INVENTORY MANAGEMENT SYSTEM FOR RETAILERS

Team ID : :PNT2022TMID34293

Team Members :

VARSHA R

SARASWATHY V

LIKITHA T

SOWNDARYA S

1.Introduction:

1.1 Project Overview:

Retail inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply.

In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application.

Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.

1.2 Purpose:

2. Literature Survey :

2.1 Introduction:

Inventory management is a challenging problem in supply chain management. A tool or system to aid the inventory management would be a beneficial tool in thisarea. The term inventory refers to a company's stockpile of material and the components that make up the output. Inventory management refers to managing the quantity, quality, location and transportation of various products utilised in manufacturing by various industrial organisations or in sales by various retailers. Accurately maintaining the quantity (in numbers) of the finished goods in the inventory makes it possible to quickly asses the quantity of products needed for the upcoming sales. It also improves the communication between the entities of the supply chain like retailers, manufacturers, customers, etc.

2.2 Literature Survey:

1. Design of a Computerized Inventory Management System for Supermarkets.

2. The inventory management system for automobile spare parts in a central warehouse

3. Automated Inventory Management Systems and its impact on Supply Chain Risk Management in Manufacturing.

4.Design of smart inventory management system for construction sector based on IoT and cloud computing.

5. A study on Inventory management in Tamil Nādu State Transport

Corporation Limited, Kumbakonam.

6.Impact of Inventory Management on the Profitability of SMES in Tanzania.

7. An assessment of the Inventory Management Practices of Small and Medium Enterprises (SMEs) in the Northern Region of Ghana.

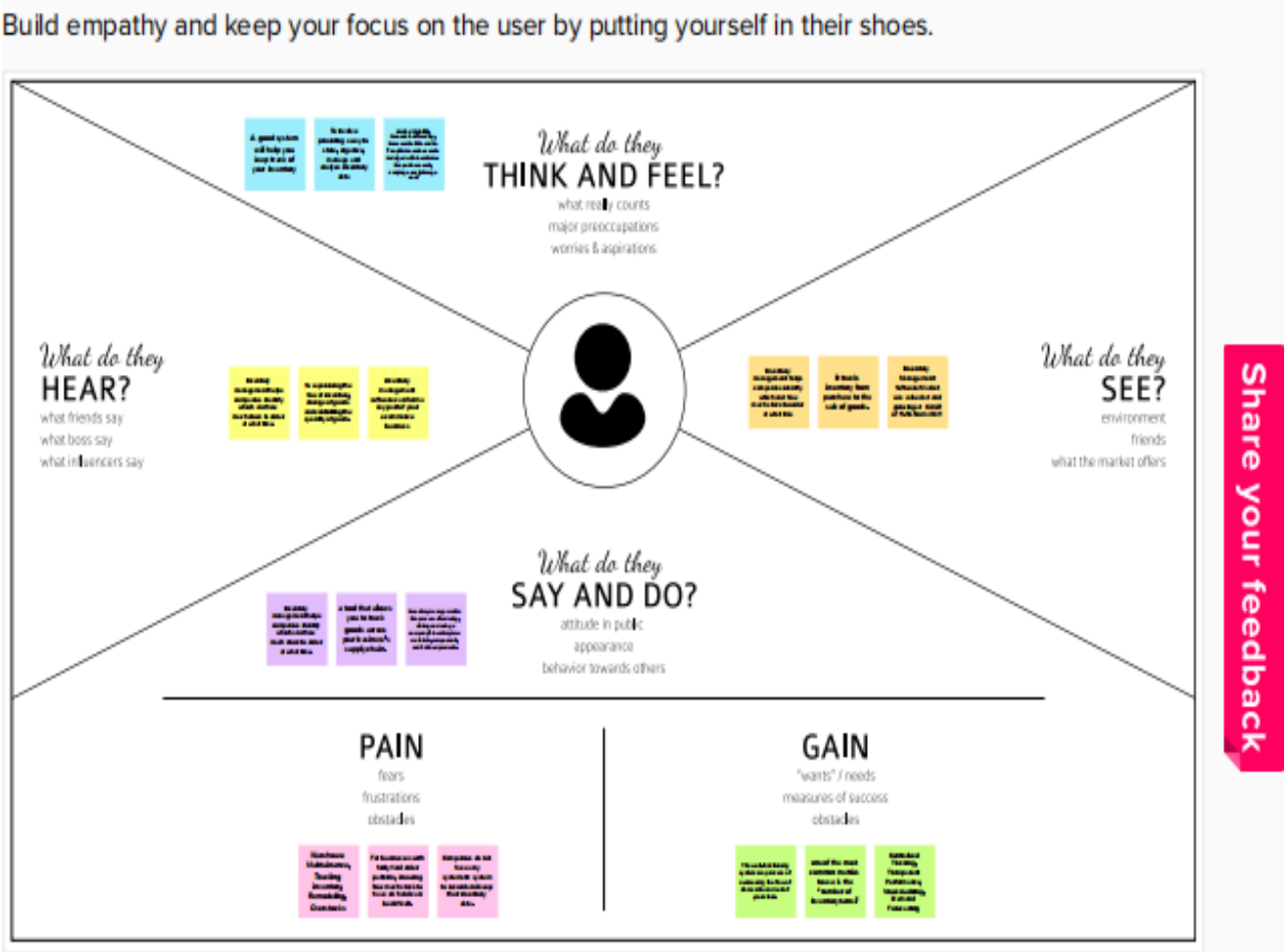
2.3 References:

1. Abisoye, O. A., Boboye, F., & Abisoye, B. O. (2013). Design of a

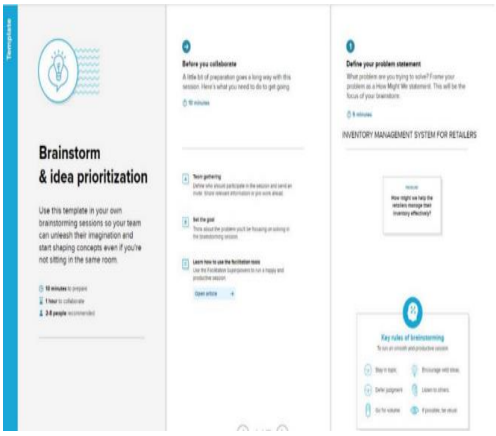
computerized inventory management system for supermarkets.
2. Li, S. G., & Kuo, X. (2008). The inventory management system for automobile spare parts in a central warehouse. Expert Systems with Applications, 34(2), 1144-1153.
3. Saleem, A. (2020). Automated inventory management systems and its impact on supply chain risk management in manufacturing firms of Pakistan. Int J Supply Chain Manag, 9, 220-231.
4. Bose, R., Mondal, H., Sarkar, I., & Roy, S. (2022). Design of smart

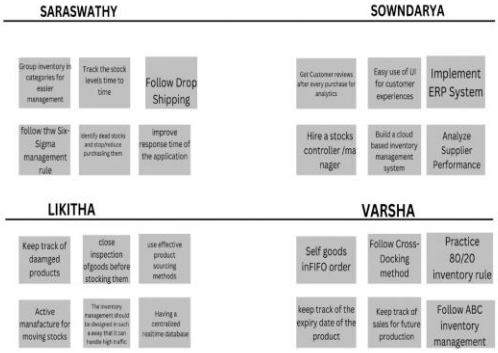
inventory management system for construction sector based on IoT and cloud computing. e-Prime-Advances in Electrical Engineering, Electronics and Energy, 2, 100051.
5. Madishetti, S., & Kibona, D. (2013). IMPACT OF INVENTORY MANAGEMENT ON THE PROFITABILITY OF SMES IN TANZANIA. CLEAR International Journal of Research in Commerce & Management, 4(2)
6. Kasim, H., Zubieru, M., & Antwi, S. K. (2015). An assessment of the inventory management practices of small and medium enterprises (SMEs) in the Northern Region of Ghana. *European Journal of Business and Management*, *7*(20), 28-40.
7. Panigrahi, C. M. A. (2013). Relationship between inventory management and profitability: An empirical analysis of Indian cement companies. *Asia Pacific Journal of Marketing & Management Review*, *2*(7).

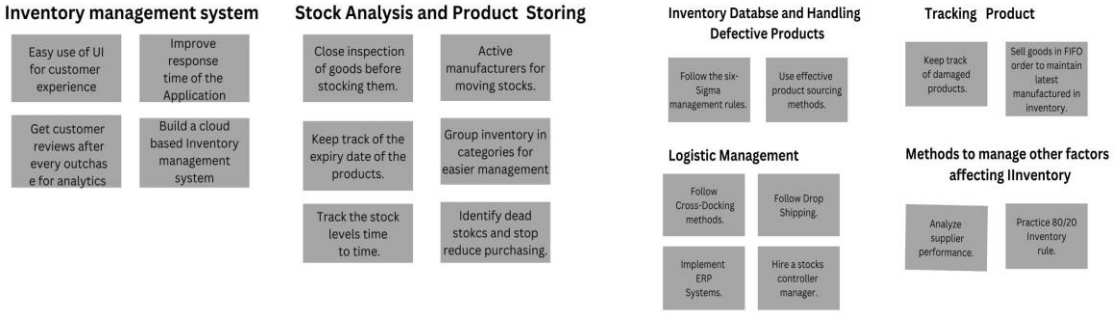
3. IDEATION & PROPOSED SOLUTION

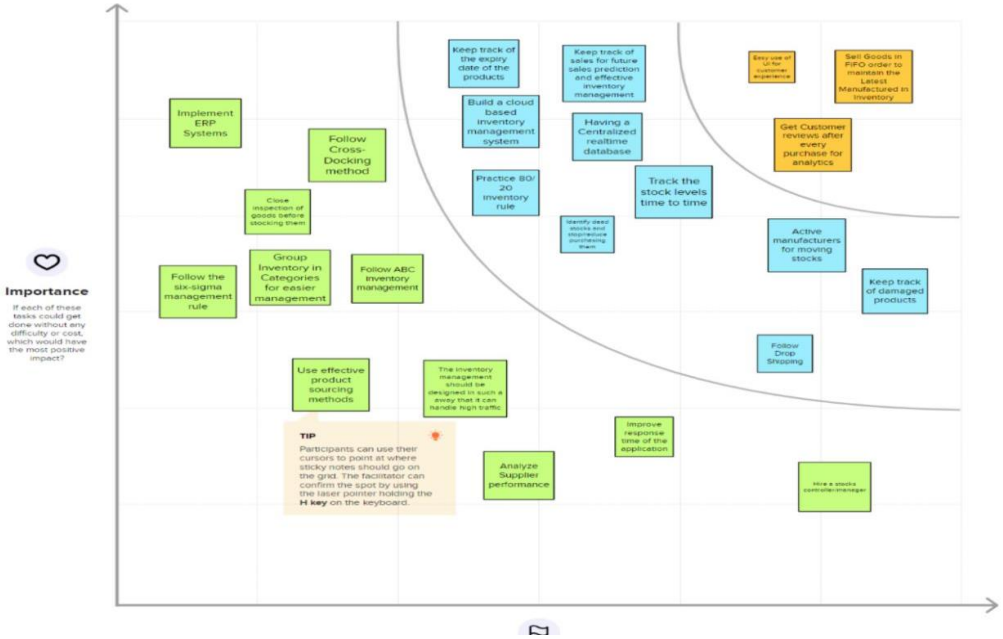


3.2 Ideation & Brainstorming:

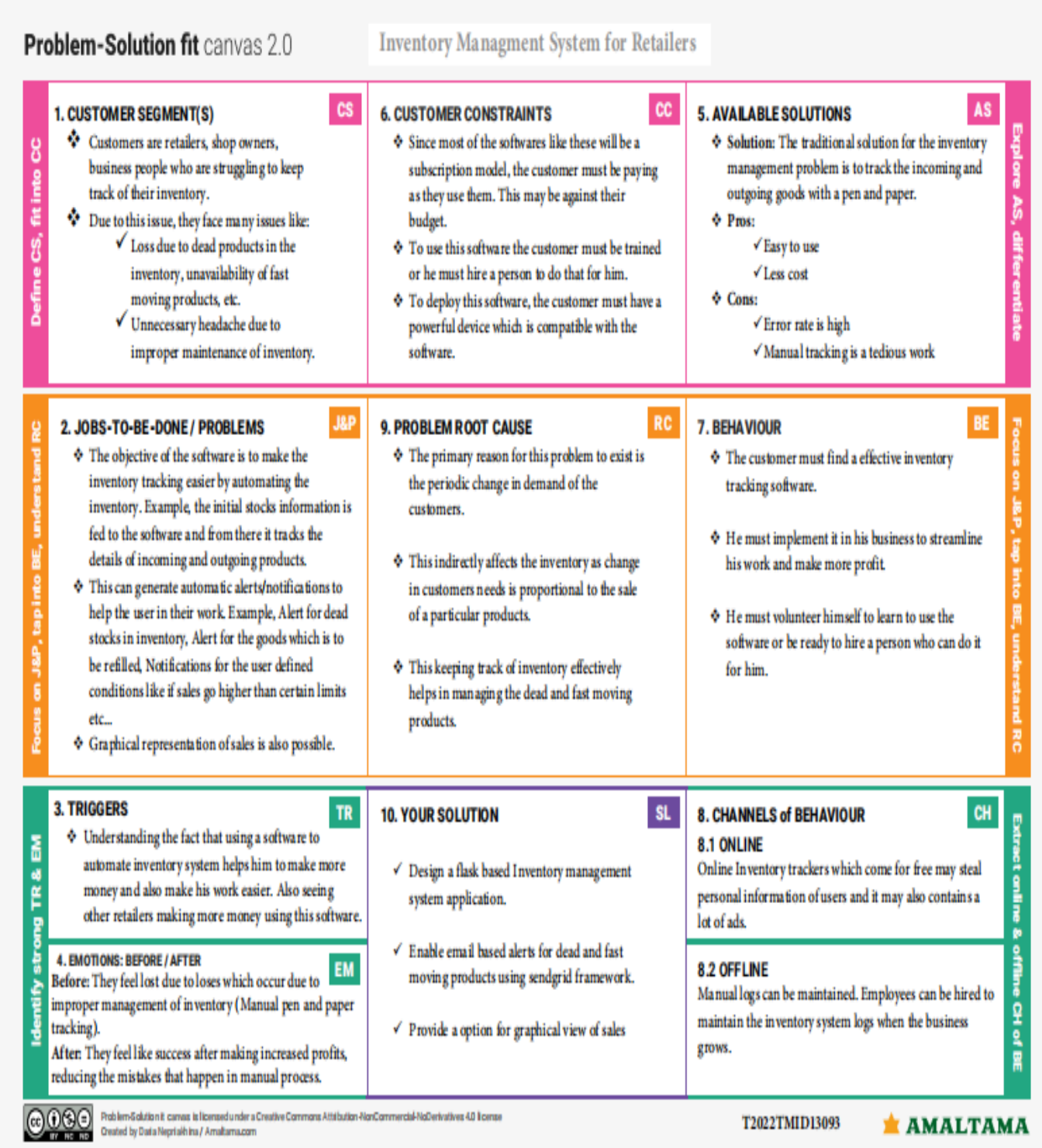








3.3 Proposed Solution fit:



3.4 Proposed Solution:

S.No	Parameter	Description
1	Problem Statement (Problem to be solved)	The problem to be solved is to make an application for retailers to track their inventory stocks to manage the purchases, sales, stocks, etc
2	Idea / Solution description	The idea to solve this is by developing an application to track and manage stocks related to their own products. The retailers create their accounts by proving their details and entering the stocks/inventory of their products. Once done, they can login to the application and view their stocks, sales, update their stocks when restocking, etc. They can see which stocks are fast moving and when in case of running out, they get notification and they can restock their fastmoving stocks.
3	Novelty / Uniqueness	As we have data of the sales per stocks, we can include a prediction of stocks to guess which will be the most purchased stocks so that the retailers can restock up on that prior. The data can be obtained by regression and previous sales data within our application. We can also make maintenance and development easier by containerizing via Docker application.
4	Social Impact / Customer Satisfaction	By using our application, we can see which stocks are being sold and which are not much as expected, so by using that data we can purchase and

		restock only the required stocks and thus reducing excess stocks in the inventory which might be a wastage of products. Since we will know which products are needed in bulk, we can request vendors and suppliers the required number of stocks and negotiate better deals with them beforehand.
5	Business Model (Revenue Model)	Retailers can order the fast moving products and the right number of stocks from suppliers and vendors by analysing the predicted products which has higher chance of being purchased in large amounts, and thus eliminating unnecessary redundant products which might be excess when not ordered in the right amount
6	Scalability of the Solution	Scalable cloud architecture is made possible through virtualization... Unlike physical machines whose resources and performances are set by their physical hardware, processors and memory. Virtual Machines that we use in IBM Cloud are highly flexible and scalable. Kubernetes allows the users to scale the containers based on the application requirements which may vary over time. It's easy to change the number via command lines

4. REQUIREMENT ANALYSIS

4.1 Functional requirement:

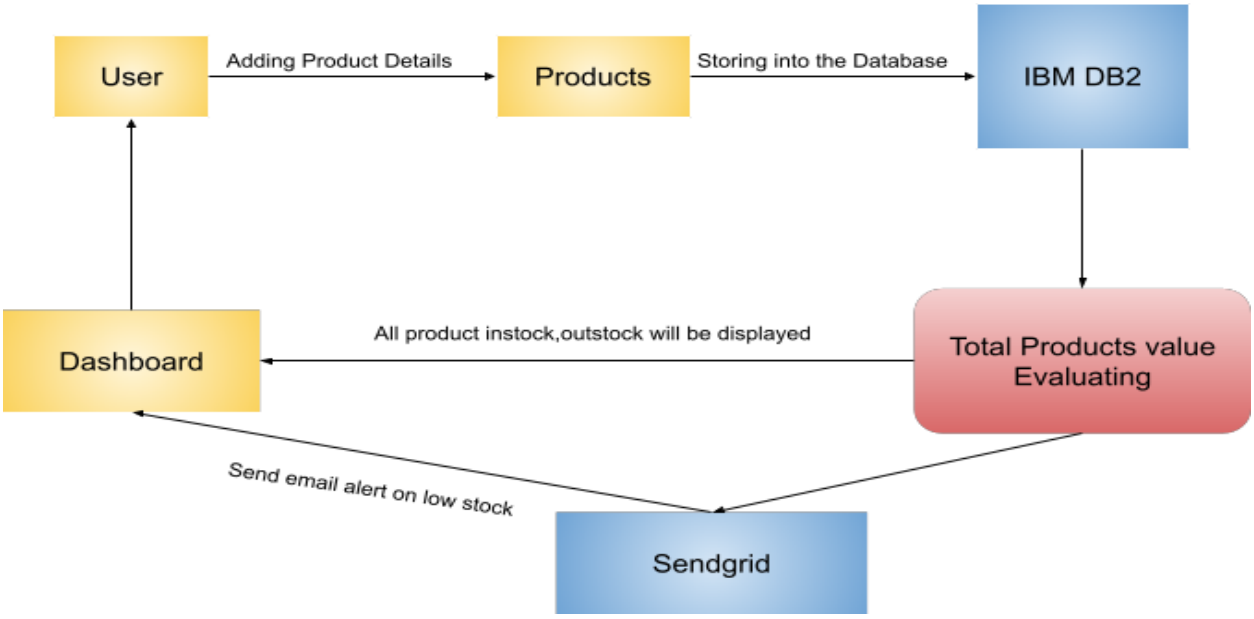
FR. No.	Functional Requirement (Epic)	Sub Requirement (Story/Sub-Task)
FR-1	User Registration	Registration through registration form. Registration through One-Tap Google Sign in.
FR-2	User Authentication and Confirmation	Authentication via Google Authentication. Confirmation via Email. Confirmation via OTP.
FR-3	Product management	Quickly produce reports for single or multiple products. Track information of dead and fast-moving products. Track information of suppliers and manufacturers of the product.
FR-4	Audit Monitoring	The technique of tracking crucial data is known as audit tracking. Monitor the financial expenses carried out throughout the whole time (from receiving order of the p
FR-5	Historical Data	Data of everything should be stored for analytics and forecasting.
FR – 6	CRM (Customer Relationship Management)	Track the customer experience via ratings given by them. Get customer reviews regularly or atleast at the time of product delivery to work on customer satisfaction. .

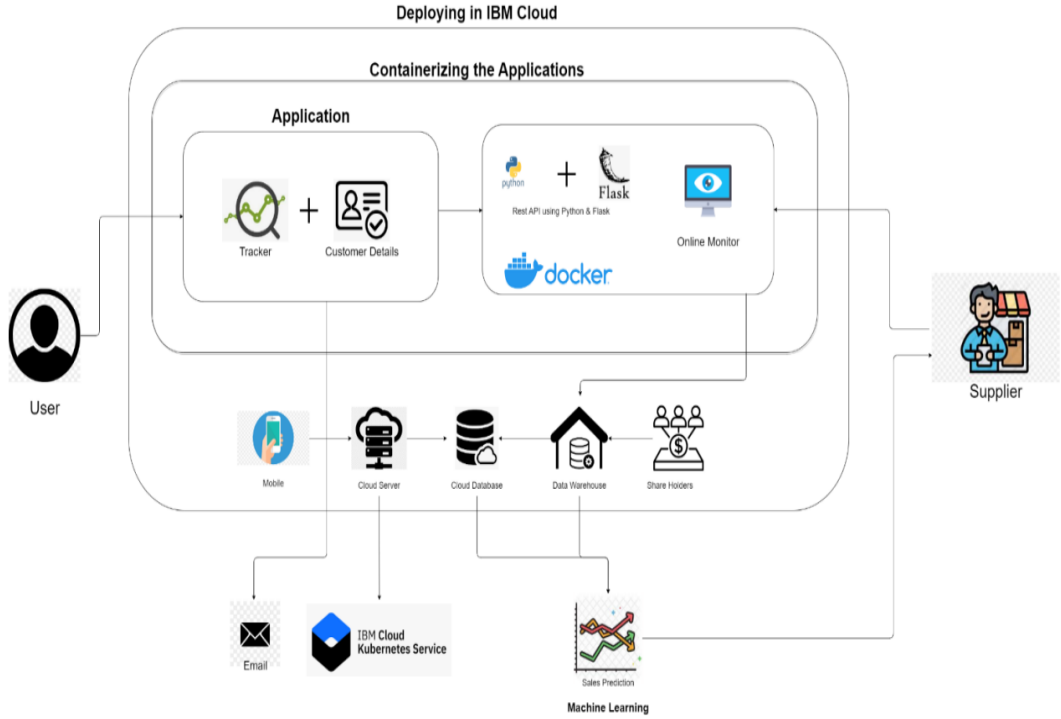
4.2 Non Functional requirement:

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The UI should be accessible to everybody despite of there diversity in languages. People with some impairments should also be able to use the application with ease. (Example, integrate google assistant so that blind people can use it)
NFR-2	Security	The security requirements deal with the primary security. Only authorized users can access the system with their credentials. Administrator or the concerned security team should be alerted on any unauthorized access or data breaches so as to rectify it immediately.
NFR-3	Reliability	The software should be able to connect to the database in the event of the server being down due to a hardware or software failure.
NFR-4	Performance	Performance of the app should be reliable with high-end servers on which the software is running
NFR-5	Availability	The software should be available to the users 24/7 with all functionalities working. New module deployment should not impact the availability of existing modules and their functionalities
NFR-6	Scalability	The whole software deployed must be easily scalable as the customer base increases.

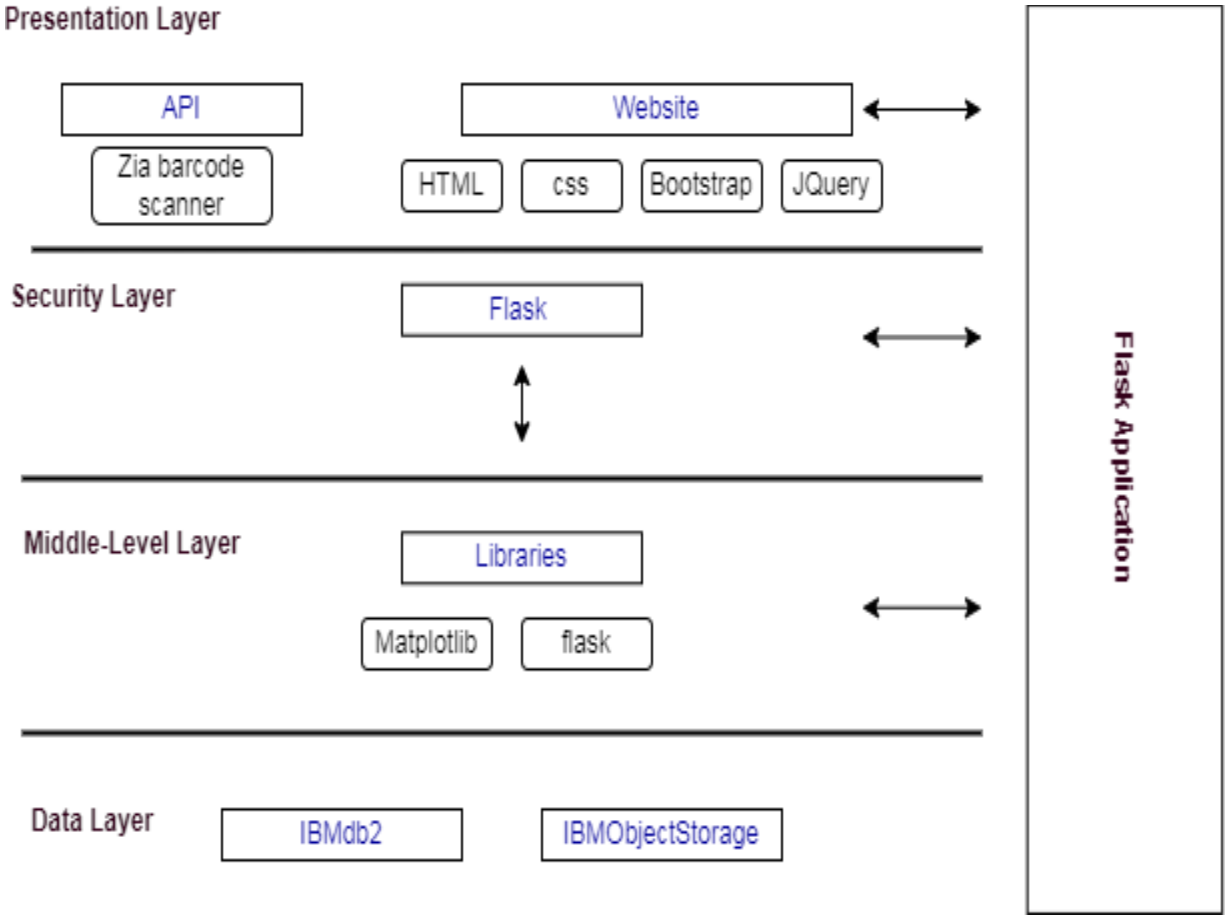
5. PROJECT DESIGN

5.1 Data Flow Diagrams:





5.2 Solution & Technical Architecture:



6. PROJECT PLANNING & SCHED ULING

6.1 Sprint Planning & Estimation:

Sprint-1	Dashboard	USN-5	As a user, I must be able to see my details on the dashboard.	3	High	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-2	Dashboard	USN-6	As a user, I should be able to change password whenever I prefer.	2	Medium	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-1	Inventory	USN-7	As a retailer, I should be able to alter product details in the app	2	Medium	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-2	Inventory	USN-8	As a retailer, I should be able to add or remove quantity of products in the app.	3	Medium	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-2	Inventory	USN-9	As a retailer, I should get alert on stock shortage or unavailability.	5	Medium	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-1	Order	USN-7	As a user, I should be able to order items on theapp	2	High	Likitha T Sowndarya S Saraswathy V **Varsha R**
Sprint-1	Payment	USN-8	As a user, I should be able to verify and pay in asecure payment gateway	3	High	Likitha T Sowndarya S Saraswathy V **Varsha R**
Sprint-3	Order status	USN-9	As a user, I should be able to get the product ontime.	5	Low	Likitha T Sowndarya S Saraswathy V **Varsha R**
Sprint-4	Maintenance	USN-1	As a administrator, I should be able to edit details of the users of the app.	8	High	Likitha T Sowndarya S Saraswathy V **Varsha R**
Sprint-4	Maintenance	USN-2	Termination user accounts temporarily or permanently if needed.	5	Low	Likitha T Sowndarya S Saraswathy V **Varsha R**

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 for the application by entering my email, password, and confirming my password.	5	High	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-2	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	3	Medium	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-4	Registration	USN-3	As a user, I can register for the application through Facebook	8	Low	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-3	Registration	USN-4	As a user, I can register for the application through Gmail	8	High	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	5	High	Likitha T Sowndarya S Saraswathy V Varsha R
Sprint-2	Login	USN-4	As a user, I can login into the application through Google one Tap Sign in	3	Medium	Likitha T Sowndarya S Saraswathy V Varsha R

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	18	6 Days	31 Oct 2022	05 Nov 2022	18	05 Nov 2022
Sprint-3	21	6 Days	07 Nov 2022	12 Nov 2022	21	12 Nov 2022
Sprint-4	21	6 Days	14 Nov 2022	19 Nov 2022	21	19 Nov 2022

6.2 Sprint Delivery Schedule:

Sprint	Total StoryPoints	Duration	Sprint StartDate	SprintEndDate (Planned)	Story Points Completed (as on Planned EndDate)	Sprint Release Date(Actual)
Sprint-1	20	6Days	24Oct2022	29Oct2022	20	29Oct2022
Sprint-2	20	6Days	31Oct2022	05Nov2022	20	03Nov2022
Sprint-3	20	6Days	07Nov2022	12Nov2022	20	10Nov2022
Sprint-4	20	6Days	14Nov2022	19Nov2022	20	17Nov2022

7. CODING & SOLUTIONING

7.1 Feature 1

Python

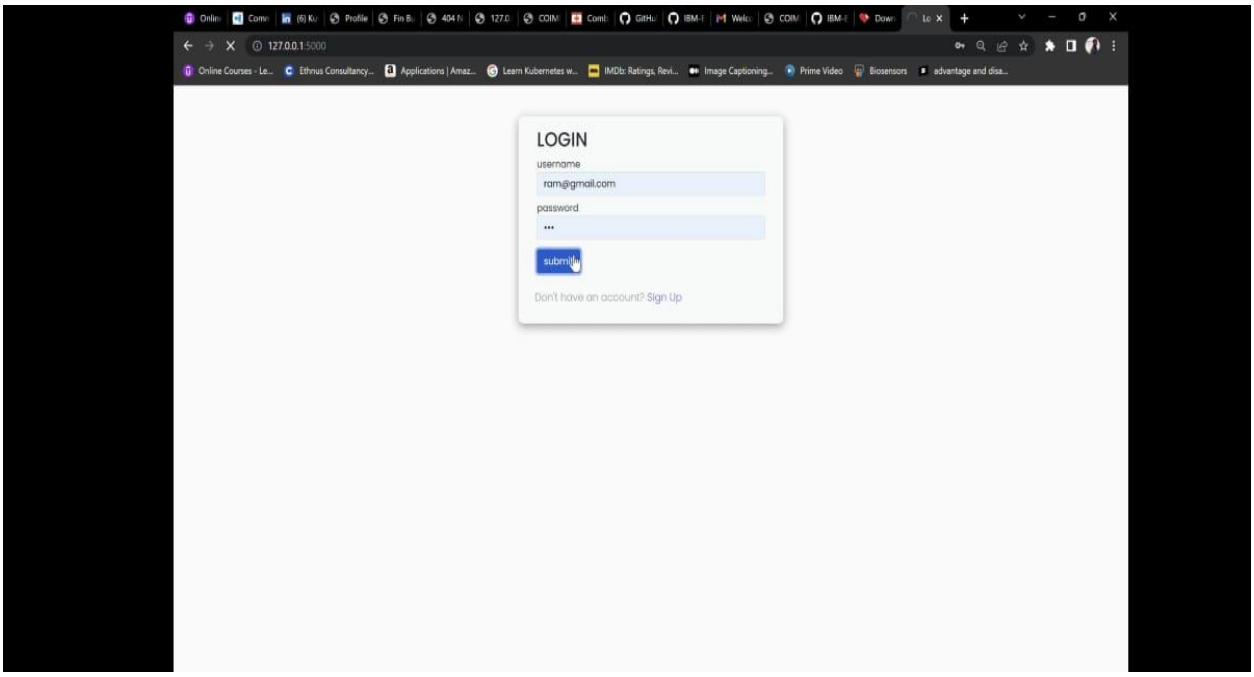
Html

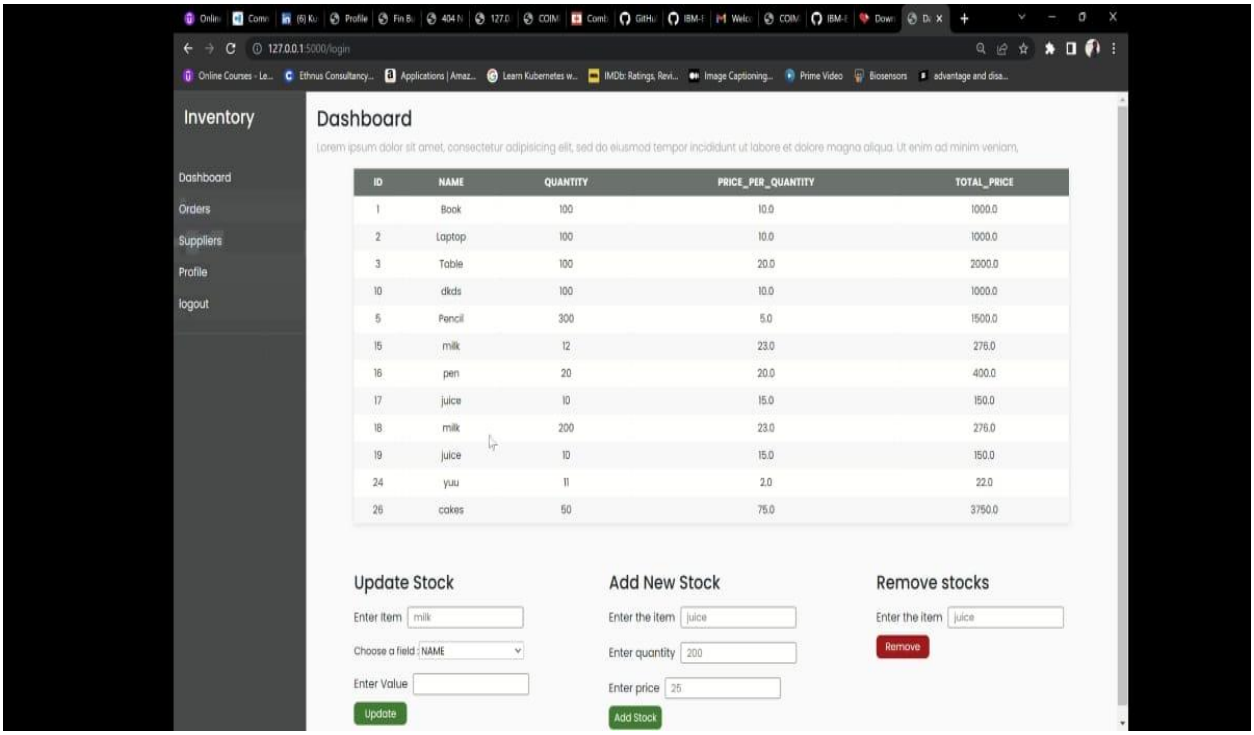
Css

Java script

Sql

8. Testing And Results:





9. Advantages:

It helps to maintain the right amount of stocks.

It leads 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.

10. Disadvantages:

Bureaucracy.

Impersonal touch.

Increased space is need to hold the inventory.

Complexity.

High implementation costs.

11. Conclusion:

Inventory management is a very complex but essential part of the supply chain. An effective inventory management system helps to reduce stock-related costs such as warehousing, carrying, and ordering costs. As you have read above, there are different techniques that businesses can utilize to simplify and optimize stock management processes and control systems.

13. Appendix

13.1 Source Code:

[10:25 pm, 20/11/2022] Mike: from sendgrid import *

#creating an app instance

app = Flask(_name_)

```
app.secret_key='a'


conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=0c77d6f2-5da9-48a9-81f8-
86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31198;SECURITY=S
SL;SSLSererCertificate=DigiCertGlobalRootCA.crt;UID=ldy44383;PWD=iW50uSQCx3l7ckXh
","","")


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


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


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

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

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

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

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

    if result>0:
        return render_template('locations.html', locations = locations)
    else:
```

```
        msg='No locations found'
        return render_template('locations.html', msg=msg)

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

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

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

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

#Register Form Class
```

```
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
    username = StringField('Username', [validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6, max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')


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


        sql1="INSERT INTO users(name, email, username, password) VALUES(?,?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,name)
        ibm_db.bind_param(stmt1,2,email)
        ibm_db.bind_param(stmt1,3,username)
        ibm_db.bind_param(stmt1,4,password)
```

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

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


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

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

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

                flash("you are now logged in","success")
                return redirect(url_for('dashboard'))
            else:
                error = 'Invalid Login'
                return render_template('login.html', error=error)
            #Close connection
            cur.close()
        else:
            error = 'Username not found'
            return render_template('login.html', error=error)
    return render_template('login.html')

#check if user logged in
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
```

```
            flash('Unauthorized, Please login','danger')
            return redirect(url_for('login'))
    return wrap

#Logout
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("You are now logged out", "success")
    return redirect(url_for('login'))

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

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


    products=[]
```

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

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

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

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

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

```
    product_cost = StringField('Product Cost', [validators.Length(min=1, max=200)])
    product_num = StringField('Product Num', [validators.Length(min=1, max=200)])

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

        sql1="INSERT INTO products(product_id, product_cost, product_num) VALUES(?,?,?)"
        stmt1 = ibm_db.prepare(conn, sql1)
        ibm_db.bind_param(stmt1,1,product_id)
        ibm_db.bind_param(stmt1,2,product_cost)
        ibm_db.bind_param(stmt1,3,product_num)

        ibm_db.execute(stmt1)

        flash("Product Added", "success")

        return redirect(url_for('products'))

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

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

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

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

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

```
        sql2="UPDATE products SET product_id=?,product_cost=?,product_num=? WHERE
product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,product_id)
        ibm_db.bind_param(stmt2,2,product_cost)
        ibm_db.bind_param(stmt2,3,product_num)
        ibm_db.bind_param(stmt2,4,id)
        ibm_db.execute(stmt2)

        flash("Product Updated", "success")

        return redirect(url_for('products'))

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

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

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

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

```
    return redirect(url_for('products'))


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


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


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


        flash("Location Added", "success")


        return redirect(url_for('locations'))


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

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

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

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

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

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

```
        flash("Location Updated", "success")

        return redirect(url_for('locations'))

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

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

    flash("Location Deleted", "success")

    return redirect(url_for('locations'))

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

```
class CustomError(Exception):
    pass

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

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

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



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

    locations=[]
```

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

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

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

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

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

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



elif from_location=="Main Inventory":
    sql2="SELECT * from product_balance where location_id=? and product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
    ibm_db.bind_param(stmt2,1,to_location)
    ibm_db.bind_param(stmt2,2,product_id)
    result=ibm_db.execute(stmt2)
    result=ibm_db.fetch_assoc(stmt2)
    print("------------------")
    print(result)
    print("------------------")
    app.logger.info(result)
    if result!=False:
        if(len(result))>0:
            Quantity = result["QTY"]
            q = Quantity + qty


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

```
            ibm_db.execute(stmt2)

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

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

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

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

        sql2="Update products set product_num=? where product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
        ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-qty)
        ibm_db.bind_param(stmt2,2,product_id)
        ibm_db.execute(stmt2)

        alert_num=current_num['PRODUCT_NUM']-qty

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

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

```
        result=ibm_db.execute(stmt2)
        result=ibm_db.fetch_assoc(stmt2)

        app.logger.info(result)
        if result!=False:
            if(len(result))>0:
                Quantity = result["QTY"]
                q = Quantity - qty

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

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

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

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

            sql2="Update products set product_num=? where product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
            ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']+qty)
            ibm_db.bind_param(stmt2,2,product_id)
            ibm_db.execute(stmt2)

            alert_num=q
            if(alert_num<=0):
                alert("Please Add {} number of {} to {} warehouse!".format(-
q,product_id,from_location))
        else:
            raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))



    else: #will be executed if both from_location and to_location are specified
        f=0
        sql = "SELECT * from product_balance where location_id=? and product_id=?"
```

```
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,from_location)
        ibm_db.bind_param(stmt,2,product_id)
        result=ibm_db.execute(stmt)
        result = ibm_db.fetch_assoc(stmt)

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

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

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

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

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

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

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

            else:

                sql2="INSERT into product_balance(product_id, location_id, qty) values(?, ?, ?)"
                stmt2 = ibm_db.prepare(conn, sql2)
```

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

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

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


        return redirect(url_for('product_movements'))

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

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

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

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

    return redirect(url_for('product_movements'))

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

13.2 GitHub:

https://github.com/IBM-EPBL/IBM-Project-34293-1660233892