVAL describes functional and non-functional requirements for business application development. Functional requirements describe service features. Non-functional requirements describe user preferences and QoS parameters. The rest of this paper is organized as follows. Section 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) 978-1-4799-4143-8/14 $31.00 © 2014 IEEE DOI 10.1109/WI-IAT.2014.1113 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)978-1-4799-4143-8/14 $31.00 © 2014 IEEE DOI 10.1109/WI-IAT.2014.11 132 illustrates how the marketplace's services are described. Section 3 presents the proposed MADONA methodology. Section 4 introduces MADONA's architecture. We describe the implementation and evaluate our work in section 5. Section 6 describes the work related to existing cloud software development approaches. Section 7 draws final conclusions and describes our future work.
[2] Stanley Ewenike, Elhadj Benkhelifa and Claude Chi Belushi, "Cloud Based Collaborative Software Development"
Cloud computing is a technology trend that is changing the IT landscape and changing collaboration [3]. One of its most notable advantage lies in its adaptability to varying contexts of use, its extensibility, as well as, the numerous possibilities and opportunities it presents for all stakeholders to collaborate [37]. However, not unlike most emerging paradigms, mixed feelings trail adoption of the Cloud [4], [5], [38]. For collaborative software development, the benefits include, but are not limited to, cost savings, scalability, agility for business and

development peak period needs, motivation for innovation and increased R&D [29]. On the other hand, there are fears about: security issues; vendor lock-in and interoperability issues, portability issues; automation, performance issues; availability issues; handling uncertainty about: heterogeneity and content type, location of client, bandwidth unpredictability, dynamic workload variations, workflow schedules, architecture and resource optimization; availability and integrity of relevant information within participating teams and systems; context awareness and reproducibility within contexts; amongst others [27], [37], [39]. Some of these challenges and issues listed here are partly inherited since Cloud Computing itself, is a paradigm that leverages a couple of other technologies [40]

Problem Statement Definition :

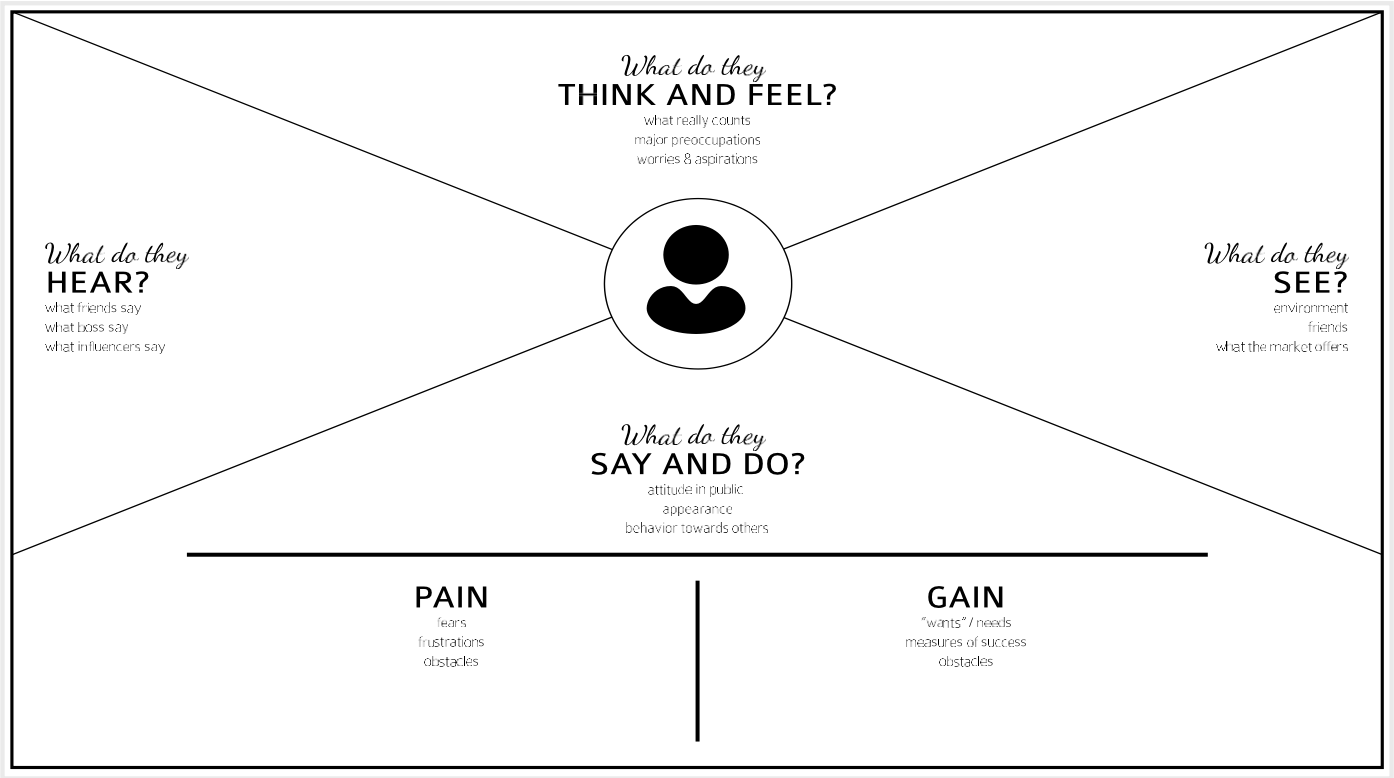
Retail inventory management is ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply. To create effective retail inventory management that results in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

IDEATION AND PROPOSED SOLUTION:

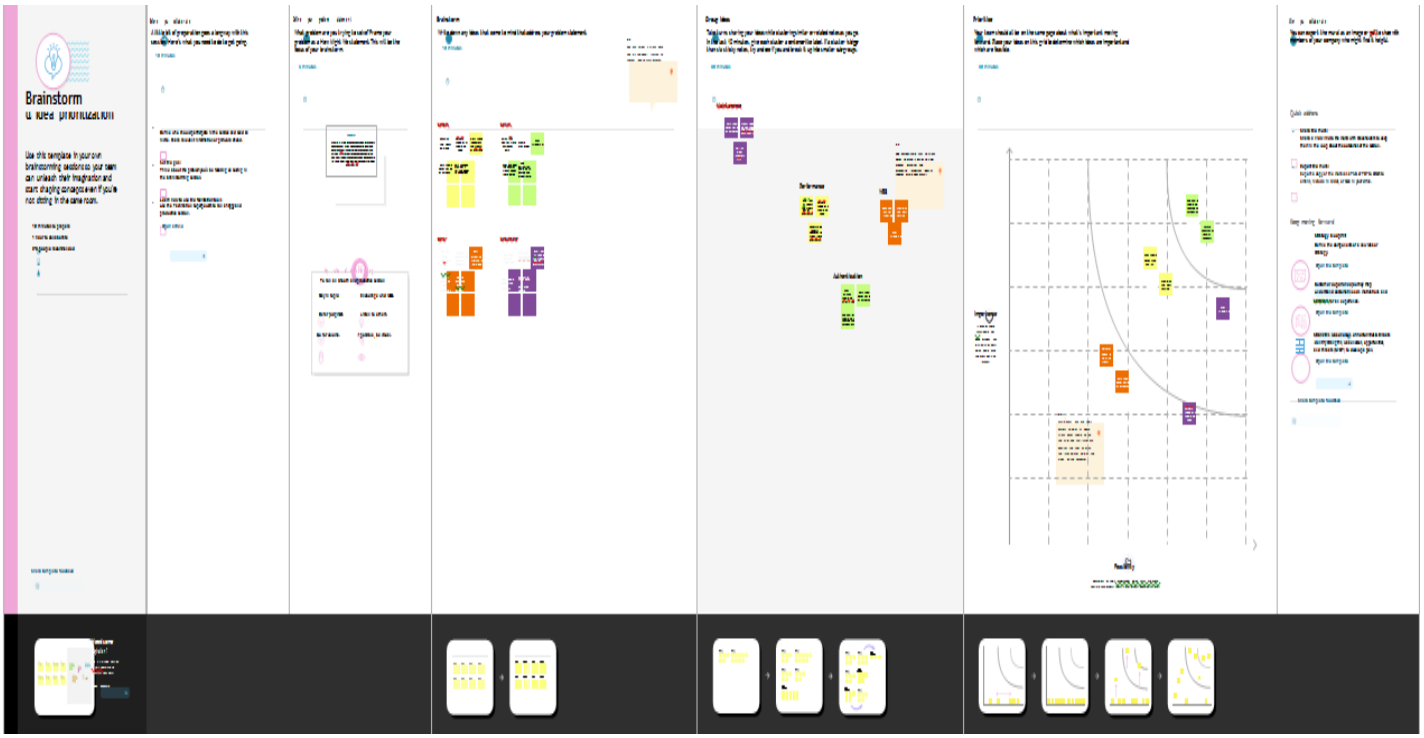
S.No.	Parameter	Description
V	Problem Statement (Problem to besolved)	The retailers generally facing issues in recording the stocks and its thresholdlimit available. The retailers doesn’t know which product is getting expired and when itis being expired. The retailers couldn’t track the availability of all the stocks up-to date. The customers are not satisfied with the retailers store since it doesn’t haveenough supplements and the deliveries were not made on time.

V	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 triggeredto indicate both the expired product and soon going to expire products. • The product availability is tracked dailyand an alert system in again kept on toindicate those products which falls below the threshold limit. • All the customers can register their accounts after which they will be givena login credentials which they can use whenever they feel like buying the stocks. • The application allows the customers toknow all the present time available stocks and also when the new stock willbe available on the store for them to buy. • Tracking the order have become easywith this application for both the retailers and the customers.

Empathy Map Canvas



Ideation & Brainstorming



Proposed Solution fit

Project Title: Inventory Managment System for Retailers

Project Design Phase-I - Solution Fit Template

Team ID: PNT2022TMID18551

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) — CS

Customer segmentation is a powerful tool for businesses to closely align their strategy and tactics with their customers and better target them. Every customer is different, and every customer journey is different, so one approach often won't work for all. This is where customer segmentation becomes a valuable process.

The marketing "persona" is a personification of a customer segment, and it is not uncommon for businesses to create several personas to match their different customer segments.

6. CUSTOMER CONSTRAINTS — CC

Common types of resource constraints include **limits on raw materials, machine capacity, workforce capacity, inventory investment, storage space, or the total number of orders placed**. Customer-owned inventory management is the practice of a supply chain management company controlling warehousing, pick, pack, ship, and delivery of inventory that is owned by the customer whether sourced through non-traditional suppliers or traditionally distributed.

5. AVAILABLE SOLUTIONS — AS

Inconsistent Tracking:
Using manual inventory tracking procedures across different software and spreadsheets is time-consuming, redundant, and vulnerable to errors.

Warehouse Efficiency:
Inventory management controls at the warehouse is labor-intensive and involves several steps, including receiving and put away, picking, packing, and shipping

Explore AS, differentiate

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

2. JOBS-TO-BE-DONE / PROBLEMS — J&P

1. Goods are delivered to your facility.
2. Inspect, sort and store goods.
3. Monitor inventory levels.
4. Stock orders are placed.
5. Stock orders are approved.
6. Take goods from stock
7. Update inventory levels
8. Low stock levels trigger purchasing/reordering.

9. PROBLEM ROOT CAUSE — RC

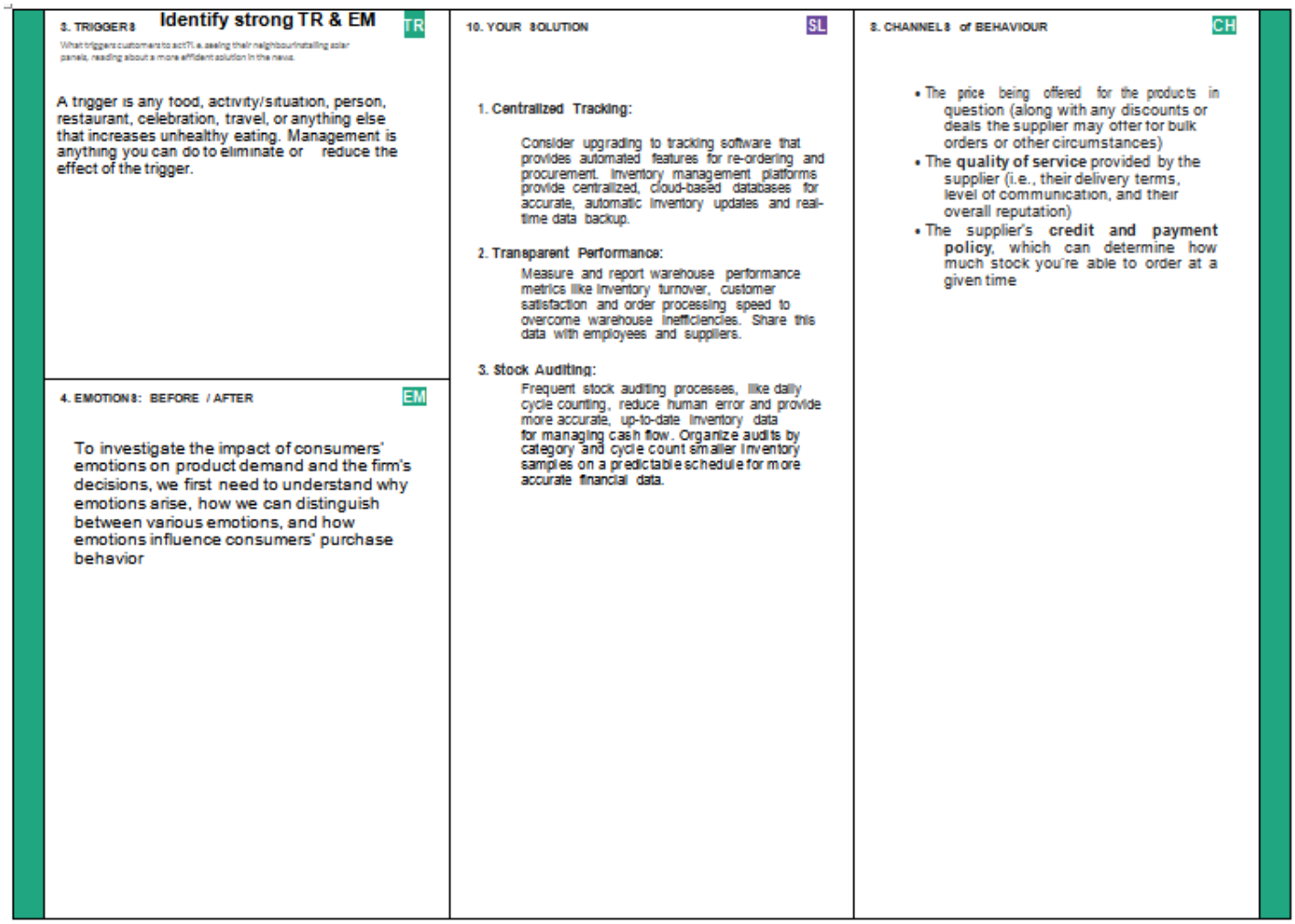
The main reasons identified for the accumulation of inventory are
(1) forecasting error,
(2) bulk purchase,
(3) data entry error,
(4) communication gaps,
(5) quality-related issues,
(6) product category not traceable and
(7) wrong material being procured.

One of the most common challenges to sound inventory management is preventing the overselling of products and running out of inventory. Using historical and seasonal data trends can help you accurately predict customer orders

7.

Staffing: To provide a valuable experience to your customers, you must have a sufficient staff of qualified individuals in place. Market Research: Understanding the ins and outs of your industry, as well as where your competitors stand. Logistics: It's vital that you have a plan for moving your products from the supplier to the customer — and why your company is essential in this regard. Finances: Overall, you'll have to have a clear idea of your operational costs, as well as your intended profit margins

J&P, tap into BE, understand RC

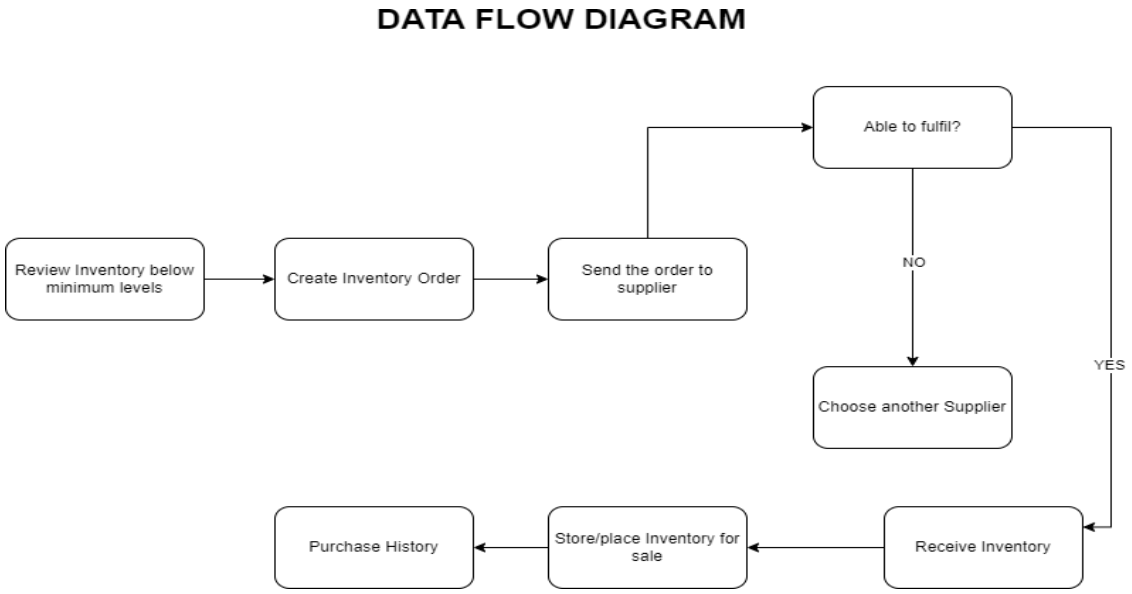


REQUIREMENT ANALYSIS

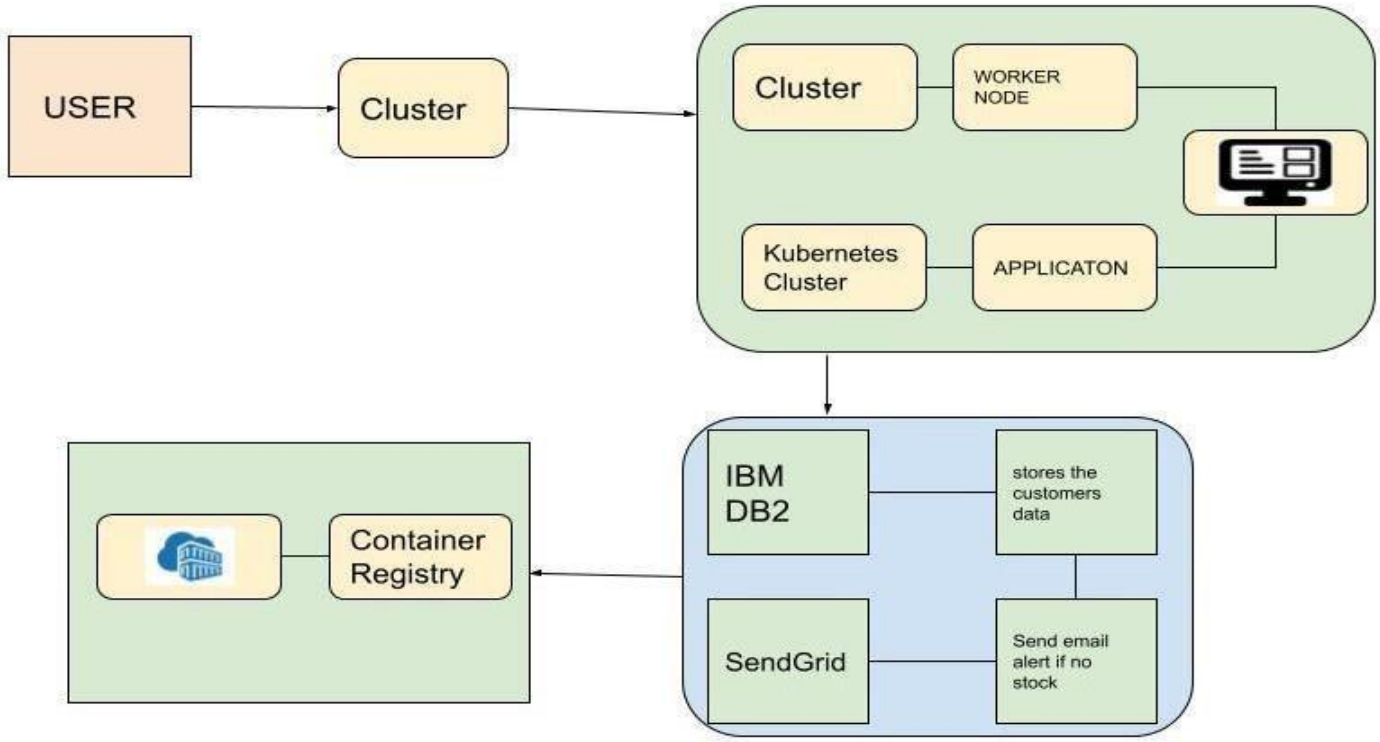
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration through own application Form	Registration through LinkedIN Registration through own application Form Registration through Google Docs.
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login through User name and Password. 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 product	Product name Product category Product I'd Stock Count Vendor details

1. PROJECT DESIGN

DATA FLOW DIAGRAM



Solution and Technical Architecture :



PROJECT PLANNING & SCHEDULING

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Literature survey on selected project and gathering information by referring the project's related technical papers, research publications, etc.	28 SEPTEMBER 2022
Prepare Empathy Map	Prepare empathy map canvasto capture the user's pains & gains and prepare the list of problem statements.	24 SEPTEMBER 2022
Ideation	To list by the organizing brainstorm sessions and prioritize the top three ideas based on the feasibility and importance.	25 SEPTEMBER 2022
Proposed Solution	To prepare the proposed solution documents, which includes the novelty, feasibilityof ideas, business model, social impact, scalability of thesolution, etc.	23 SEPTEMBER 2022
Problem Solution Fit	Preparing the problem solution fit document.	30 SEPTEMBER 2022
Solution Architecture	To prepare the solution architecture document	28 SEPTEMBER 2022
Customer Journey	Prepare the customers journey map help the customers understand the user interaction and experiences with the application from the beginning to the end.	20 OCTOBER 2022
Functional Requirement	Prepare the functional requirement document.	8 OCTOBER 2022
Data Flow Diagrams	Draw the data flow diagrams and submit for the review.	9 OCTOBER 2022

Technology Architecture	Prepare technical architecture diagram.	10 OCTOBER 2022
Prepare Milestone & Activity List	Prepare the milestones and activity of the project.	22 OCTOBER 2022
Project Development – **Delivery of Sprint-1, 2, 3 & 4**	Develop and submit the developed code by testing it and having no errors.	18 NOVEMBER 2022

Product Backlog, Sprint Schedule, and Estimation

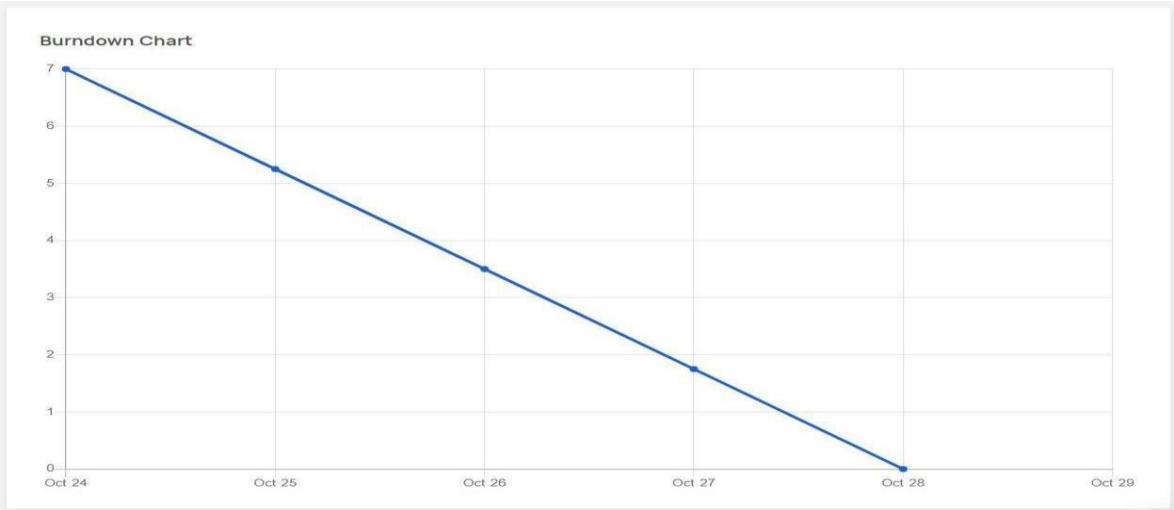
Sprint	**Functional Requirement(Epic)**	**User Story Number**	**User Story / Task**	**Story Points**	**Priority**	**Team Members**
Sprint-1	Registration	USN-1	As a user, I can register forthe application by enteringmy email, password, and confirming my password.	2	High	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI
Sprint-1		USN-2	As a user, I can register forthe application through E- mail	1	Medium	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	2	Medium	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI

Sprint	Functional Requirement(Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	4	High	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI
Sprint-2	Add items	USN-6	As a user, I can add the products I wish to buy tothe carts.	5	Medium	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI
Sprint-3	Stock Update	USN-7	As a user, I available in the dashboardto the stock list.	5	Medium	S.BALASUBRAMANIYAN, G.NAVIN, S.SRIRAM, P.ANBUMANI

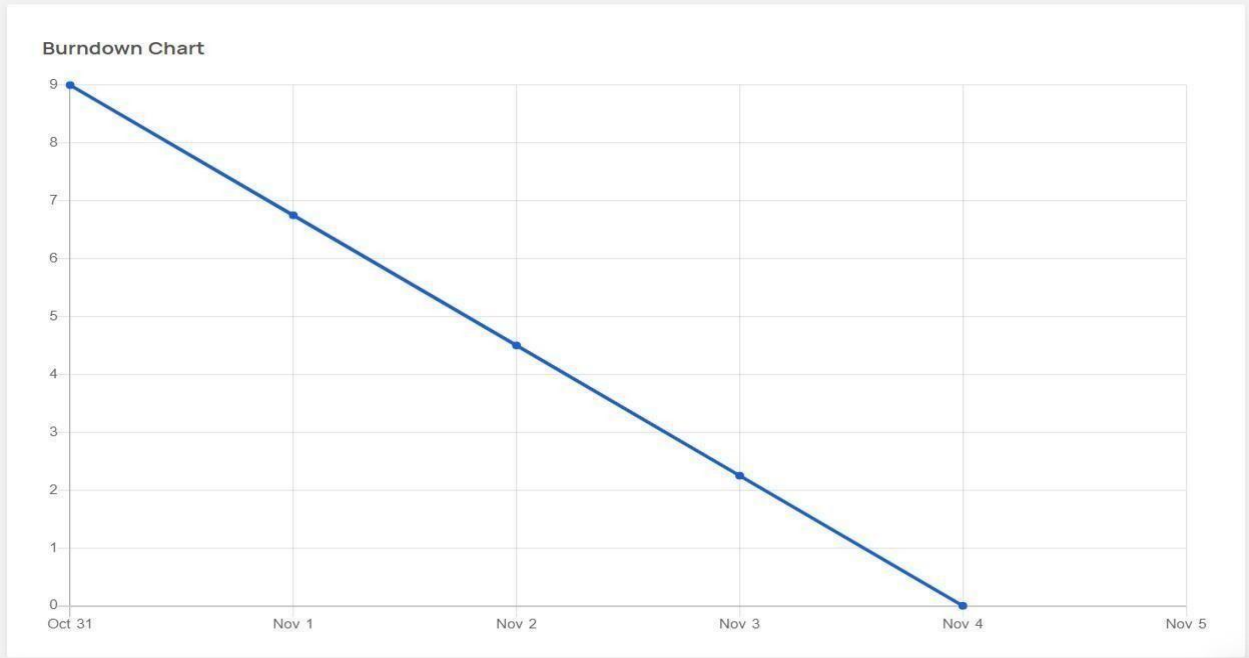
Sprint-4	Reques t to Custom er Care	USN-8	As a user, I can contact the Customer Care E xecutiveand request any services I want from t	5	Low	S.BALASUB RAMANIYA N, G.NAVIN, S.SRIRA M, P.ANBU MANI

Burndown Chart:

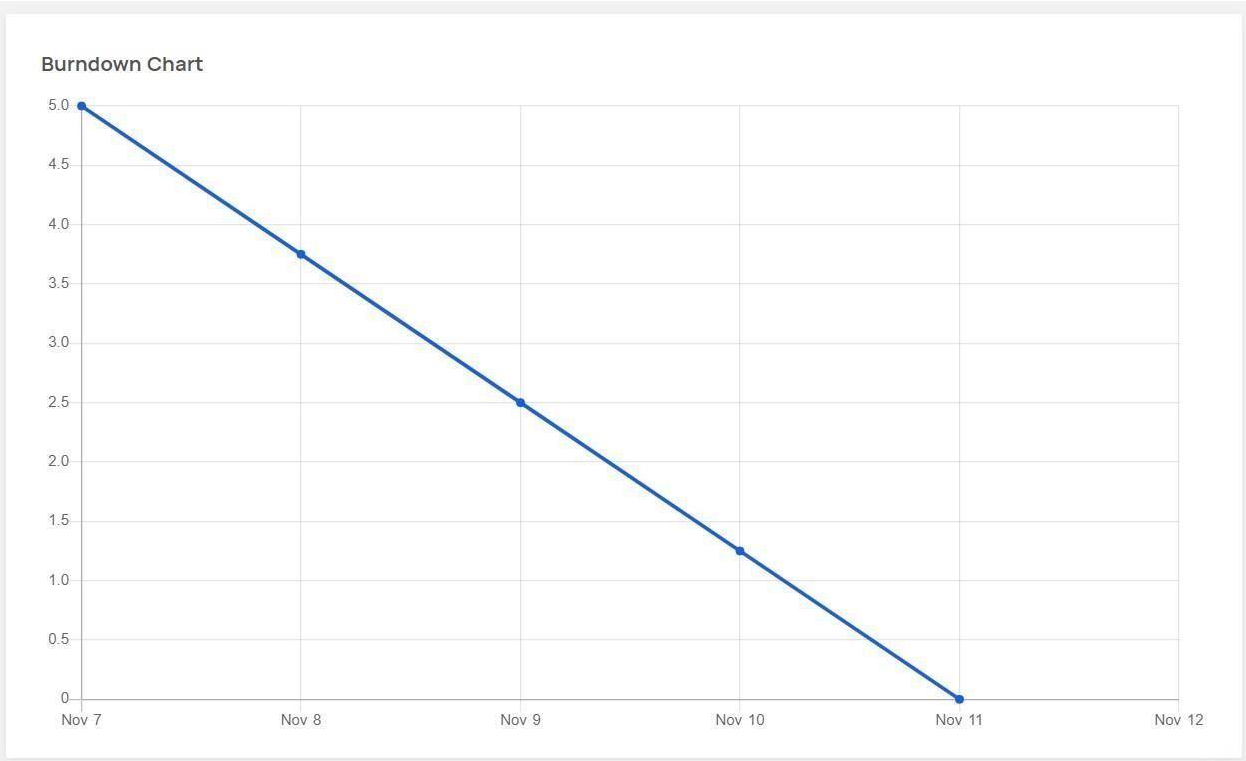
Sprint - 1



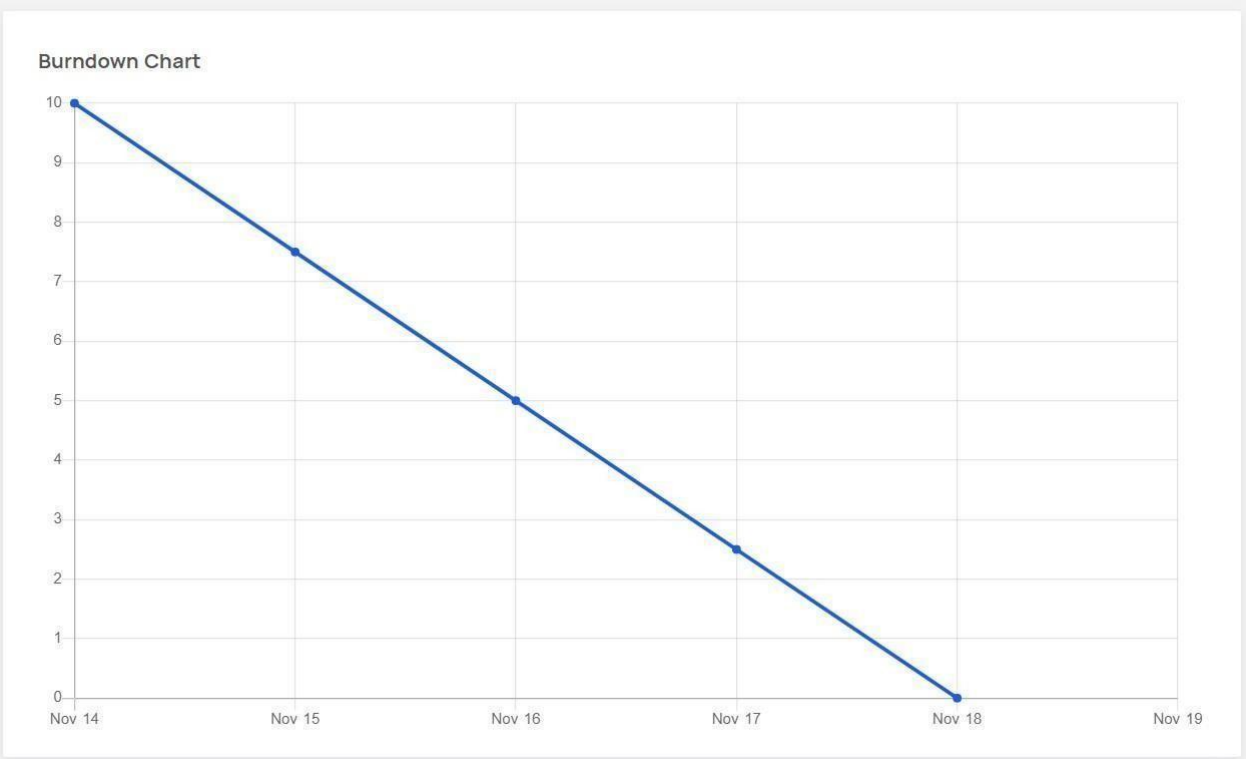
Sprint - 2



Sprint - 3



Sprint-4



CODING:

```
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=815fa4db-
dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;POR
T=30367;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRoot
CA.crt;UID=gkx49901;PWD=kvWCsySl7vApfsy2", ", ")
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,FIRSTNAME,LASTNAME,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, 4, email)
        ibm_db.bind_param(pstmt, 5, 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, 5, price)
            ibm_db.execute(pstmt)
    except Exception as e:
        print(e)

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



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

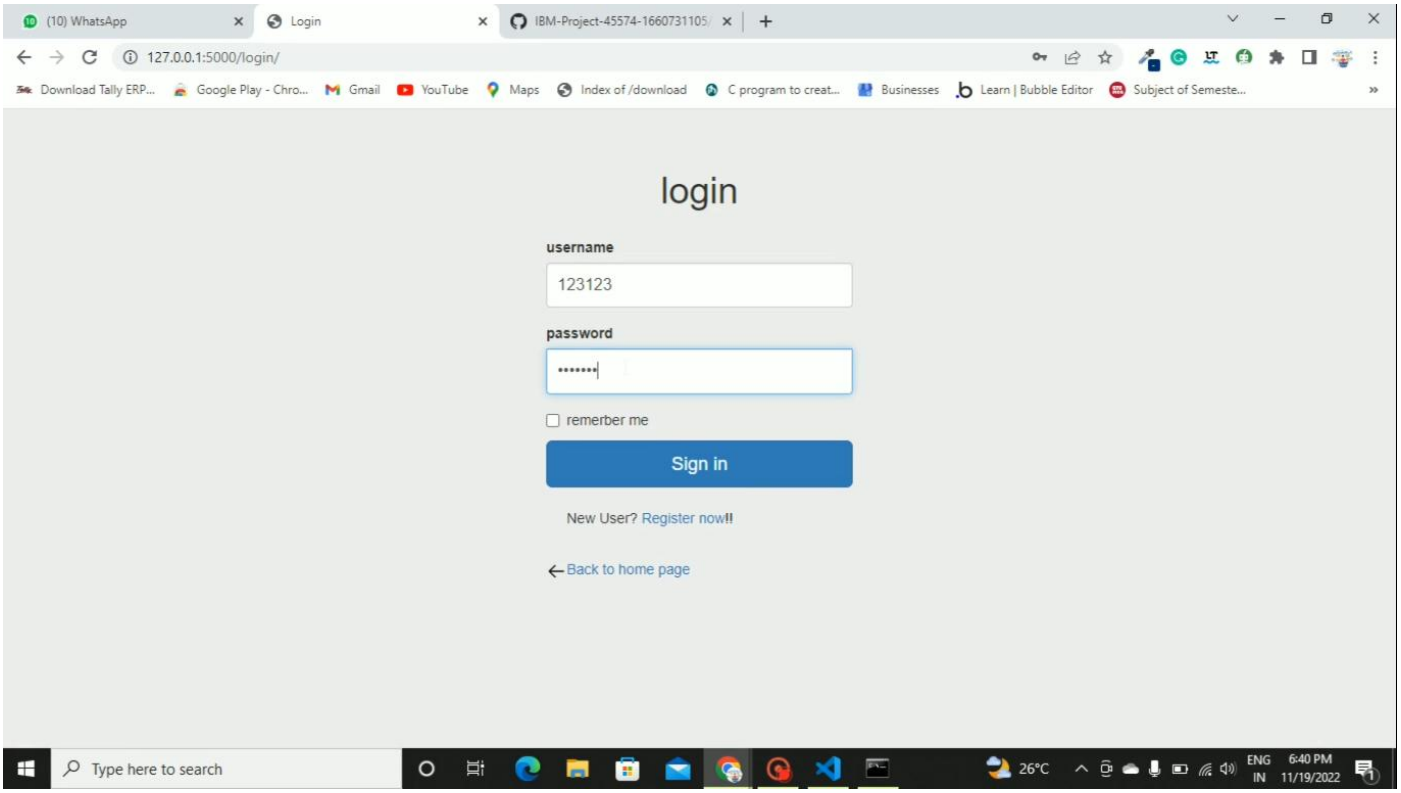
```
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

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


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

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

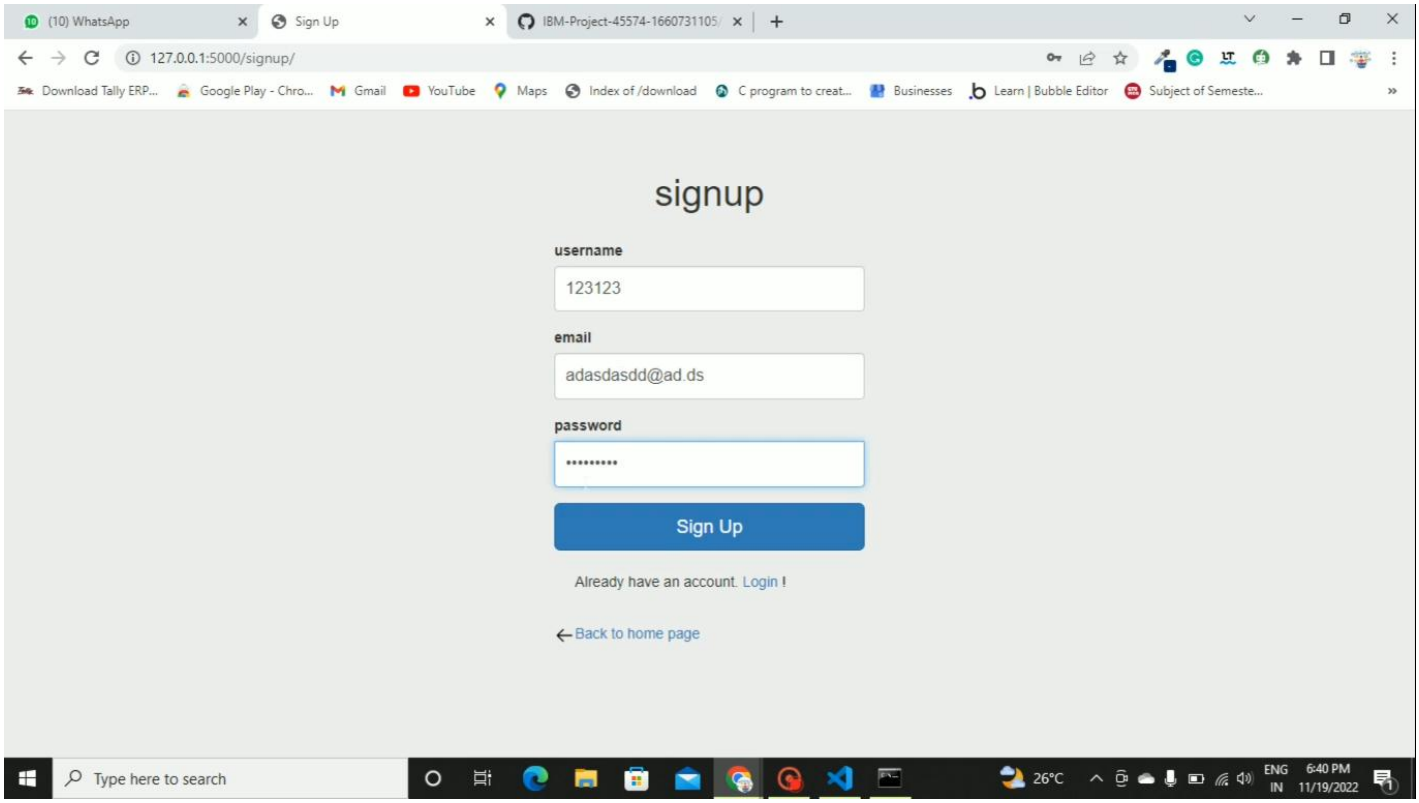
```
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

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

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



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

```
        finally:
            return redirect(url_for('suppliers'))
@app.route('/addsupplier', methods=['POST'])
@login_required
def addSupplier():



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

        except Exception as e:
            msg = e

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



@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
```

```
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

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


@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:
            email = session['id']
            insert_sql = 'SELECT * FROM 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__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

OUTPUT SCREENSHOTS





Inbox (160) - selvamb | bnsa - ProtectedText | IBM | IBM-Project-14752-16 | Recent - Google Drive | IBM Cloud

cloud.ibm.com

Gmail | YouTube | Maps | Online Java Compiler | s.balasubramaniyan... | AnyAPI | SmartInternz | Recursive Digit Sum... | Home Page | Placement Papers 2... | Top 40 Coding Inter...

IBM Cloud | Search resources and products... | Catalog | Manage | Balasubramaniyan S's ...

Dashboard

Edit dashboard | Upgrade account | Create resource +

For you | Select an option

Build
Explore IBM Cloud with this selection of easy starter tutorials and services.

Build a web app with Watson Speech to Text
Deploy a conversational interface compatible with any application, device, or channel.
Getting started | 15 min

Get Started with Watson Studio
Get started with using AI and Cloud Object Storage in 15 minutes.
Popular | 2 hr

Build a Virtual Private Cloud (VPC)
Upgrade to a paid account to create your your own protected space in the IBM Cloud.
Getting started | 7 min

Learn about IAM Roles
Learn about roles in IBM Cloud and how they work to control access.
Recommended | 5 min

Build a
Lift and
worklo
Getting

User access | Manage users
Enter email addresses below to jump directly into the invite user setup

News | View all
Introducing Badges to IBM Cloud Certification

Planned maintenance | View

Type here to search | 25°C | ENG | 11:58 PM 11/19/2022

Inbox (160) - sel | bnsa - Protected | IBM | IBM-Project-483 | IBM-Project-147 | Recent - Google | Service Details -

cloud.ibm.com/services/dashdb-for-transactions/crn%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aus-south%3Aa%2Fc442fecabf3149e2938f98d554c1d...

Gmail | YouTube | Maps | Online Java Compiler | s.balasubramaniyan... | AnyAPI | SmartInternz | Recursive Digit Sum... | Home Page | Placement Papers 2... | Top 40 Coding Inter...

IBM Cloud | Search resources and products... | Catalog | Manage | Balasubramaniyan S's ...

Resource list /

inventory

Active | Add tags

Details | Actions...

- Manage
- Getting started
- Service credentials
- Connections

Getting started

Where can I find my credentials?
Get your username and password by clicking the "Service Credentials" link to the left and selecting "New Credentials".
Don't see this menu on the left? Click on "Manage in IBM Cloud" to open the IBM Cloud dashboard.

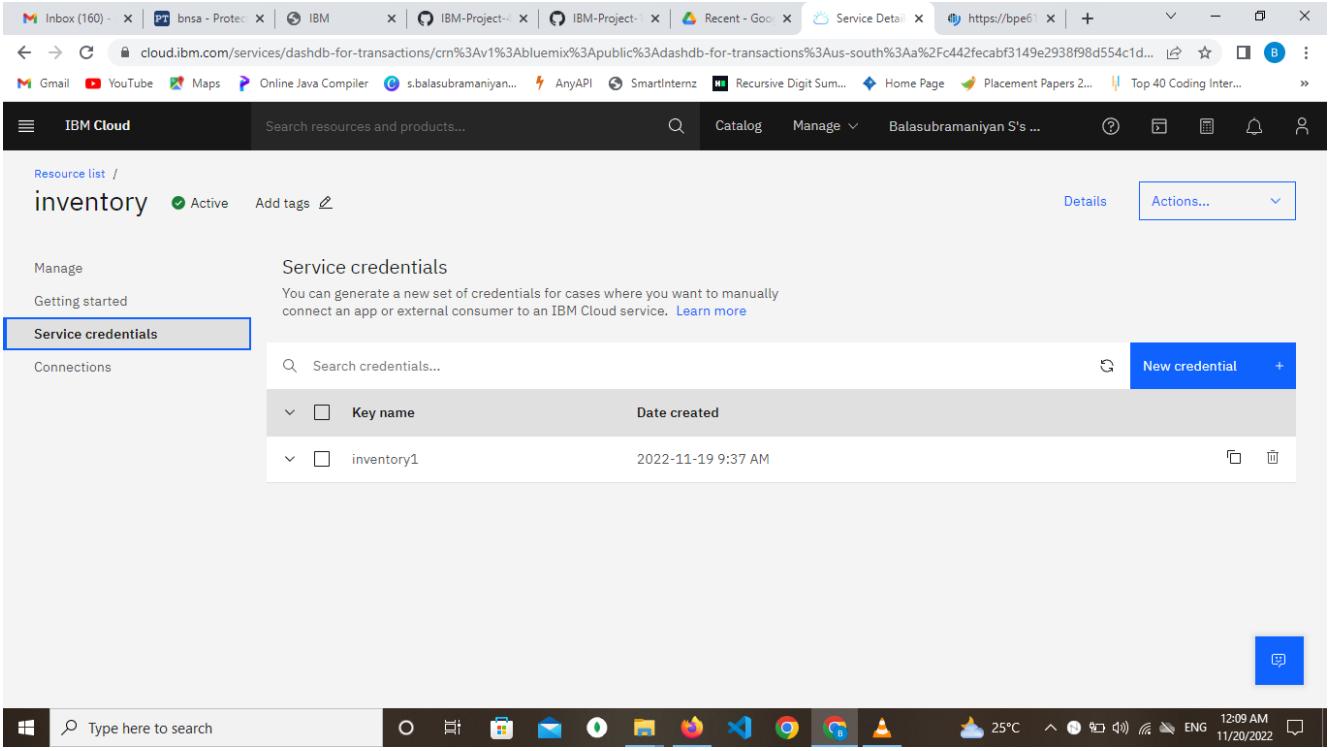
Go to UI | Getting started docs

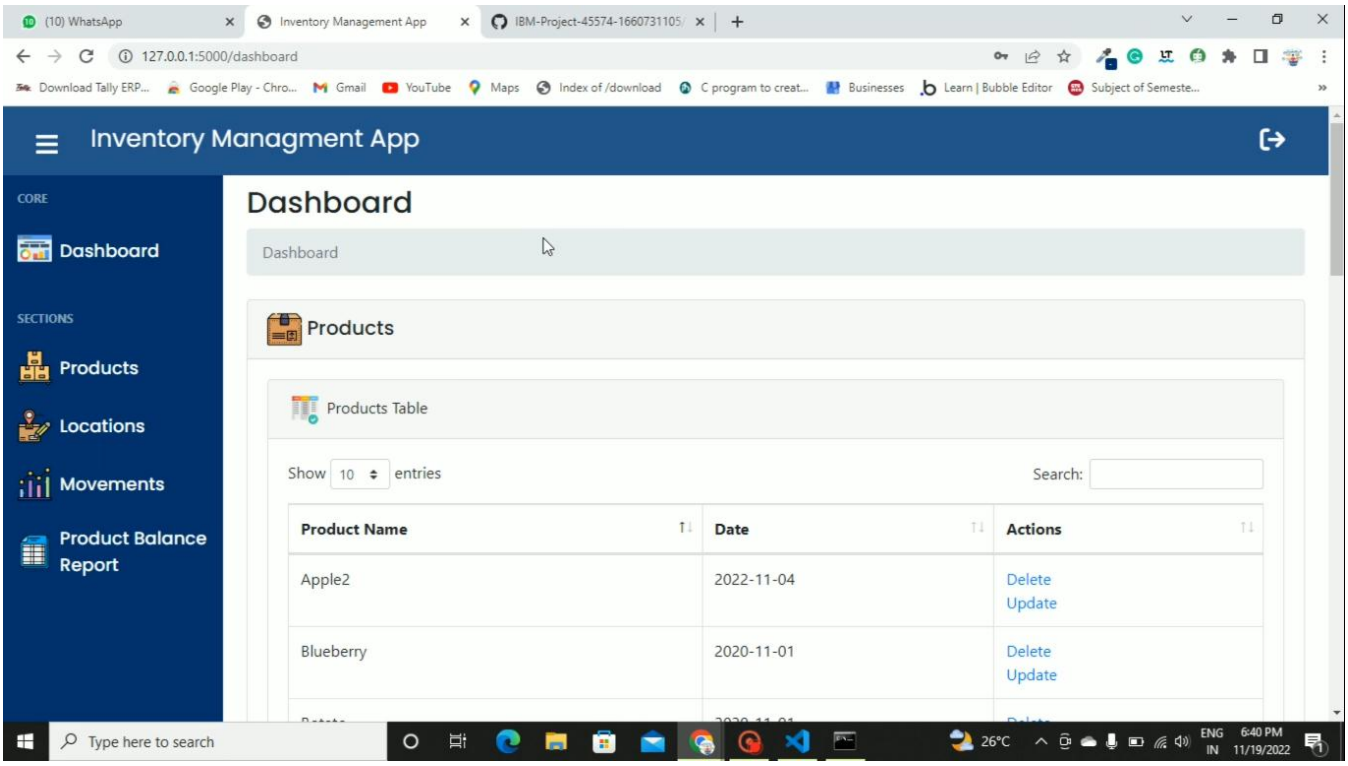
Need help?

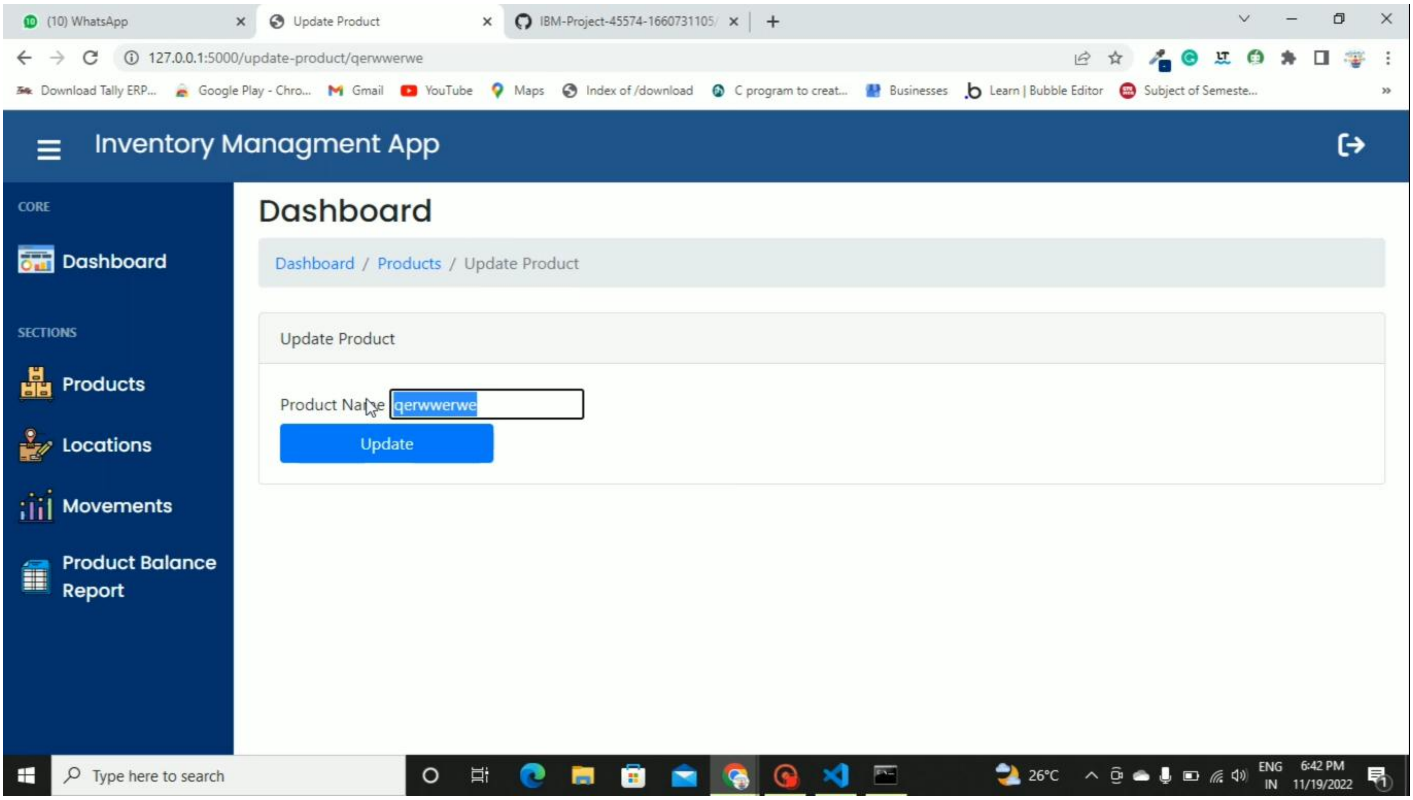
Submit a IBM Cloud Support Case to our team.

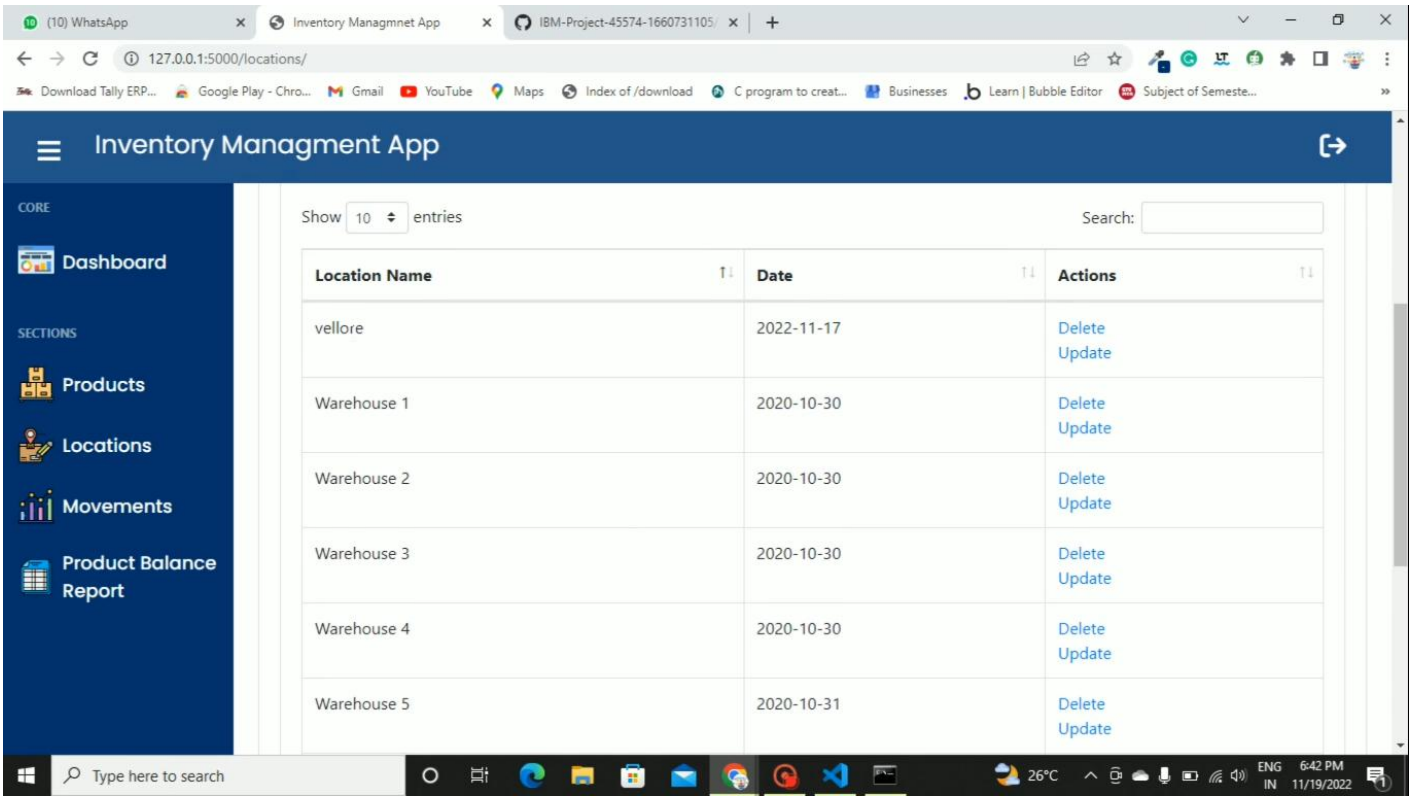
Support case

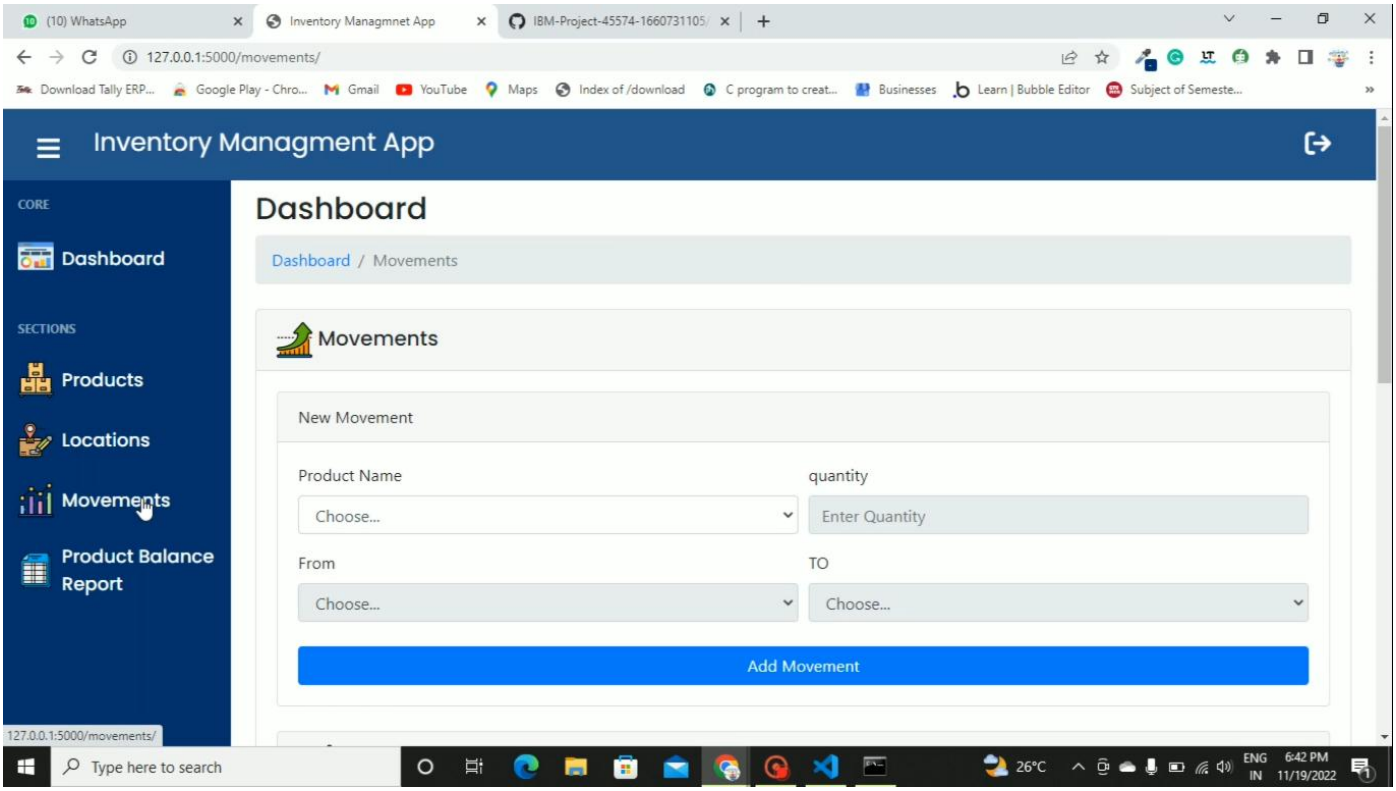
Type here to search | 25°C | ENG | 12:09 AM 11/20/2022

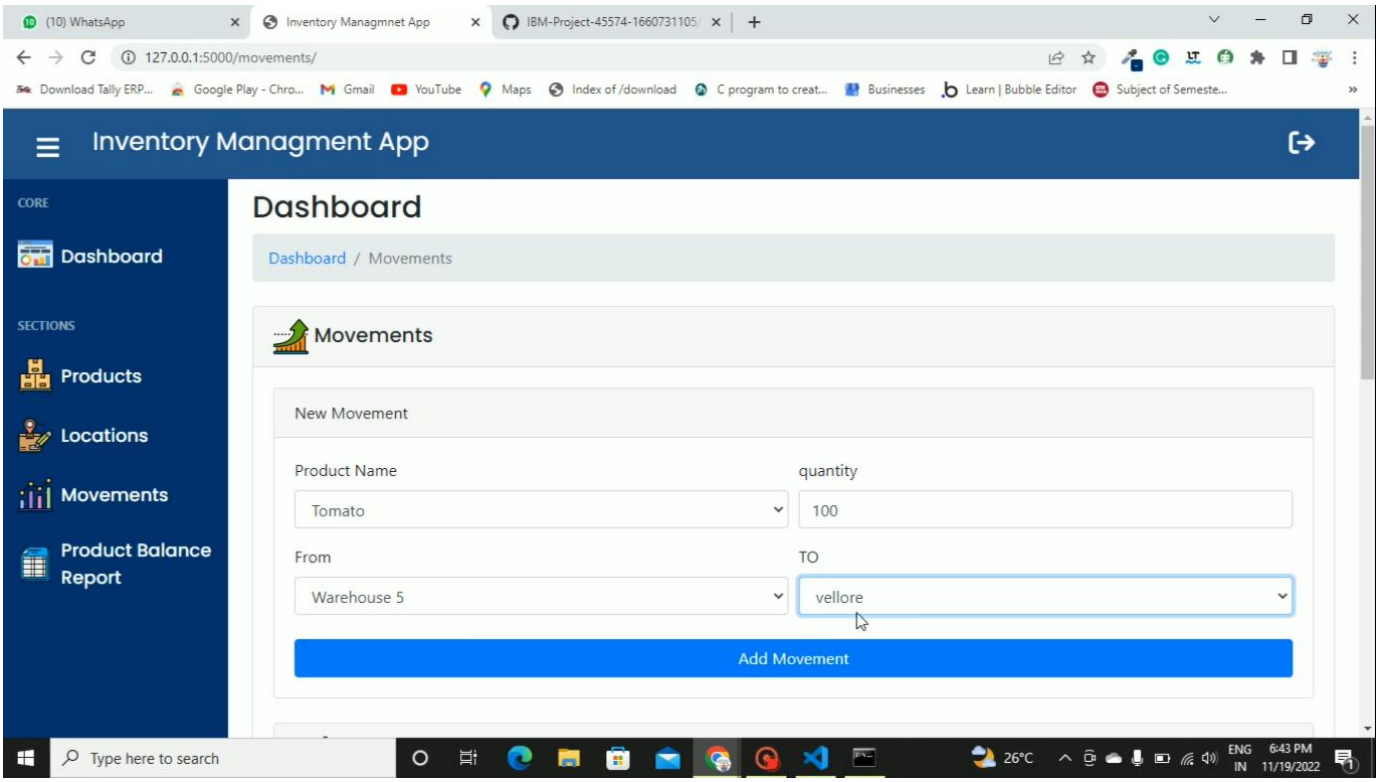

IBM Cloud
Search resources and products...
Catalog
Manage
Balasubramaniyan S's ...
Resource list /
inventory
Active
Add tags
Details
Actions...
Manage
Getting started
Service credentials
Connections
Service credentials
You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service. Learn more
Search credentials...
New credential
Key name
Date created
inventory1
2022-11-19 9:37 AM

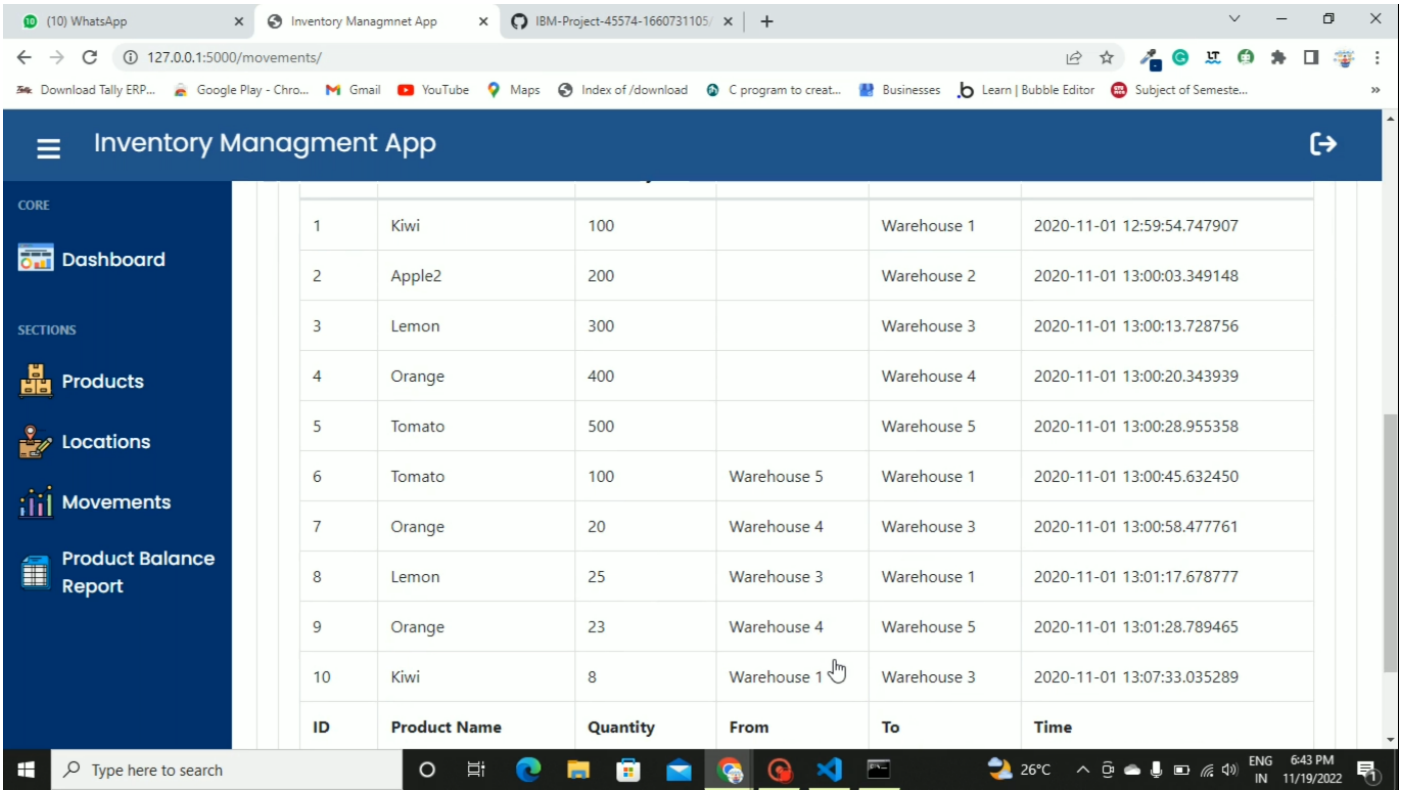

127.0.0.1:5000/dashboard
Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Dashboard
Dashboard
Products
Products Table
Show 10 entries
Search:
Product Name
Date
Actions
Apple2
2022-11-04
Delete
Update
Blueberry
2020-11-01
Delete
Update


(10) WhatsApp
Update Product
IBM-Project-45574-1660731105/
127.0.0.1:5000/update-product/qerwwerwe
Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Dashboard
Dashboard / Products / Update Product
Update Product
Product Name
qerwwerwe
Update
Type here to search
26°C
ENG IN
6:42 PM
11/19/2022


(10) WhatsApp
Inventory Managmnet App
IBM-Project-45574-1660731105/
127.0.0.1:5000/locations/
Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Show 10 entries
Search:
Location Name
Date
Actions
vellore
2022-11-17
Delete
Update
Warehouse 1
2020-10-30
Delete
Update
Warehouse 2
2020-10-30
Delete
Update
Warehouse 3
2020-10-30
Delete
Update
Warehouse 4
2020-10-30
Delete
Update
Warehouse 5
2020-10-31
Delete
Update
Type here to search
26°C
ENG IN
6:42 PM
11/19/2022


(10) WhatsApp
Inventory Managmnet App
IBM-Project-45574-1660731105/
127.0.0.1:5000/movements/
Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Dashboard
Dashboard / Movements
Movements
New Movement
Product Name
quantity
Choose...
Enter Quantity
From
TO
Choose...
Choose...
Add Movement
Type here to search
26°C
ENG IN
6:42 PM 11/19/2022


(10) WhatsApp
Inventory Managmnet App
IBM-Project-45574-1660731105/
127.0.0.1:5000/movements/
Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Dashboard
Dashboard / Movements
Movements
New Movement
Product Name
quantity
Tomato
100
From
TO
Warehouse 5
vellore
Add Movement
Type here to search
26°C
ENG IN
6:43 PM 11/19/2022



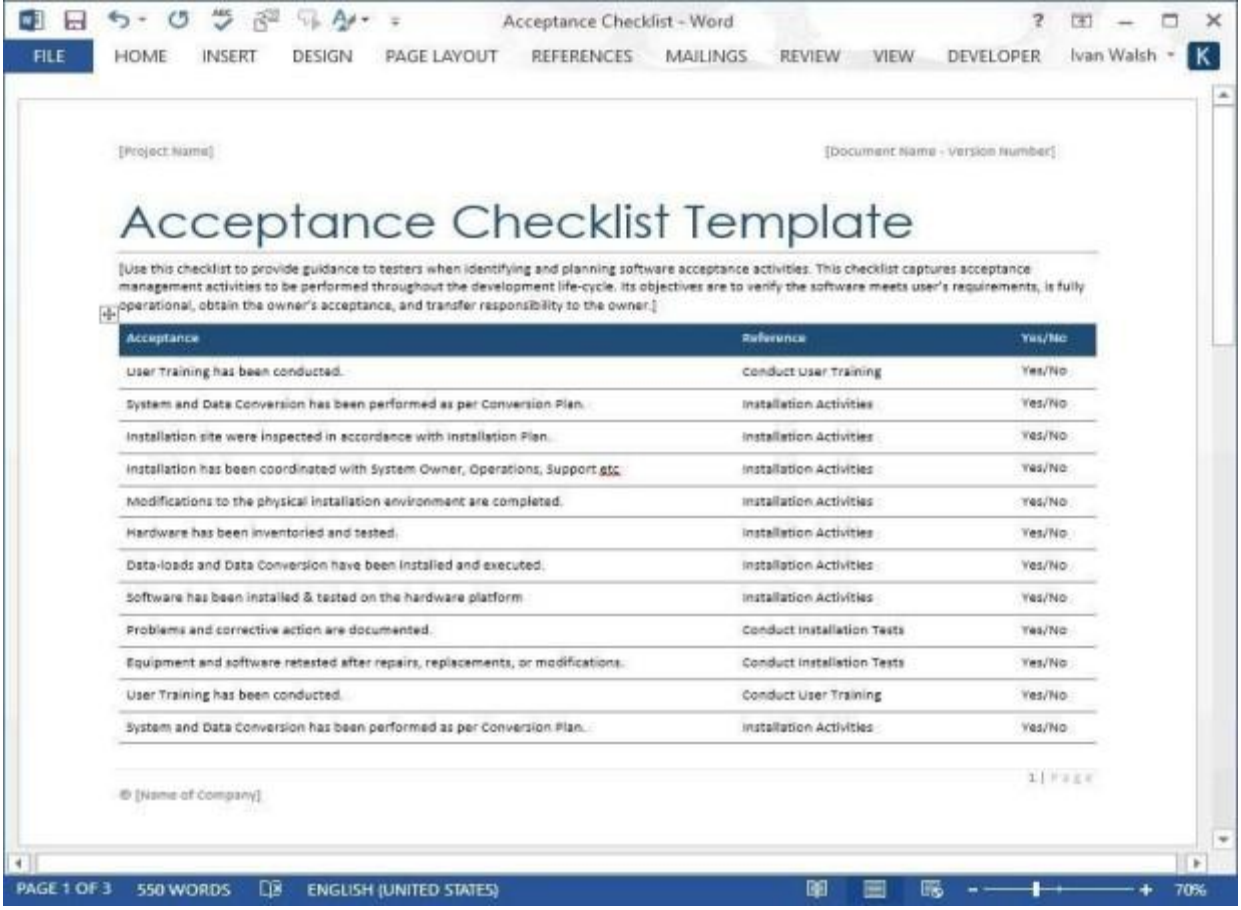
Source code:

```
dsn_hostname = "9938aec0-8105-433e-8bf9-0fbb7e483086.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud" # e.g.: "54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud
" dsn_uid = "krh91100" # e.g. "abc12345" dsn_pwd = "gSPyVtDpdim5wKGL" # e.g. "7dBZ3wWt9XN6$o0J"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "bludb" # e.g. "BLUDB"
dsn_port
= "32459" # e.g. "32733"
dsn_protocol = "TCPIP" # i.e.
"TCPIP" dsn_security = "SSL"#i.e.
"SSL"


dsn = (
"DRIVER={0}
;"
"DATABASE={1};"
"HOSTNAME={2};"
"PORT={3};"
```

```
"PROTOCOL={4};"
"UID={5};"
"PWD={6};"
"SECURITY={7};").format(dsn_driver, dsn_database,
dsn_hostname, dsn_port, dsn_protocol, dsn_uid,
dsn_pwd,dsn_security)
try
conn = ibm_db.connect(dsn, "", "") print ("Connected to database: ",
dsn_database, "as user: ", dsn_uid, "on host: ", dsn_hostname)

except: print ("Unable to connect: ", ibm_db.conn_errormsg()
```

User Acceptance Testing

Step	Procedures	Expected Result	Result
1	Insert admin, username, and password	Save the insert data into database	Success
2	Insert correct username, password for login	Verify the admin	Success
3	Click 'Register,' 'Login' button	Application redirect admin to Login page after register and Main page after login	Success
4	Repeat step 2 and 3 for login using false username, password	Application display error message	Success
5	Update Admin Account	New update data saved into database	Success
6	Log Out Account	Log out redirected to Login page	Success
Precondition		No credentials are currently login	
Post-condition		New and updated Admin name, username, and password saved in	



[Project Name] [Document Name - Version Number]

Acceptance Checklist Template

[Use this checklist to provide guidance to testers when identifying and planning software acceptance activities. This checklist captures acceptance management activities to be performed throughout the development life-cycle. Its objectives are to verify the software meets user's requirements, is fully operational, obtain the owner's acceptance, and transfer responsibility to the owner.]

Acceptance	Reference	Yes/No
User Training has been conducted.	Conduct User Training	Yes/No
System and Data Conversion has been performed as per Conversion Plan.	Installation Activities	Yes/No
Installation site were inspected in accordance with installation Plan.	Installation Activities	Yes/No
Installation has been coordinated with System Owner, Operations, Support etc	Installation Activities	Yes/No
Modifications to the physical installation environment are completed.	Installation Activities	Yes/No
Hardware has been inventoried and tested.	Installation Activities	Yes/No
Data-loads and Data Conversion have been installed and executed.	Installation Activities	Yes/No
Software has been installed & tested on the hardware platform	Installation Activities	Yes/No
Problems and corrective action are documented.	Conduct Installation Tests	Yes/No
Equipment and software retested after repairs, replacements, or modifications.	Conduct Installation Tests	Yes/No
User Training has been conducted.	Conduct User Training	Yes/No
System and Data Conversion has been performed as per Conversion Plan.	Installation Activities	Yes/No

© [Name of Company]

APP.PY

```
rom 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=815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=gkx49901;PWD=kvWCsySl7vApfsy2", '', '')

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,FIRSTNAME,LASTNAME,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, 4, email)
            ibm_db.bind_param(pstmt, 5, 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, 5, price)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

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



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

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



    @app.route('/cancelOrder', methods=['POST'])
    @login_required
    def cancelOrder():
        if request.method == "POST":
            try:
```

```
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

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



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

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

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

@app.route('/updatesupplier', methods=['POST'])
@login_required
def UpdateSupplier():
    if request.method == "POST":
```

```
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

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

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

        except Exception as e:
            msg = e

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

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

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

        finally:
            return redirect(url_for('suppliers'))
@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:
            email = session['id']
            insert_sql = 'SELECT * FROM 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__':
```

```
app.run(debug=True)
```

RESULT

Inventory Performance is a measure of how effectively and efficiently inventory is used and replenished. The goal of inventory performance metrics is to compare actual on-hand dollars versus forecasted cost of goods sold.

- Weeks on Hand. ...
- Inventory Turnover Rate. ...
- Days on Hand. ...
- Stock to Sales Ratio. ...

ADVANTAGE AND DISADVANTAGE

Advantage:

- To maintain the right amount of stocks
- To a more organized warehouse
- It saves time and money
- Improves efficiency and productivity
- A well-structured inventory management system leads to improved customer retention:
- It leads It helps Avoid lawsuits and regulatory fines
- Schedule maintenance
- Reduction in holding costs
- Flexibility

Disadvantage:

- Bureaucracy
- Impersonal touch
- Production problem
- Increased space is need to hold the inventory
- Complexity

Conclusion

In conclusion As you can see the importance of inventory management is very serious, it is one of the most important aspects of any business. The aspect of this part of the business is whether or not you can satisfy the demand of your customers if you aren't sure if you have all the materials availableto make the final product Without having the proper inventory management they would not be able to supply their customers with their ordered ambulance. And this product is what their entire business is based on, so it is of great importanceWhen they are choosing from the different types of programs or automated systems to help with keeping records accurate, needs to keep in mind that the customer is not concerned with which materials are needed to complete the finished product, but the product is operating as promised based on the contract. In addition, the plans for the maintenance of having proper inventory levels need to be in place and also adjusted when the company grows and as the business dictates implements the new suggestions they will be on the right track to having a well established business