PROJECT REPORT

Inventory Managment System For Retailers

TEAM ID: PNT2022TMID42656

JANSONS INTITUTE OF TECHNOLOGY, COIMBATORE

SUBMITTED BY,

MANISANKAR P - 711119106016

THAMIZHINI A - 711119106030

GOKUL S - 711119106010

SATHISH KUMAR B - 711119106024

1. INTRODUCTION:

1.1 PROJECT OVERVIEW

In the real time scenario, the retailers without knowing the demand for products, they bought excess stock which is not a popular one among the customers. Retail inventory management system is the process of ensuring you cary 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. Smart inventory management system is a project which will satisfy the retail shop business men.

1.2 PURPOSE

Customers or retailers can use this software to generate revenue with a high profit by avoiding the purchase of non selling products. This software will help the customer to get the information about product availability by analysing the list of product as soon as possible which helps to avoid the delay in purchase. By using this profitable income can be earned by customer

2. LITERATURE SURVEY:

2.1 EXISTING PROBLEMS:

In the real time scenario, the retailers without knowing the demand for products, they bought excess stock which is not a popular one among the customers. Retail inventory management system is the process of ensuring you cary 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. Smart inventory management system is a project which will satisfy the retail shop business men.

2.2 REFERENCE:

- A case study of inventory management system for an internattional lifestyle product retailer in Bolivia.

Author - Boris Herbas Torrico, Sebastian Alem Oyola - Universidad Catolica Boliviana San Pablo Cochabamba.
Outcome - The implementation of reactive flexibility practices can greatly improve inventory practices Yet, the limited sample size of their study limits the generalizability of results. The particular business characteristics of case study limit the capacity to extend forecasting results to other products and industries. However, the framework of strategies used to reduce supply uncertainty can be used in other industries.

- Improving Inventory Management in the Retail Store: The effectiveness of RFID Tagging across Product Categories.

Author - Bill C. Hardgrave - Auburn University, Sandeep Goyal - University of Louisville, John Aloysious - University of Arkansas.
Outcome - RFID tagging ameliorates the effects of five (item cost, sales velocity, Sales volume, inventory density and product variety) of these determinants of inventory record inaccuracy(they experimentally control for the effects of the other two determinants: audit frequency and distribution structure).For example, the(positive)influence of sales velocity on inventory record inaccuracy is moderated by inventory visibility due to RFID tagging.

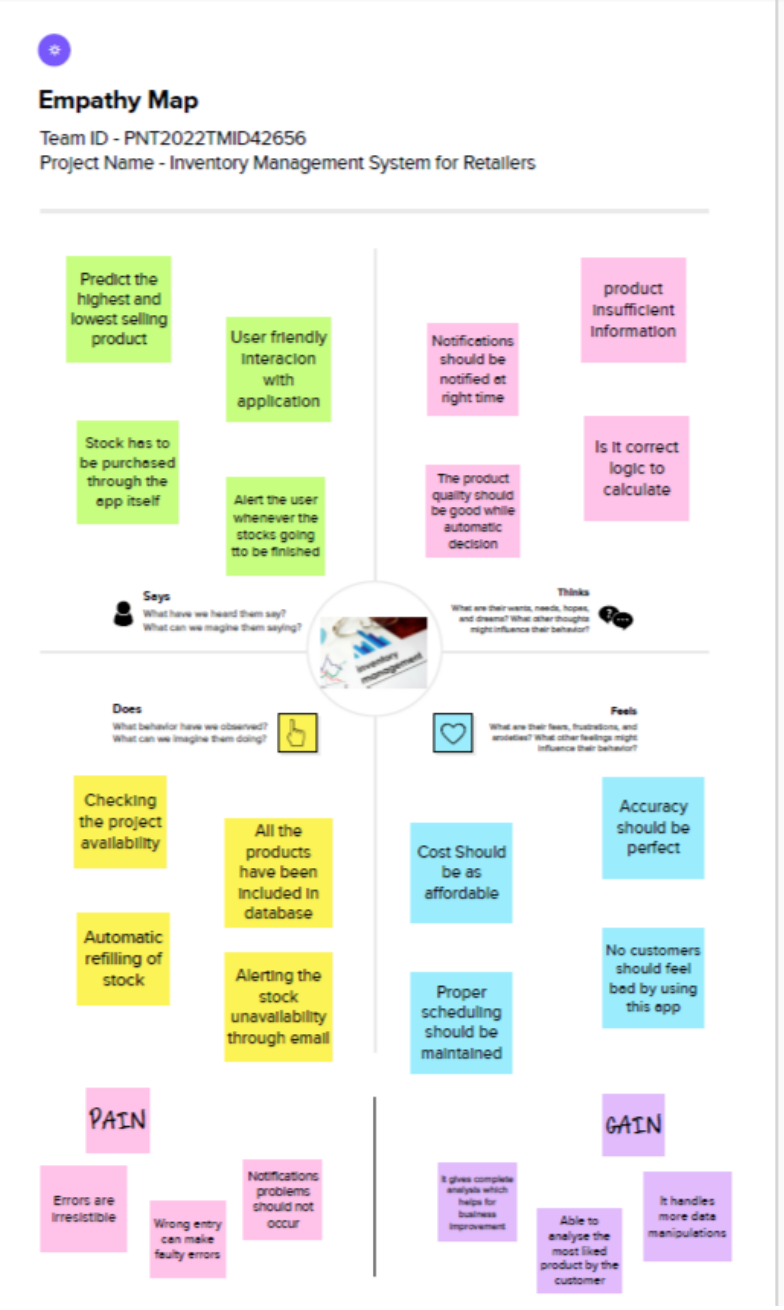
PROBLEM STATEMENT DEFINITION:

In the real time scenario, the retailers without knowing the demand for products, they bought excess stock which is not a popular one among the customers. Retail inventory management system is the process of ensuring you cary 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. Smart inventory management system is a project which will satisfy the retail shop business men. 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

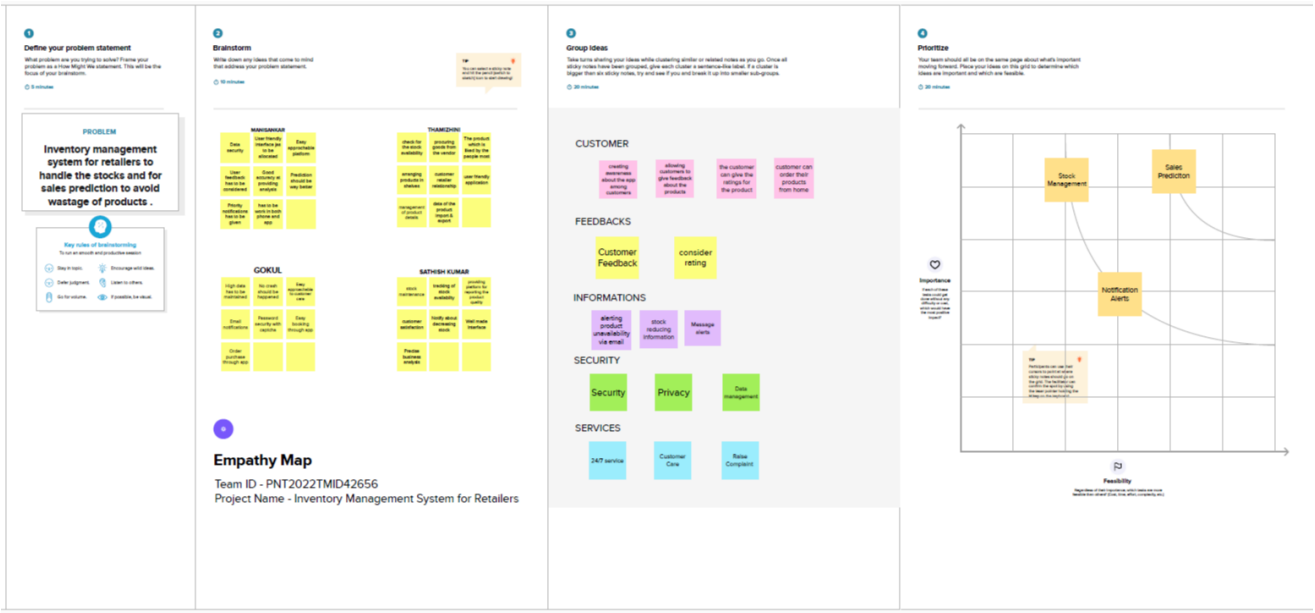
carrying excess supply. The system will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION AND BRAINSTORMING:



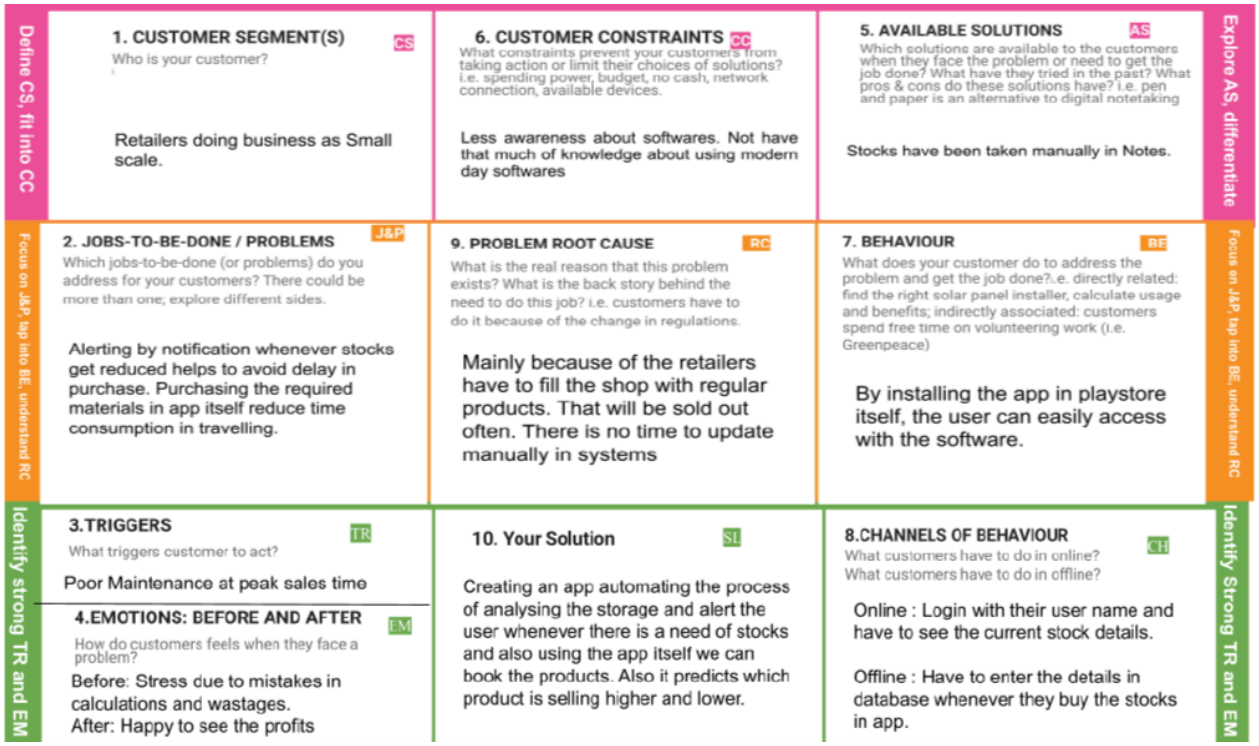
3.3 PROPOSED SOLUTION:

S.No.	Parameter	Description
•	Problem Statement (Problem to be solved)	In the real time scenario, the retailers without knowing the demand for products, they bought excess stock which is not a popular one among the customers. Retail inventory management system 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. Smart inventory management system is a project which will satisfy the retail shop business men.

•	Idea / Solution description	The key research objectives are as follows: • Either QR code or sensors or rfid tags will be fixed in the inventory shelves. This will help to analyze the product sales where we can find out the highest selling product and the demand of other products. • The software application will have the normal login procedure where the user has to login with their email. In case the stock is going to be completed, immediately it will alert the user. • Also the mail will be sent related to the highest selling product. Once the user knows that the product is not available in the inventory, the user can directly book through the app at that time itself.
•	Novelty / Uniqueness	As we have the sales data, we can able to predict the most purchased stocks so that the retailers can restock up on that prior. The data can be obtained by previous sales data within our application. We can also able to maintain and develop easily by containerizing via Docker application. Additionally we use RFID tags for automating the calculation of movement of products whenever it increases or decreases.
•	Social Impact / Customer Satisfaction	Helps the retailer to avoid buying the least selling product make them focus on highest selling product which gives them profit and avoid wastage of product after expiry date for sale. Helps the customer by analysing their need, always the product availability will be maintained by ensuring stock count automatically.

•	Business Model (Revenue Model)	Customers or retailers can use this software to generate revenue with a high profit by avoiding the purchase of non selling products. The prediction will help the customer to get the information about product availability as soon as possible which helps to avoid the delay in purchase. By using this profitable income can be earned by customer
•	Scalability of the Solution	The cloud makes it easy to add users, functionality, warehouses and suppliers—without the large-scale cost and effort required to implement a new on-premise system. Using kubernetes we can automate the deployment, scheduling and operation of application containers on clusters of machines.

3.4 PROBLEM SOLUTION FIT:



4 REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT:

Following are the functional requirements of the proposed solution:

Si.No	Functional Requirement (Epic)	Sub requirements(Story/Sub-Task)
1	User Registration	Through Web Application
2	User Confirmation	Through Mail and OTP
3	User Login	Login via Web App Initial Interface
4	Data Entry of Products	Through Web App Option for data entry
5	Displaying Analysis	Individual page of analysis in web app
6	E-Mail	Login alertness through smartGrid
7	Feedback	Customer feedback

4.2 NON FUNCTIONAL REQUIREMENT:

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

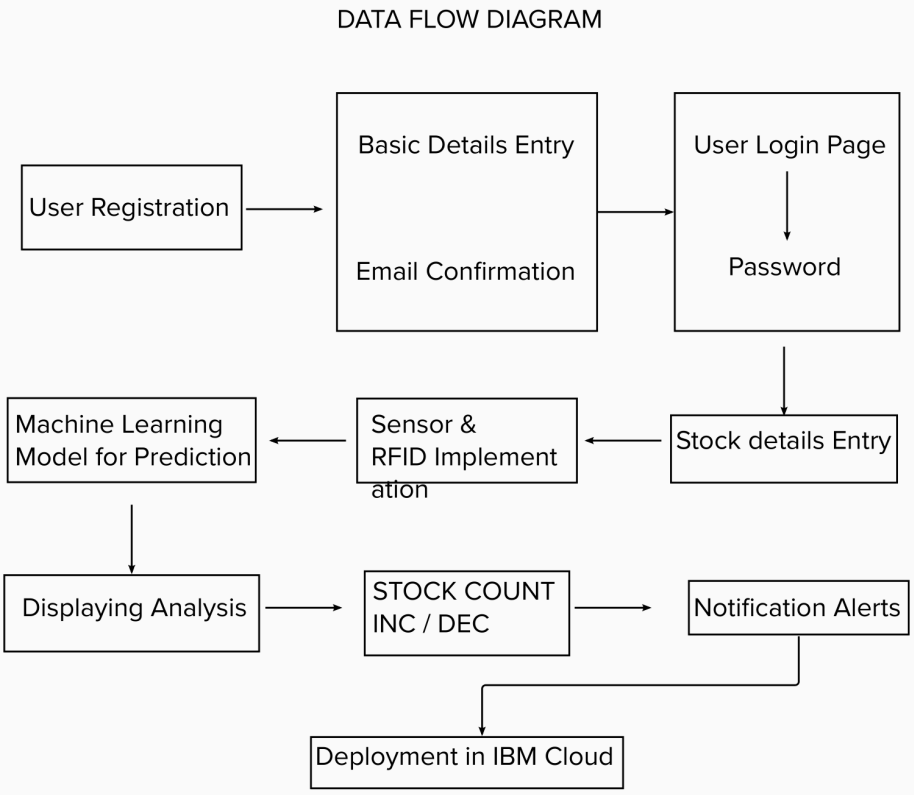
Si.No	Non-Functional Requirement	Description
1	Usability	To ease the process of inventory management
2	Security	Track of login authentication
3	Reliability	Tracking of decade status through email
4	Performance	Effective development of web application
5	Availability	24/7 service
6	Scalability	Products storage scalability based on shop size

5. PROJECT DESIGN:

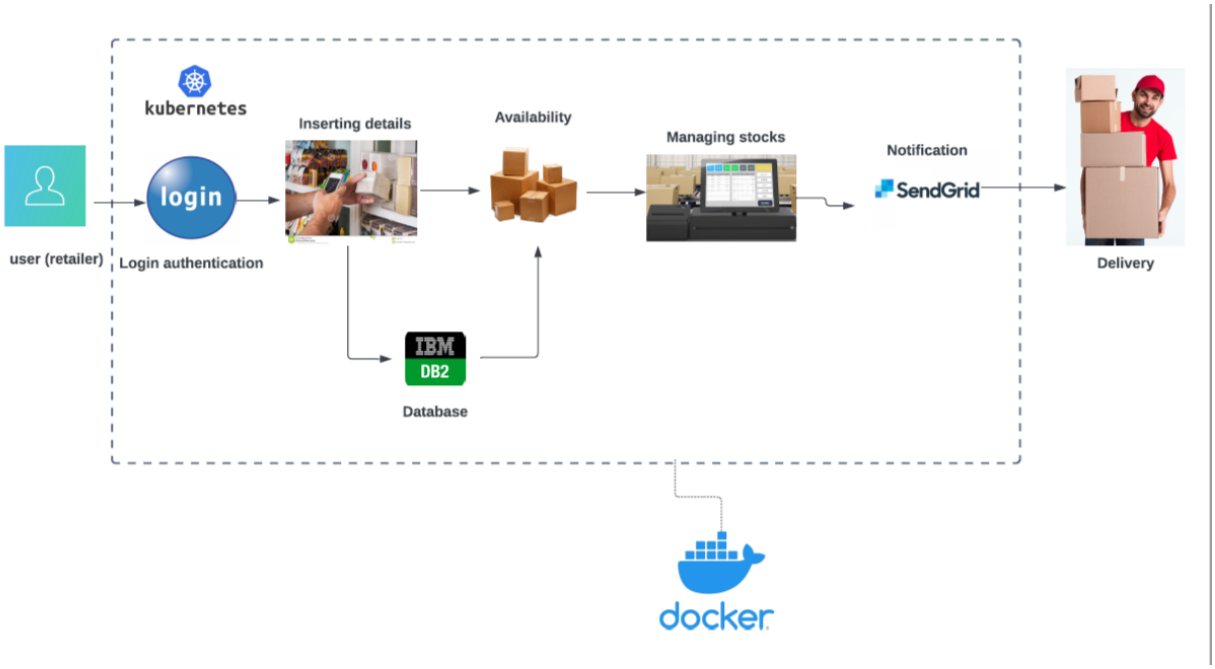
5.1 DATA FLOW DIAGRAMS:

PROBLEM DESIGN PHASE 2 - DATA FLOW DIAGRAM

Team ID - PNT2022TMID42656
Project Name - Inventory Management System for Retailers



5.2 SOLUTION AND TECHNICAL ARCHITECTURE:



5.3 USER STORIES:

STAGE	AWARENESS	CONSIDERATION	DECISION	SERVICE	LOYALTY
CUSTOMER ACTIVITIES	Advertising the app using social medias, news papers etc	Comparing features and pricing, make a demo of the app practically to the customers	Install the app and make a purchase	Details have to be updated then the analytics process will be completed	Sharing feedback and experience
TOUCHPOINTS	Social Media, Newspaper Ad, Word of Mouth	Social Media, Websites	Websites, Mobile App	Email	Social Media
CUSTOMER EXPERIENCE	Interested, Hesitant	Curious, Excited	Excited	Frustrated	Satisfied, Excited
KPIS	Customer feedback	New app visitors	Conversional rate	App workflow	Playstore feedback rating
RESPONSIBLE	Communications through email and phone	Communications	Customer service	Customer service	Customer service, Customer success

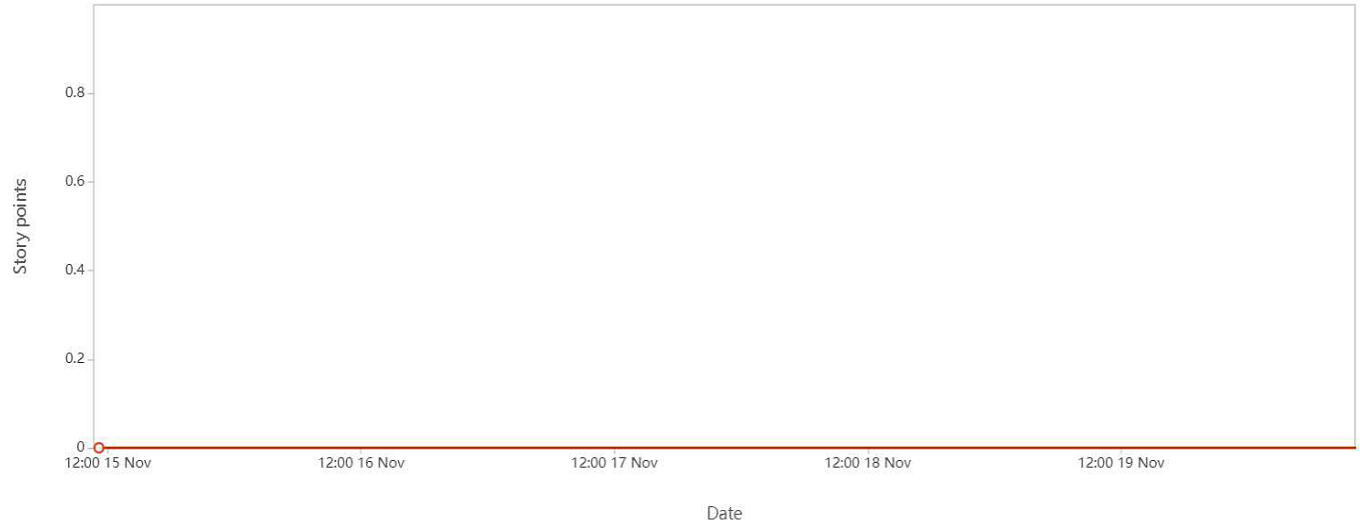
6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION, SPRINT DELIVERY

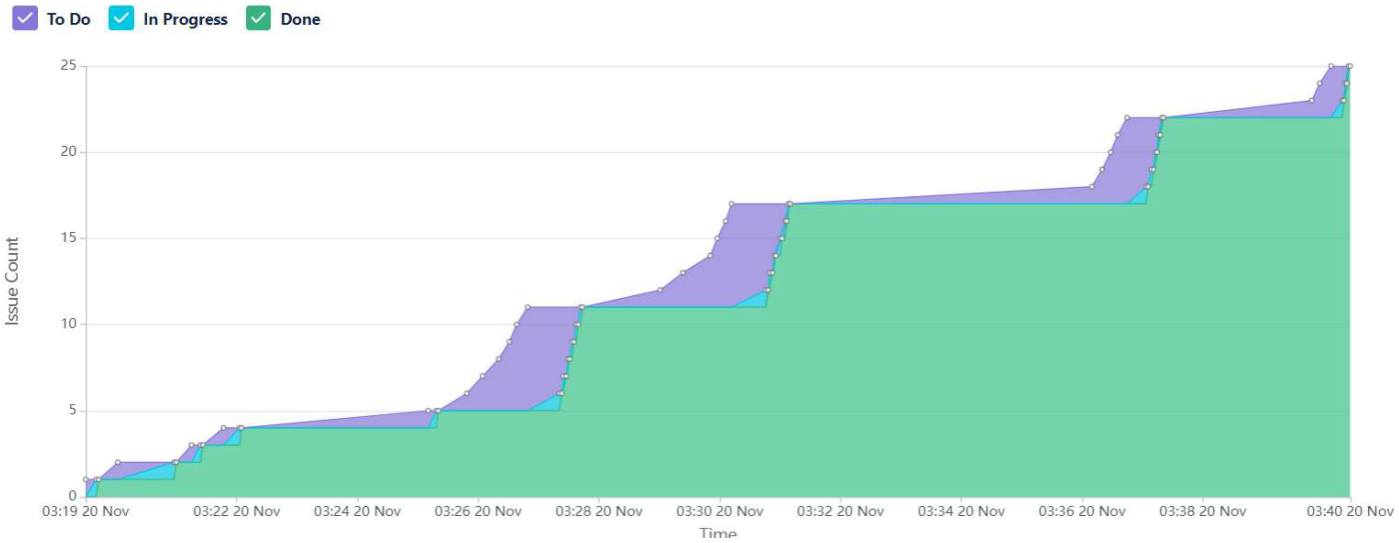
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my login credentials..	5	High	Manisankar P Thamizhini A Gokul S Sathishkumar B
Sprint-1		USN-2	As a user, I can register for the application through email.	3	High	Manisankar P Thamizhini A Gokul S Sathishkumar B
Sprint-1	Login	USN-3	As a user, I can log into the application by entering email & password	5	High	Manisankar P Thamizhini A Gokul S Sathishkumar B
Sprint-2	Dashboard	USN-4	As a user, I can view the products which are available	5	Medium	Manisankar P Thamizhini A Gokul S Sathishkumar B
Sprint-2	Add items to cart	USN-5	As a user, I can add the products which i have bought from the store	4	High	Manisankar P Thamizhini A Gokul S Sathishkumar B
Sprint-3	Stock update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list	5	High	Manisankar P Thamizhini A Gokul S Sathishkumar B

6.2 REPORTS FROM JIRA

i)SPRINT BURNDOWN CHART



ii)CUMMULATIVE CHART



7.CODING AND SOLUTION

SOURCE CODE FEATURES:

```
from flask import Flask, render_template, flash, redirect, url_for, session,
request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField,
validators, SelectField, IntegerField
import ibm_db
from passlib.hash import sha256_crypt
from functools import wraps
from sendgrid import *
app = Flask(__name__,template_folder='template')

app.secret_key='a'

conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=19af6446-6171-4641-8aba-
9dcff8e1b6ff.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30699;SECUR
ITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=fsv02618;PWD=T89J
Pw0608zxj3Lb",'','')
@app.route('/')
```

```
def index():
    return render_template('home.html')


@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)


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

@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)

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


@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)


    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)

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')

@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)
        flash("You are now registered and can log in", "success")

        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 QYT', [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,movementi_id,time) 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,movementi_id,time) 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 product_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)
```

8.USER ACCEPTANCE TESTING

DEFECT ANALYSIS

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	3	2	2	1	8
Duplicate	0	1	2	1	4
External	1	3	2	1	7
Fixed	4	2	3	15	24
Not Reproduced	0	0	1	1	2
Skipped	1	0	1	1	3
Won't Fix	2	3	2	1	8
Totals	11	11	13	21	56

TEST CASE ANALYSIS

Section	Total Cases	Not Tested	Fail	Pass
Login	8	0	0	8
Dashboard	19	0	0	19
Db2 Database	9	0	0	9
Flask Application	4	0	0	4

9.RESULTS

9.1.Performance Metrices

Performance metrices are figures and facts that quantify an organization's capabilities, actions, and general level of excellence. Profit, revenue, customer satisfaction, ROI, customer reviews, overall quality, individual reviews, and reputation in marketplace are only a few examples of different performance measures. Be aware that performance measurements can vary depending on which industry they are seen through. Performance metrics are crucial to the success of any firm. Since performance metrics support and ensure an organization's success, it is imperative for any business to select its performance metrics and the pay attention to those areas. Some crucial success factors are beneficial if they are monitored and acknowledged. Business measurements must be constantly monitored to ensure they are producing the important answers and that the appropriate questions are being posed.

10.ADVANTAGES AND DISADVANTAGES

10.1.Advantages

- Manage multiple warehouses
- Reduce Business Cost
- Greater Productivity
- Improve Supply chain
- Prevent Stock Overselling

10.2. Disadvantages

- Reduced physical audits
- Malicious hack
- System Crash
- Improper Inventory Tracking
- Loss of items

11. CONCLUSION

Maintaining precise records of the products that are prepared for transportation is a component of inventory management. This sometimes keeps a enough supply of the goods on hand to cover the inventory totals and deducting the most recent shipments of finished goods to customers. When a corporation has a return policy in place, the finishing goods inventory typically include a subcategory to account for any returned items that have been reclassified or are of second grade quality. It is feasible to promptly provide information to sales employees about what is available and prepared for shipmen tat any given time by the buyer by maintaining accurate numbers on the completed goods inventory. Inventory management is crucial for meeting regulations and reducing costs. Supply and demand must be carefully balanced.

12.FUTURE SCOPE

An inventory system can be used to value goods, track inventory changes, and prepare for future inventory levels, among other things. The balance sheet's financial reporting is based on the inventory value at the end of each period. The corporation can calculate the cost of goods sold during the time period by measuring

the change in inventory. As a result, the business can prepare for future inventory requirements.

Github Link : https://github.com/IBM-EPBL/IBM-Project-20818-1659764152