PROJECT REPORT

Date	18.11.2022
Team ID	PNT2022TMID45514
Project Name	Inventory Management System For Retailers
Team Members	T. Rajdeep Tukaram Patil R. Abishek M. Yokesh Raja M. Mohamed Nawas

1. INTRODUCTION

1.1 Project Overview

The objective of this system is to manage the items in an inventory such as tracking orders, placing orders to other suppliers and checking the items in the inventory. The system allows the admin to maintain the items in the inventory.

Whenever the item levels go low, the system places an order to the supplier. The supplier gets the notification of these orders as soon as they are placed and can send the items to the inventory. There are two login pages each for the admin and supplier.

The software has been developed using the most powerful and secured backend Python and IBM Cloud for the databases and most widely accepted frontend JavaScript with HTML and CSS coding

1.2 Purpose

The primary purpose of inventory management is to ensure there is enough goods or materials to meet demand without creating overstock, or excess inventory

Retail management refers to the process of helping customers find products in your store. It includes everything from increasing your customer pool to how products are presented, and how you fulfil a customer's needs. A good store manager helps customers leave the store with a smile.

2. LITERATURE SURVEY

2.1 Existing problem

- The problem faced by the company is they do not have any systematic system to record and keep their inventory data. It is difficult for the admin to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.
- Good planning and sales forecast before setting optimal inventory levels, appropriate inventory management requires close coordination between the areas of sales, purchasing and finance.

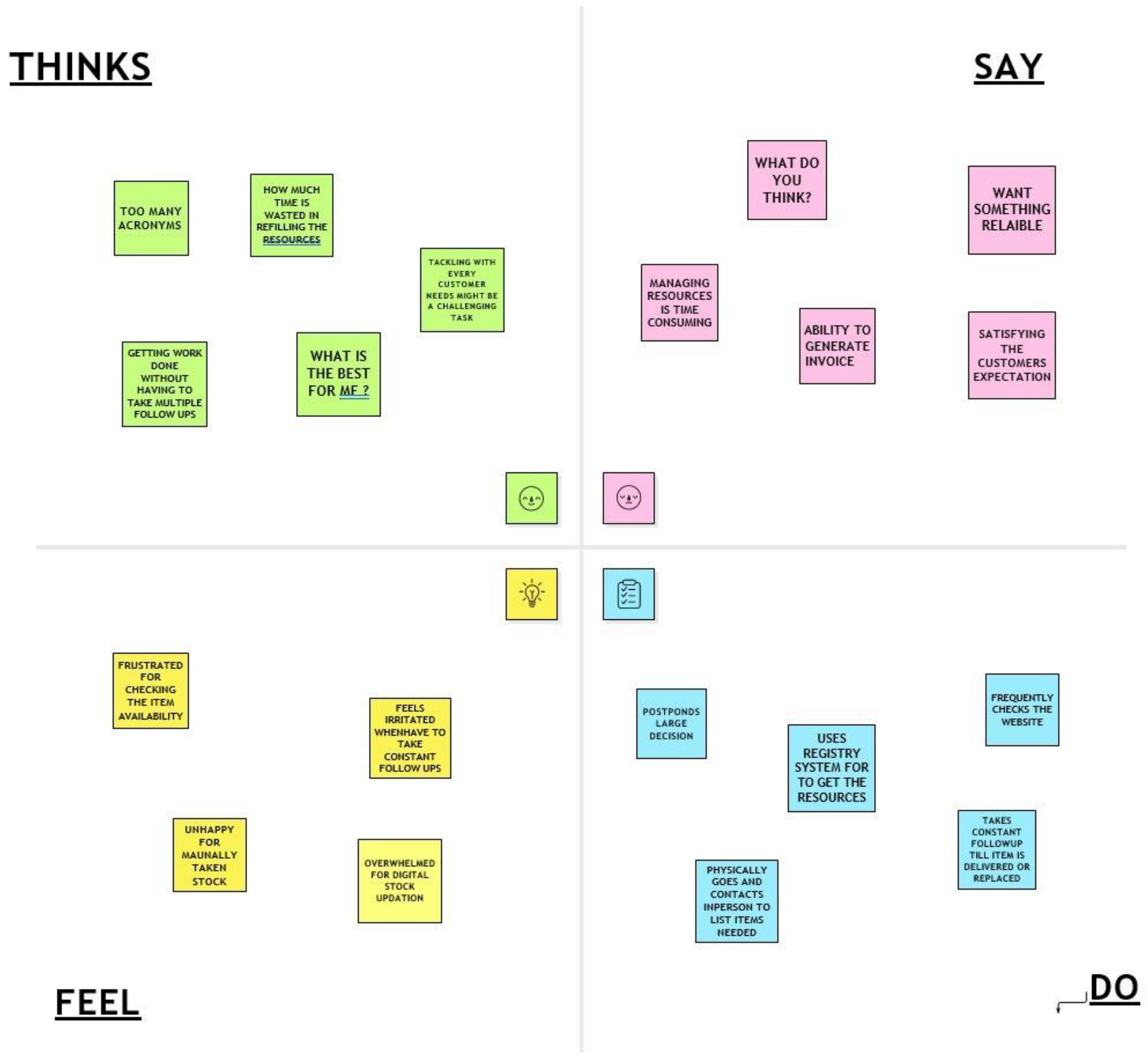
2.2 Problem Statement Definition

Retail inventory management works by creating systems to log products, receive them into inventory, track changes when sales occur, manage the flow of goods from purchasing to final sale and check stock counts.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

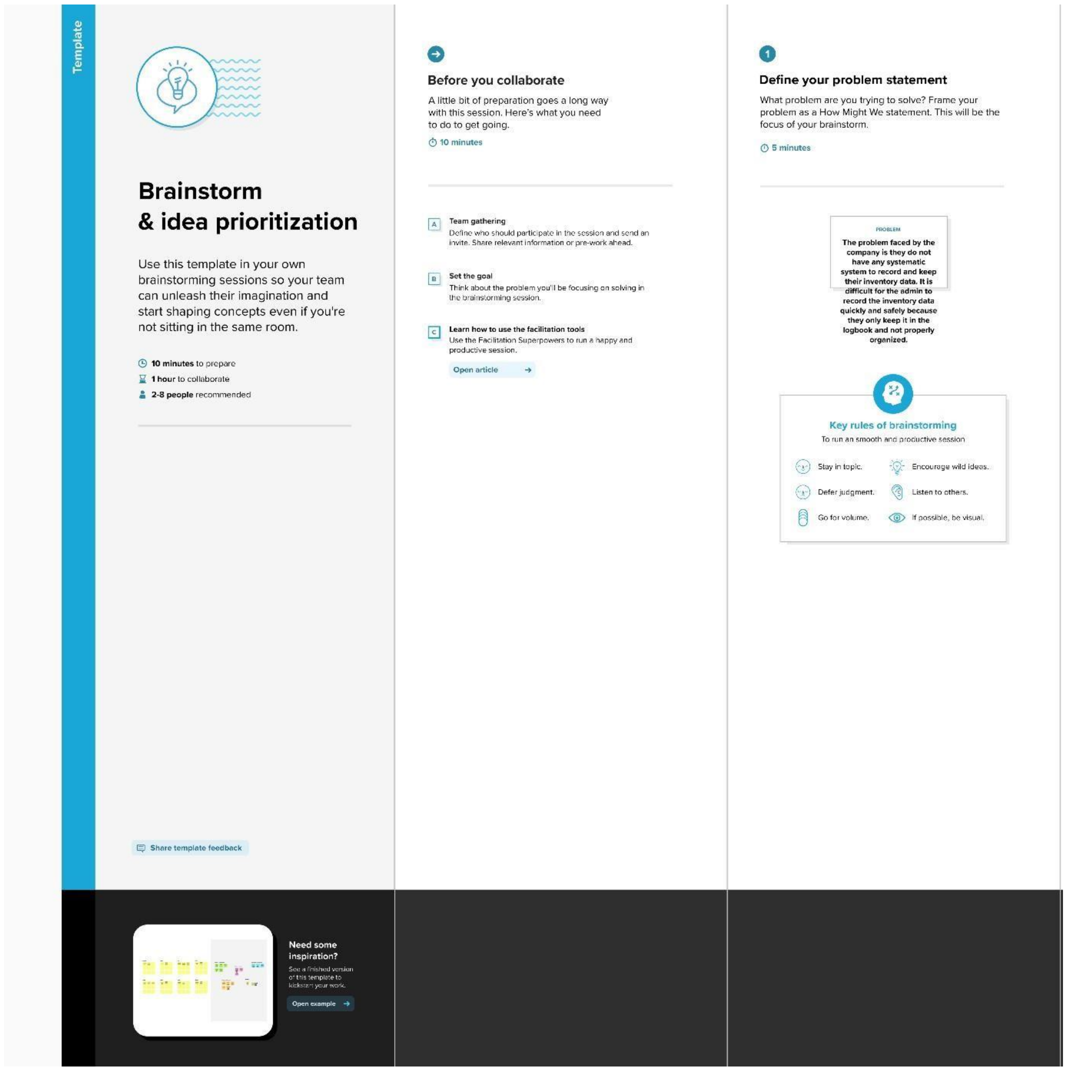

THINKS
SAY
TOO MANY ACRONYMS
HOW MUCH TIME IS WASTED IN REFILLING THE RESOURCES
TACKLING WITH EVERY CUSTOMER NEEDS MIGHT BE A CHALLENGING TASK
GETTING WORK DONE WITHOUT HAVING TO TAKE MULTIPLE FOLLOW UPS
WHAT IS THE BEST FOR ME ?
WHAT DO YOU THINK?
WANT SOMETHING RELAIBLE
MANAGING RESOURCES IS TIME CONSUMING
ABILITY TO GENERATE INVOICE
SATISFYING THE CUSTOMERS EXPECTATION
FRUSTRATED FOR CHECKING THE ITEM AVAILABILITY
FEELS IRRITATED WHENHAVE TO TAKE CONSTANT FOLLOW UPS
UNHAPPY FOR MAUNALLY TAKEN STOCK
OVERWHELMED FOR DIGITAL STOCK UPDATION
POSTPONDS LARGE DECISION
USES REGISTRY SYSTEM FOR TO GET THE RESOURCES
FREQUENTLY CHECKS THE WEBSITE
TAKES CONSTANT FOLLOWUP TILL ITEM IS DELIVERED OR REPLACED
PHYSICALLY GOES AND CONTACTS INPERSON TO LIST ITEMS NEEDED
FEEL
DO

3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Template

Brainstorm
& idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Share template feedback

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

Before you collaborate
A little bit of preparation goes a long way with this session. Here's what you need to do to get going.
10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.
Open article

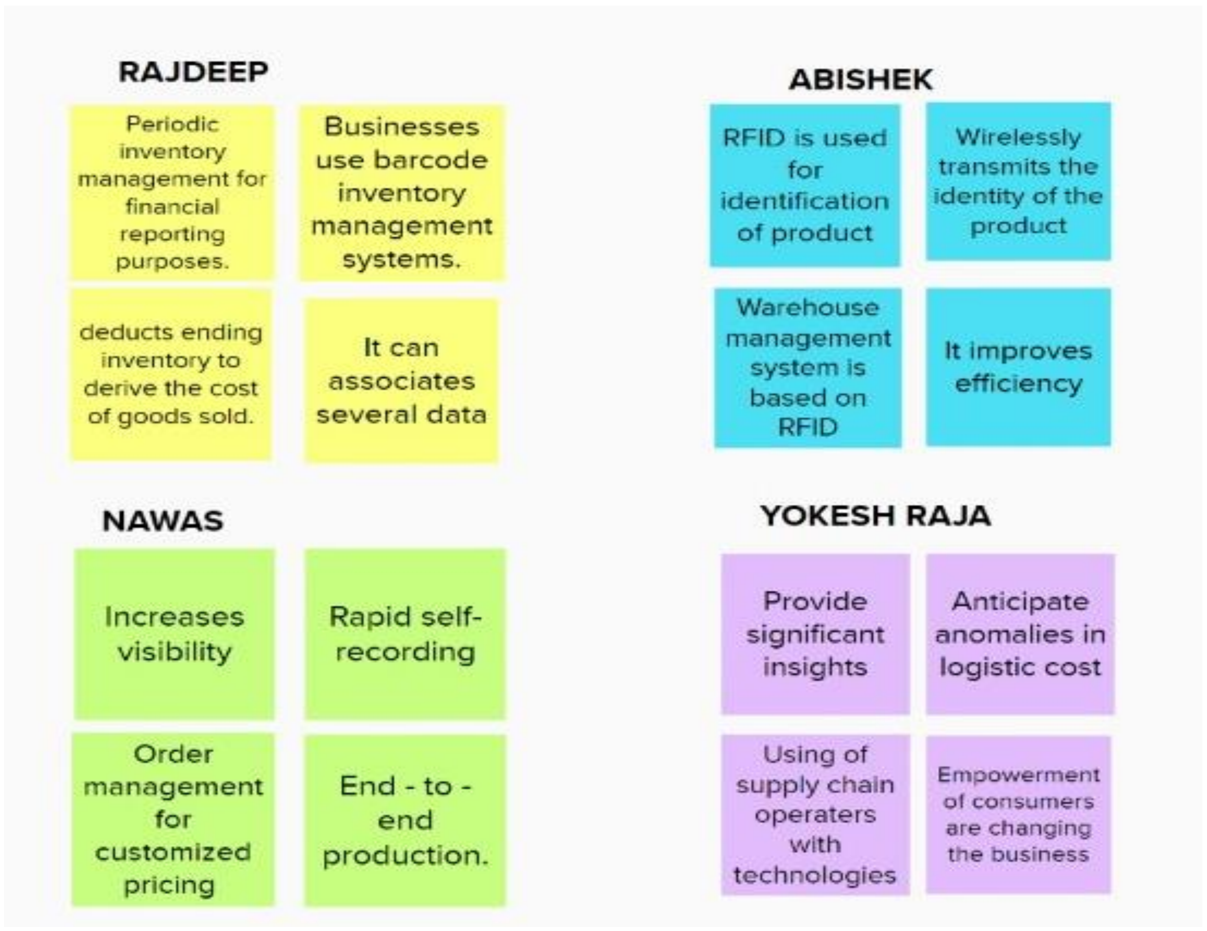
1 **Define your problem statement**
What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.
5 minutes

PROBLEM
The problem faced by the company is they do not have any systematic system to record and keep their inventory data. It is difficult for the admin to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.

Key rules of brainstorming
To run an smooth and productive session

Stay in topic.	Encourage wild ideas.
Defer judgment.	Listen to others.
Go for volume.	If possible, be visual.

Step-2: Brainstorm, Idea Listing and Grouping



RAJDEEP

Periodic inventory management for financial reporting purposes.

Businesses use barcode inventory management systems.

deducts ending inventory to derive the cost of goods sold.

It can associates several data

ABISHEK

RFID is used for identification of product

Wirelessly transmits the identity of the product

Warehouse management system is based on RFID

It improves efficiency

NAWAS

Increases visibility

Rapid self-recording

Order management for customized pricing

End - to - end production.

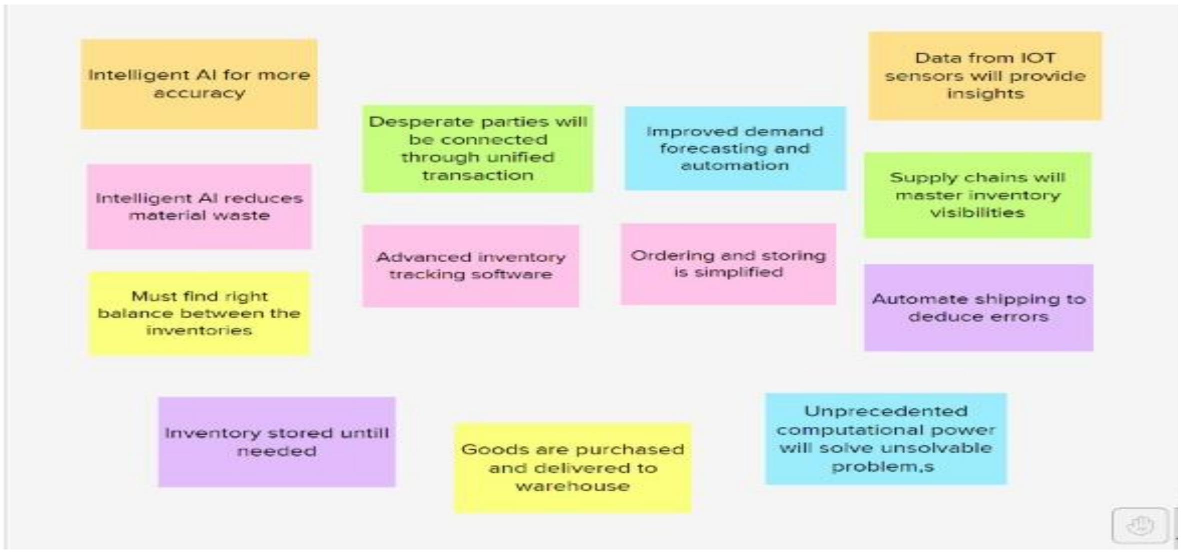
YOKESH RAJA

Provide significant insights

Anticipate anomalies in logistic cost

Using of supply chain operaters with technologies

Empowerment of consumers are changing the business



Intelligent AI for more accuracy

Desperate parties will be connected through unified transaction

Improved demand forecasting and automation

Data from IOT sensors will provide insights

Intelligent AI reduces material waste

Supply chains will master inventory visibilities

Advanced inventory tracking software

Ordering and storing is simplified

Must find right balance between the inventories

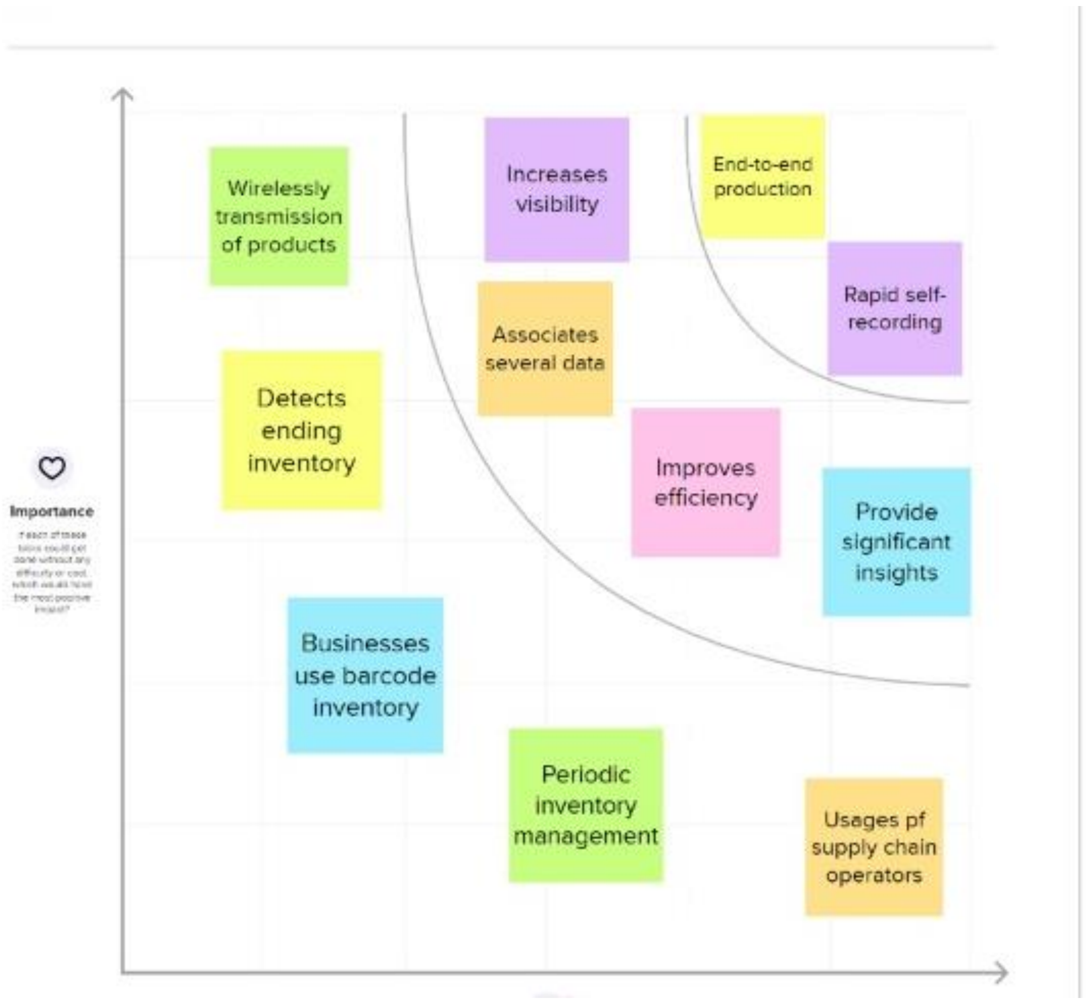
Automate shipping to deduce errors

Inventory stored untill needed

Goods are purchased and delivered to warehouse

Unprecedented computational power will solve unsolvable problem,s

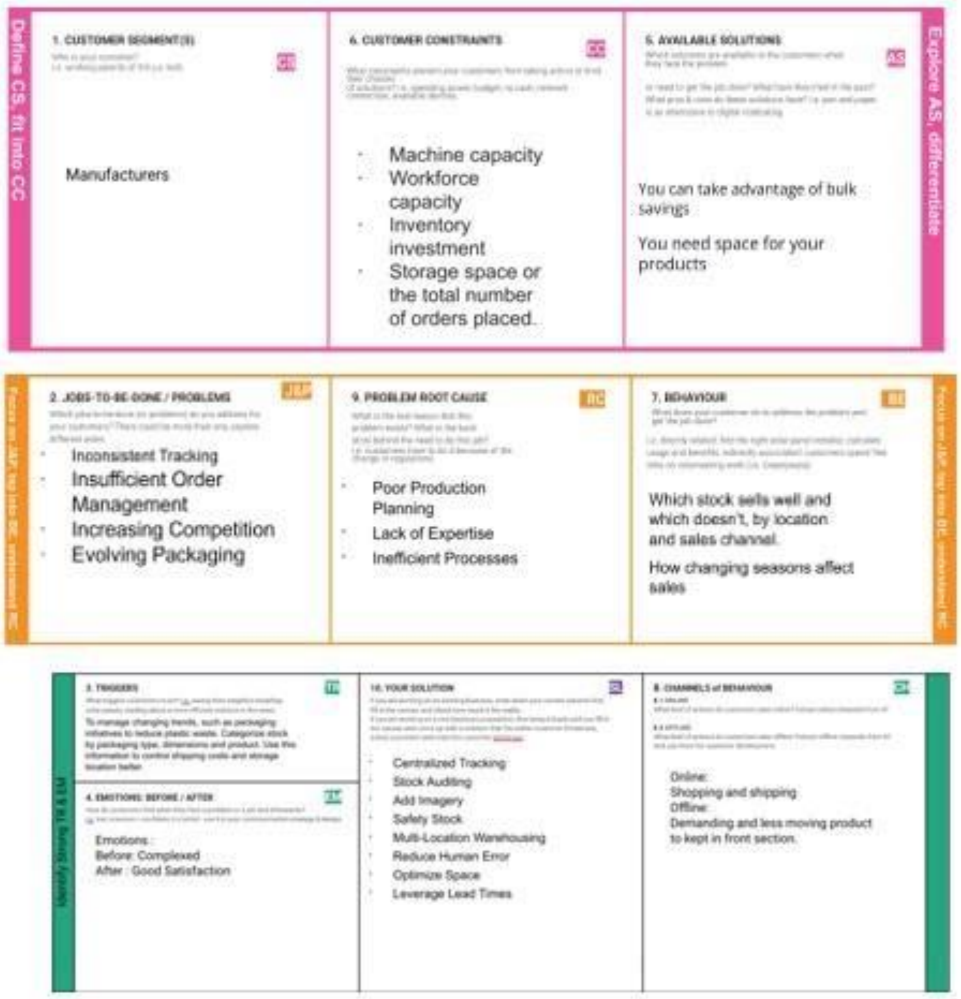
Step-3: Idea Prioritization


Wirelessly transmission of products
Increases visibility
End-to-end production
Rapid self-recording
Associates several data
Detects ending inventory
Importance
Improves efficiency
Provide significant insights
Businesses use barcode inventory
Periodic inventory management
Usages pf supply chain operators

3.3 Proposed Solution

The system customizes and only shows recommended jobs based on the user's skill set and preferences (Using graphql api)
Similarly, the same recommendation system helps provide job applicant recommendations to the job recruiters to find the most eligible candidates for their firm.
All important data - job seeker's and hoster's personal information needs to be also stored safely and securely. Using a sql database is the most easiest, safest and convienent way possible.
Data needs to also be private in some cases like when information is shared with the host while applying for a job.

3.4 Problem Solution fit



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Manufacturers

6. CUSTOMER CONSTRAINTS CC

- Machine capacity
- Workforce capacity
- Inventory investment
- Storage space or the total number of orders placed.

5. AVAILABLE SOLUTIONS AS

You can take advantage of bulk savings

You need space for your products

Explore AS, differentiate

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

2. JOBS-TO-BE-DONE / PROBLEMS J&P

- Inconsistent Tracking
- Insufficient Order Management
- Increasing Competition
- Evolving Packaging

9. PROBLEM ROOT CAUSE RC

- Poor Production Planning
- Lack of Expertise
- Inefficient Processes

7. BEHAVIOUR BE

Which stock sells well and which doesn't, by location and sales channel.

How changing seasons affect sales

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

Identify strong TR & EM

3. TRIGGERS TR

To manage changing trends, such as packaging initiatives to reduce plastic waste. Categorize stock by packaging type, dimensions and product. Use this information to control shipping costs and storage location better.

4. EMOTIONS: BEFORE / AFTER EM

Emotions :
Before: Complexed
After : Good Satisfaction

10. YOUR SOLUTION SL

- Centralized Tracking
- Stock Auditing
- Add Imagery
- Safety Stock
- Multi-Location Warehousing
- Reduce Human Error
- Optimize Space
- Leverage Lead Times

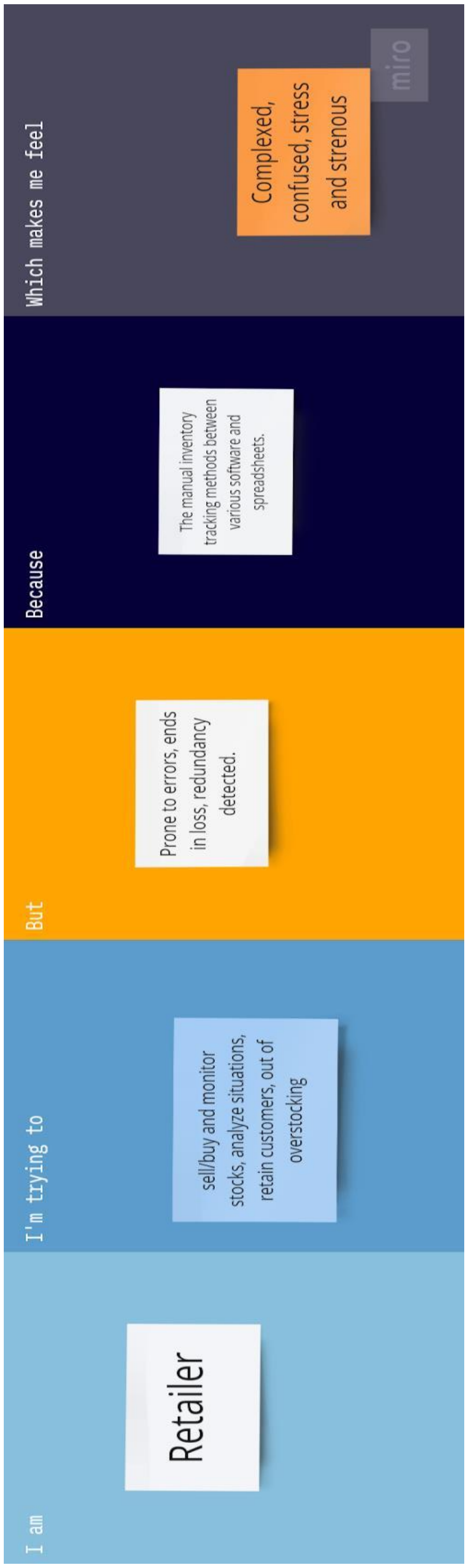
8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE

8.2 OFFLINE

Online:
Shopping and shipping
Offline:
Demanding and less moving product to kept in front section.

3.5 Customer Problem Statement



I am

Retailer

I'm trying to

sell/buy and monitor stocks, analyze situations, retain customers, out of overstocking

But

Prone to errors, ends in loss, redundancy detected.

Because

The manual inventory tracking methods between various software and spreadsheets.

Which makes me feel

Complexed, confused, stress and strenous

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

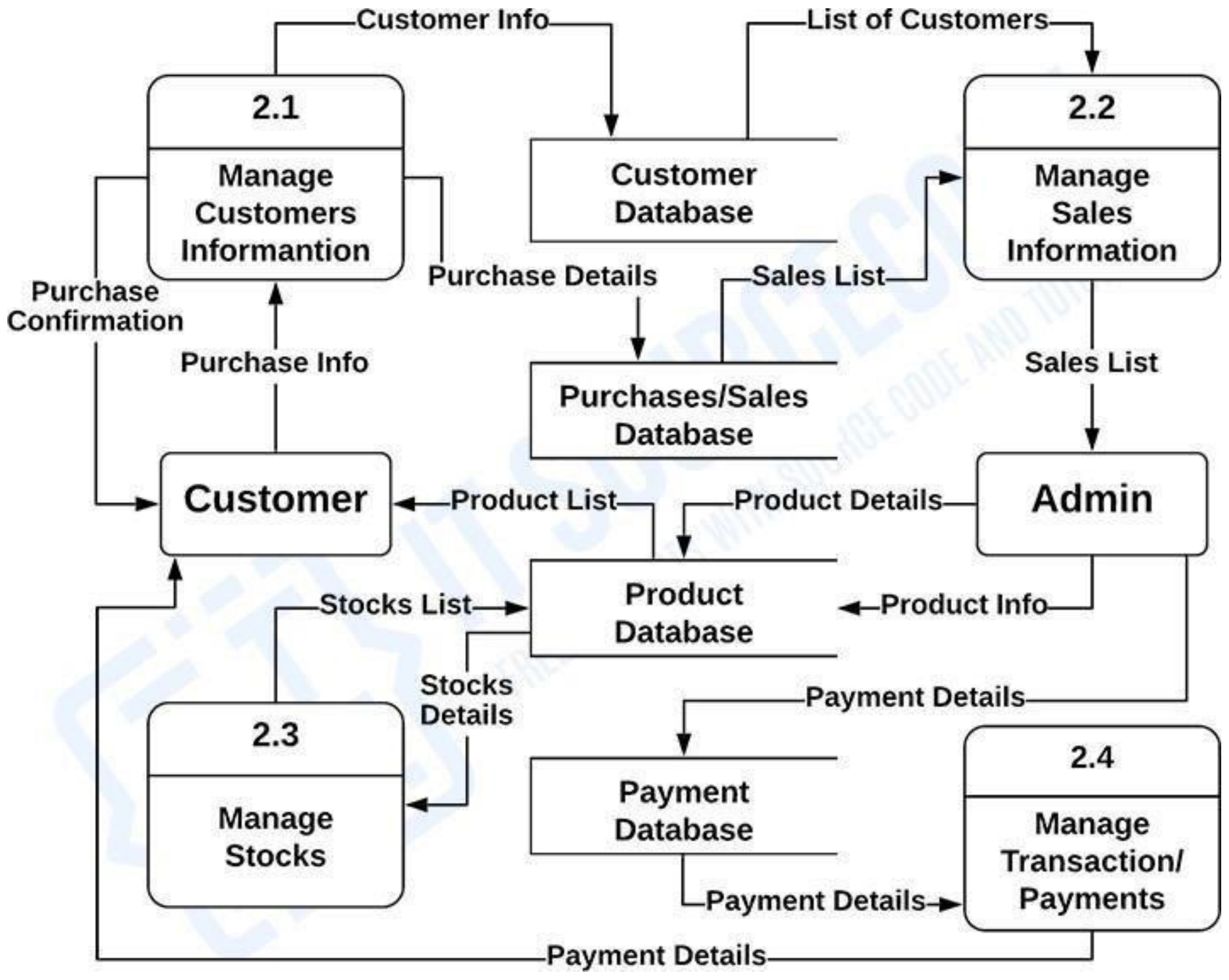
- The System aims at providing an efficient interface to the user for managing of inventory, it shall also provide the user varied options for managing the inventory through various functions at hand. The ingredient levels are continuously monitored based on their usage and are checked for the threshold levels in the inventory and accordingly the user is alerted about low levels of certain ingredients. The design is such that the user does not have to manually update the inventory every time, the System does if for the user.
- The System calculates and predicts the amount of usage for specific set days that are pre-set by the user(admin) , it also alerts the user of an impending action to order ingredients before the specific day set by the user. Therefore the user never has to worry about manually calculating the estimated usage of the ingredients as the System does it for the user.
- The simple interface of the System has functions like adding a recipe, removing or updating the recipe. It also extends to functions such as adding a vendor for an ingredient,, removing the vendor, checking threshold levels, processing orders, altering processed orders etc.

4.2 Non-Functional requirements

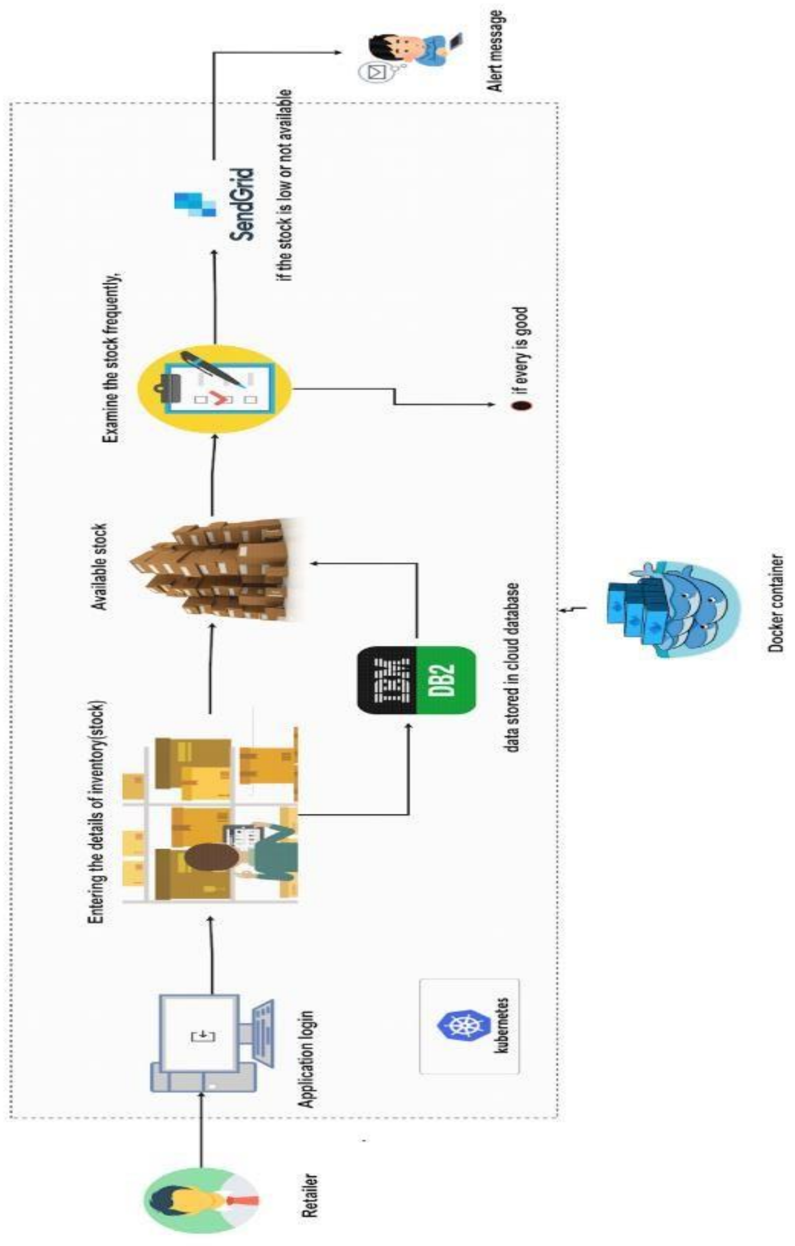
- The system must not lag, because the workers using it don’t have down-time to wait for it to complete an action.
- The system must complete updating the databases, adding of recipe, ingredient, vendor and occasions successfully every time the user requests such a process.
- All the functions of the system must be available to the user every time the system is turned on.
- The calculations performed by the system must comply according to the norms set by the user and should not vary unless explicitly changed by the user
- The System must give accurate inventory status to the user continuously. Any inaccuracies are taken care by the regular confirming of the actual levels with the levels displayed in the system.
- The System must successfully add any recipe, ingredients, vendors or special occasions given by the user and provide estimations and inventory status in relevance with the newly updated entities.

5. PROJECT DESIGN

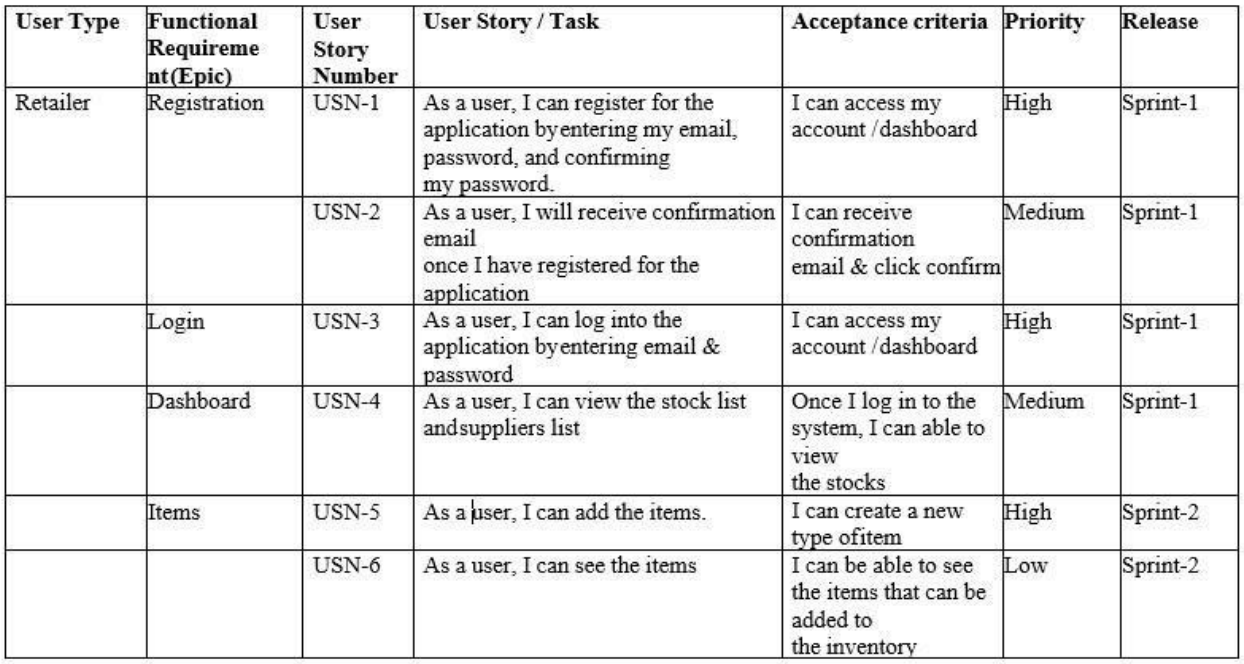
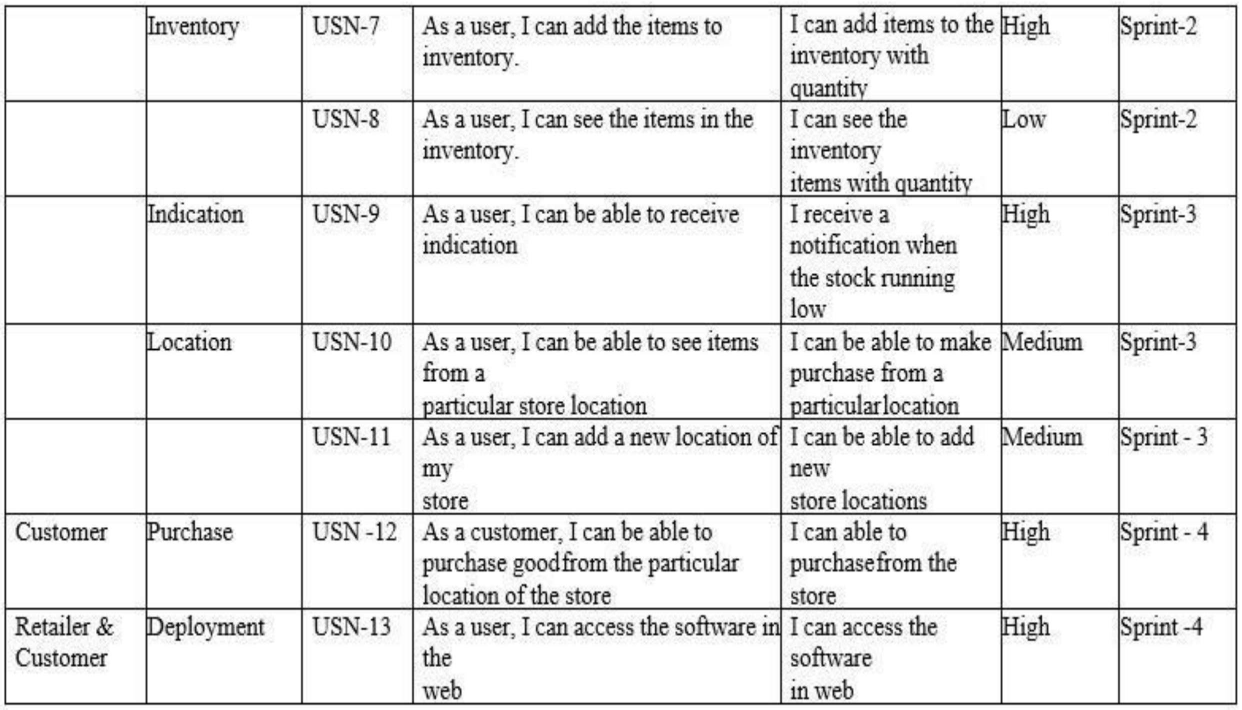
5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture


Retailer
Application login
Entering the details of inventory(stock)
Available stock
Examine the stock frequently.
SendGrid
if the stock is low or not available
Alert message
if every is good
IBM DB2
data stored in cloud database
kubernetes
Docker container

5.3 User Stories

User Type	Functional Requirement(Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Retailer	Registration	USN-1	As a user, I can register for the application byentering my email, password, and confirming my password.	I can access my account /dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	Medium	Sprint-1
	Login	USN-3	As a user, I can log into the application byentering email & password	I can access my account /dashboard	High	Sprint-1
	Dashboard	USN-4	As a user, I can view the stock list andsuppliers list	Once I log in to the system, I can able to view the stocks	Medium	Sprint-1
	Items	USN-5	As a user, I can add the items.	I can create a new type ofitem	High	Sprint-2
		USN-6	As a user, I can see the items	I can be able to see the items that can be added to the inventory	Low	Sprint-2
	Inventory	USN-7	As a user, I can add the items to inventory.	I can add items to the inventory with quantity	High	Sprint-2
		USN-8	As a user, I can see the items in the inventory.	I can see the inventory items with quantity	Low	Sprint-2
	Indication	USN-9	As a user, I can be able to receive indication	I receive a notification when the stock running low	High	Sprint-3
	Location	USN-10	As a user, I can be able to see items from a particular store location	I can be able to make purchase from a particularlocation	Medium	Sprint-3
		USN-11	As a user, I can add a new location of my store	I can be able to add new store locations	Medium	Sprint - 3
Customer	Purchase	USN -12	As a customer, I can be able to purchase goodfrom the particular location of the store	I can able to purchasefrom the store	High	Sprint - 4
Retailer & Customer	Deployment	USN-13	As a user, I can access the software in the web	I can access the software in web	High	Sprint -4

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Prio
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High
Sprint-1		USN-2	As a user, I can register for the application through E-mail	1	Med
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	2	Med

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Prio
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available	4	High
Sprint-2	Add items to cart	USN-6	As a user, I can add the products I wish to buy to the carts.	5	Med
Sprint-3	Stock Update	USN-7	As a user, I can add products which are not available in the dashboard to the stock list.	5	Med
Sprint-4	Request to Customer Care	USN-8	As a user, I can contact the Customer Care Executive and request any services I want from the customer care.	5	Low
Sprint-4	Contact Administrator	USN-9	I can be able to report any difficulties I experience as a report	5	Med

6.2 Milestone And Activities

Setting up the application environment	M-01	Setting up the needed resources in the local machine
Integrating send grid service	M-02	To send emails from the application, we need to integrate the SendGrid Service.
Deployment of the app in IBM Cloud	M-03	Containerize a Flask application by using Docker and deploy it to the IBM Cloud Kubernetes Service
Implementing Web Application	M-04	To create a website and to UI, to interact with the application.
Ideation Phase	M-05	Collecting information by referring to previous research on a topic and Preparing Literature Survey on the selected Project and Information Gathering, empathy map, and ideation
Project Design Phase - I	M-06	Prepare the proposed solution, the problem-solution fit, and the Solution Architecture.
Project Design Phase - II	M-07	Create a customer journey, functional requirements, a data flow diagram, and a technology architecture
Project Planning Phase	M-08	Make a list of milestones, an activity list, and a sprint delivery plan.

ACTIVITY LIST

Activity Number	Activity	Sub Activity	Assigned To	Status
1.	Setting up Application Environment	● Create Flask Project ● Create IBM Cloud Account ● Install IBM Cloud CLI ● Docker CLI Installation ● Create An Account In Sendgrid	All Members	In Progress
2.	Implementing Web Application	● Create UI To Interact With Application	All Members	In Progress
3.	Integrating SendGrid Service	● SendGrid Integration With Python Code	All Members	Not completed
4.	Deployment of App In IBM Cloud	● Containerize The App ● Upload Image To IBM Container Registry ● Deploy in Kubernetes	All Members	Not completed
5.	Ideation Phase	● Literature Survey On The Selected Project & Information Gathering ● Prepare Empathy Map ● Ideation	All Members	Completed
6.	Project Design Phase – I	● Proposed Solution ● Problem Solution Fit ● Solution Architecture	All Members	Completed

7.	Project Design Phase – II	● Customer Journey – day3 ● Functional Requirement ● Data Flow Diagrams ● Technology Architecture	All Members	Completed
8.	Project Planning Phase	● Prepare Milestone & Activity List ● Sprint Delivery Plan	All Members	In Progress
9.	Project Development Phase	● Delivery Of Sprint-1 ● Delivery Of Sprint-2 ● Delivery Of Sprint-3 ● Delivery Of Sprint-4	All Members	In Progress

7. CODING&SOLUTIONING

(Explain the features added in the project along with code)

7.1 Feature 1

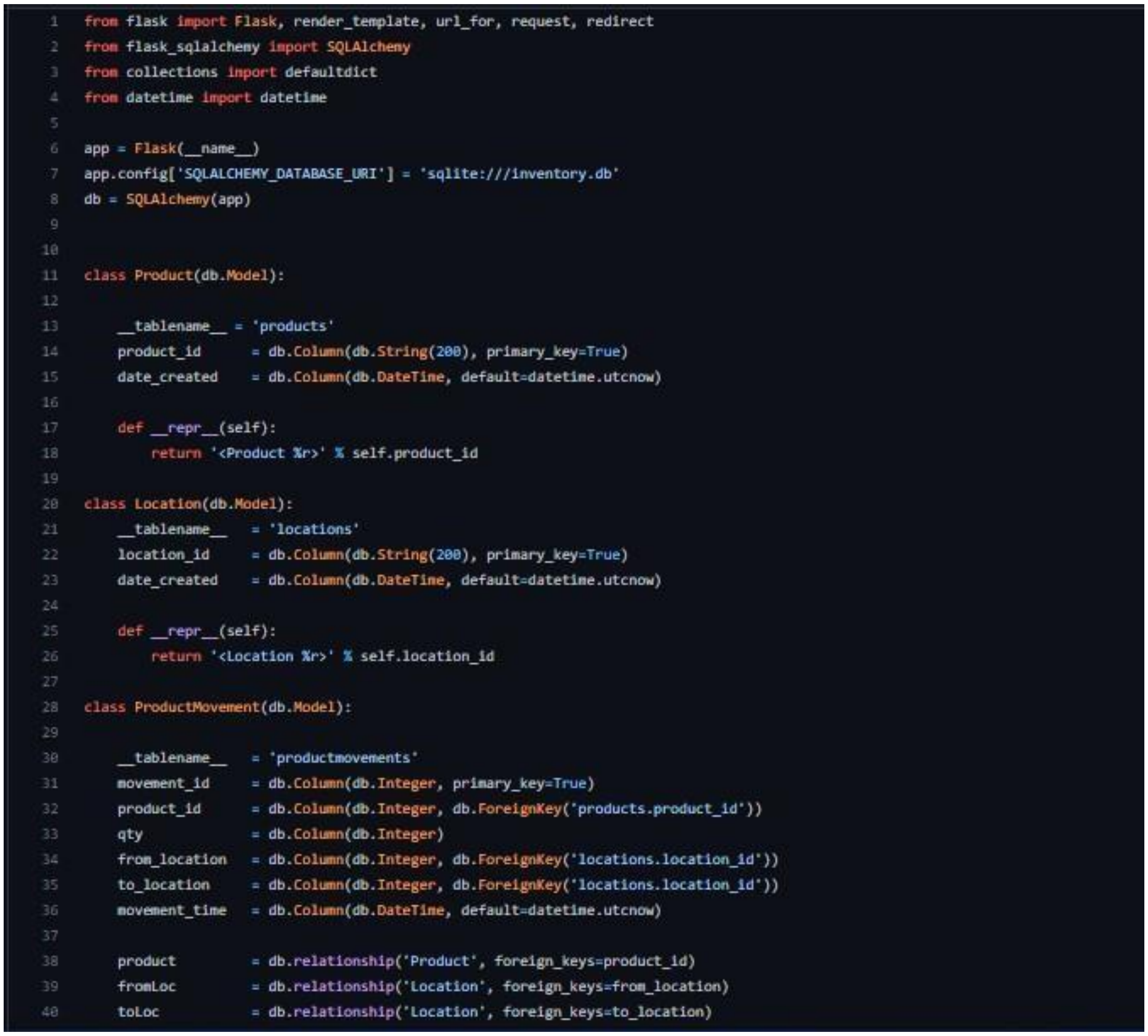
Complete insights into key products and service drivers. With the help of tables and symbols, marketers can effectively track and analyse factors that have an effect on important bottom lines like profitability. Store Managers can also effectively optimise product mix across channels, lines and brands with the product scorecards available. Some of the different KPIs that managers can avail of from product performance metrics are product sales by region, change in sales and margin per product, ROI per product, top competitor by product category and much more..

7.2 Feature 2

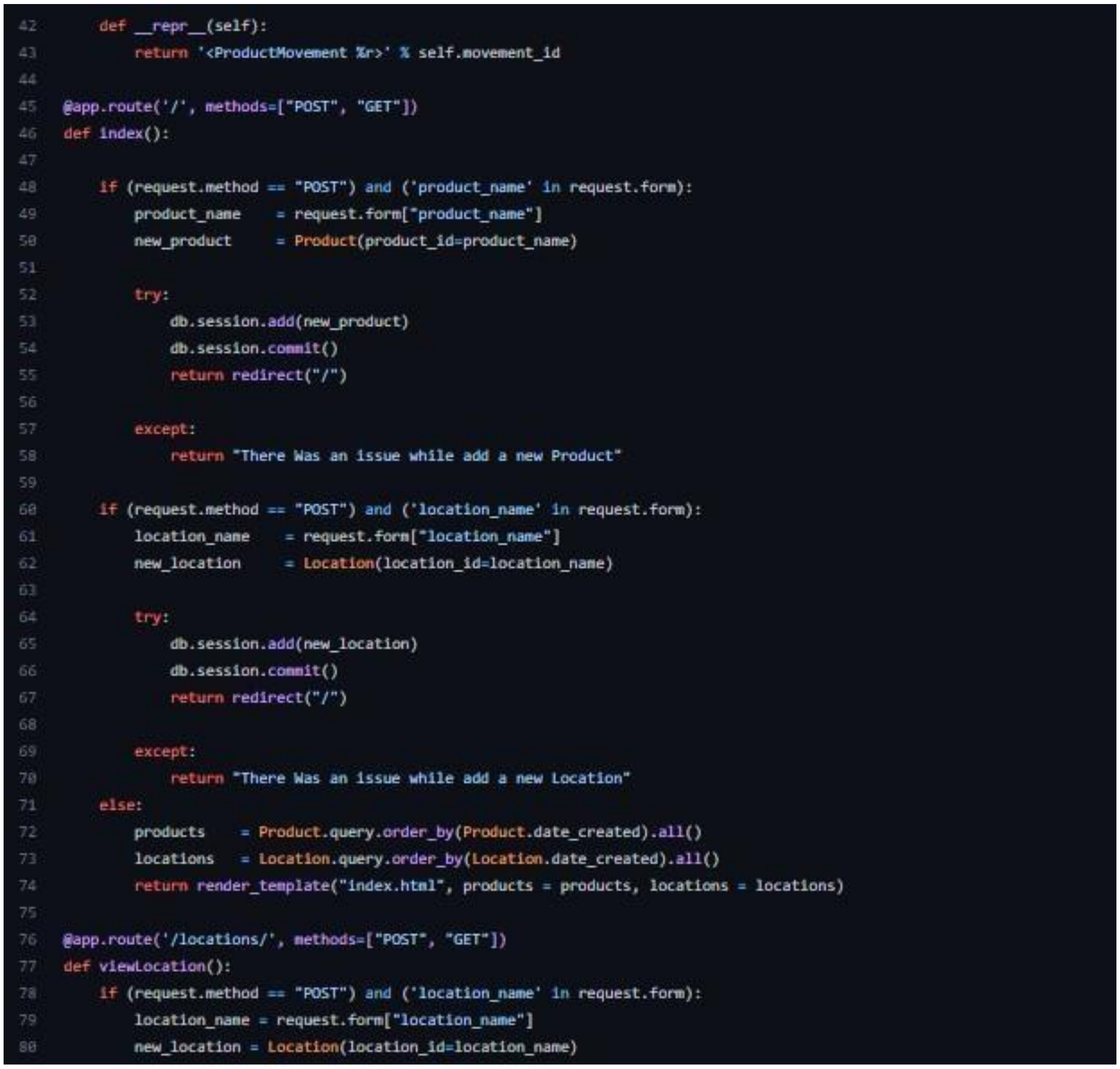
The entire organisation can access the same store data simultaneously and thus everyone has an understanding of what the customer wants. Managers can better monitor progress, respond immediately to customer needs, adjust parameters for continuous improvement, and exercise greater control over the organisation.

One can record and analyze inventory results and merchandise processes daily to know whether business decisions are based on timely, accurate information.

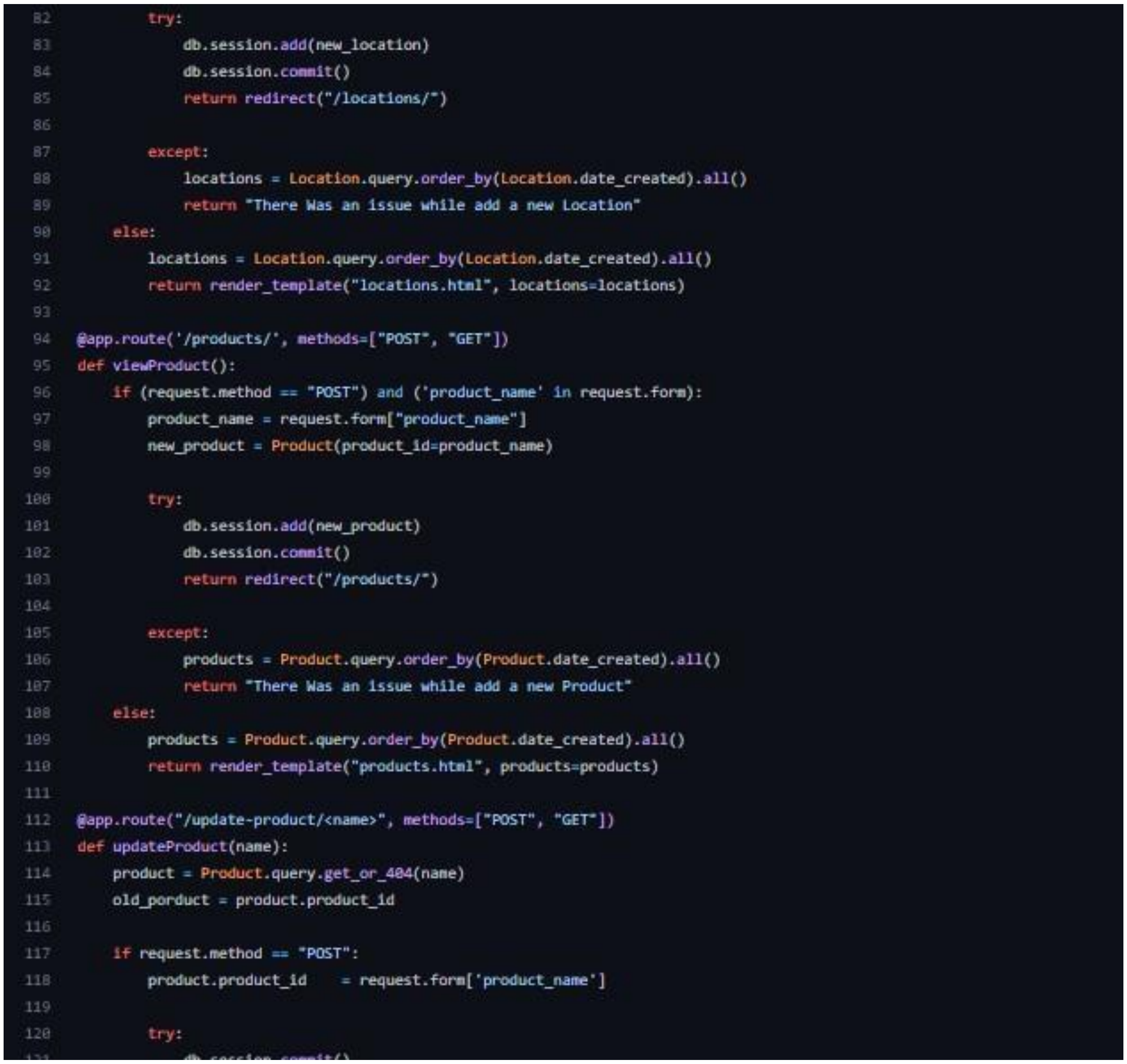
7.3 Code



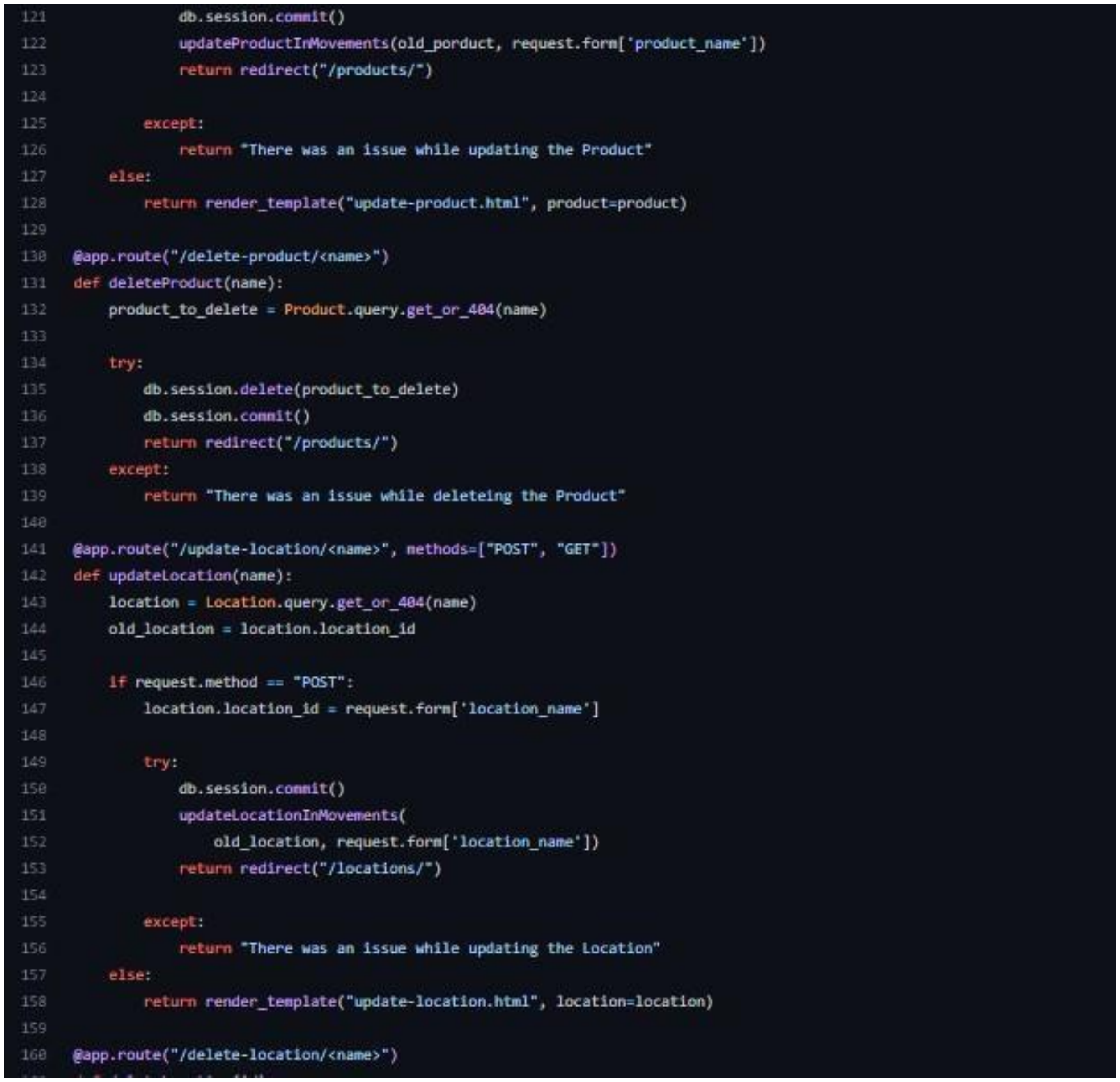
```python
from flask import Flask, render_template, url_for, request, redirect
from flask_sqlalchemy import SQLAlchemy
from collections import defaultdict
from datetime import datetime

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///inventory.db'
db = SQLAlchemy(app)


class Product(db.Model):

    __tablename__ = 'products'
    product_id      = db.Column(db.String(200), primary_key=True)
    date_created    = db.Column(db.DateTime, default=datetime.utcnow)

    def __repr__(self):
        return '<Product %r>' % self.product_id

class Location(db.Model):
    __tablename__   = 'locations'
    location_id     = db.Column(db.String(200), primary_key=True)
    date_created    = db.Column(db.DateTime, default=datetime.utcnow)

    def __repr__(self):
        return '<Location %r>' % self.location_id

class ProductMovement(db.Model):

    __tablename__   = 'productmovements'
    movement_id     = db.Column(db.Integer, primary_key=True)
    product_id      = db.Column(db.Integer, db.ForeignKey('products.product_id'))
    qty             = db.Column(db.Integer)
    from_location   = db.Column(db.Integer, db.ForeignKey('locations.location_id'))
    to_location     = db.Column(db.Integer, db.ForeignKey('locations.location_id'))
    movement_time   = db.Column(db.DateTime, default=datetime.utcnow)

    product         = db.relationship('Product', foreign_keys=product_id)
    fromLoc         = db.relationship('Location', foreign_keys=from_location)
    toLoc           = db.relationship('Location', foreign_keys=to_location)
```



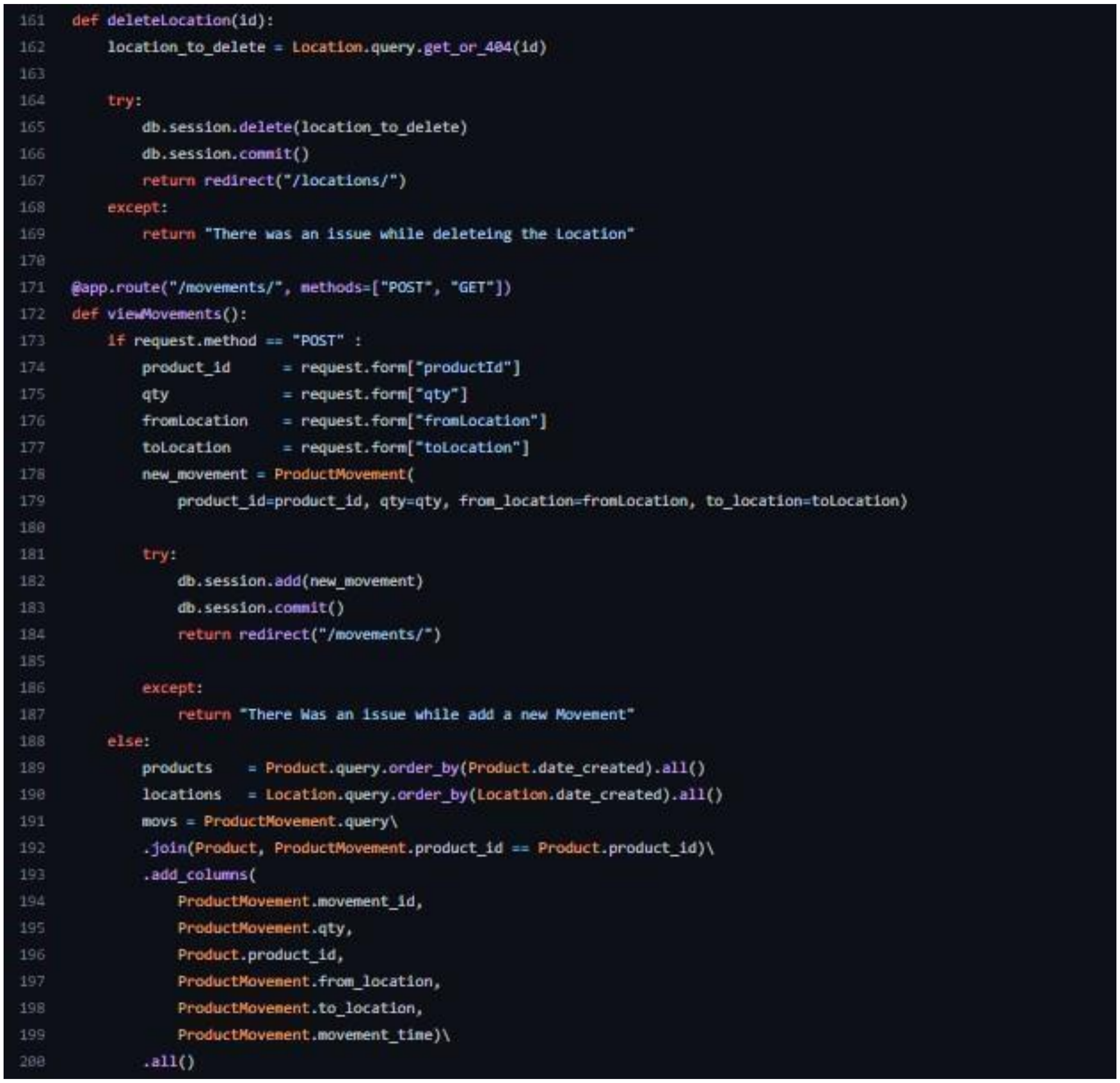
```
    def __repr__(self):
        return '<ProductMovement %r>' % self.movement_id

@app.route('/', methods=["POST", "GET"])
def index():

    if (request.method == "POST") and ('product_name' in request.form):
        product_name    = request.form["product_name"]
        new_product     = Product(product_id=product_name)

        try:
            db.session.add(new_product)
            db.session.commit()
            return redirect("/")

        except:
            return "There Was an issue while add a new Product"

    if (request.method == "POST") and ('location_name' in request.form):
        location_name    = request.form["location_name"]
        new_location     = Location(location_id=location_name)

        try:
            db.session.add(new_location)
            db.session.commit()
            return redirect("/")

        except:
            return "There Was an issue while add a new Location"
    else:
        products    = Product.query.order_by(Product.date_created).all()
        locations   = Location.query.order_by(Location.date_created).all()
        return render_template("index.html", products = products, locations = locations)

@app.route('/locations/', methods=["POST", "GET"])
def viewLocation():
    if (request.method == "POST") and ('location_name' in request.form):
        location_name = request.form["location_name"]
        new_location = Location(location_id=location_name)
```



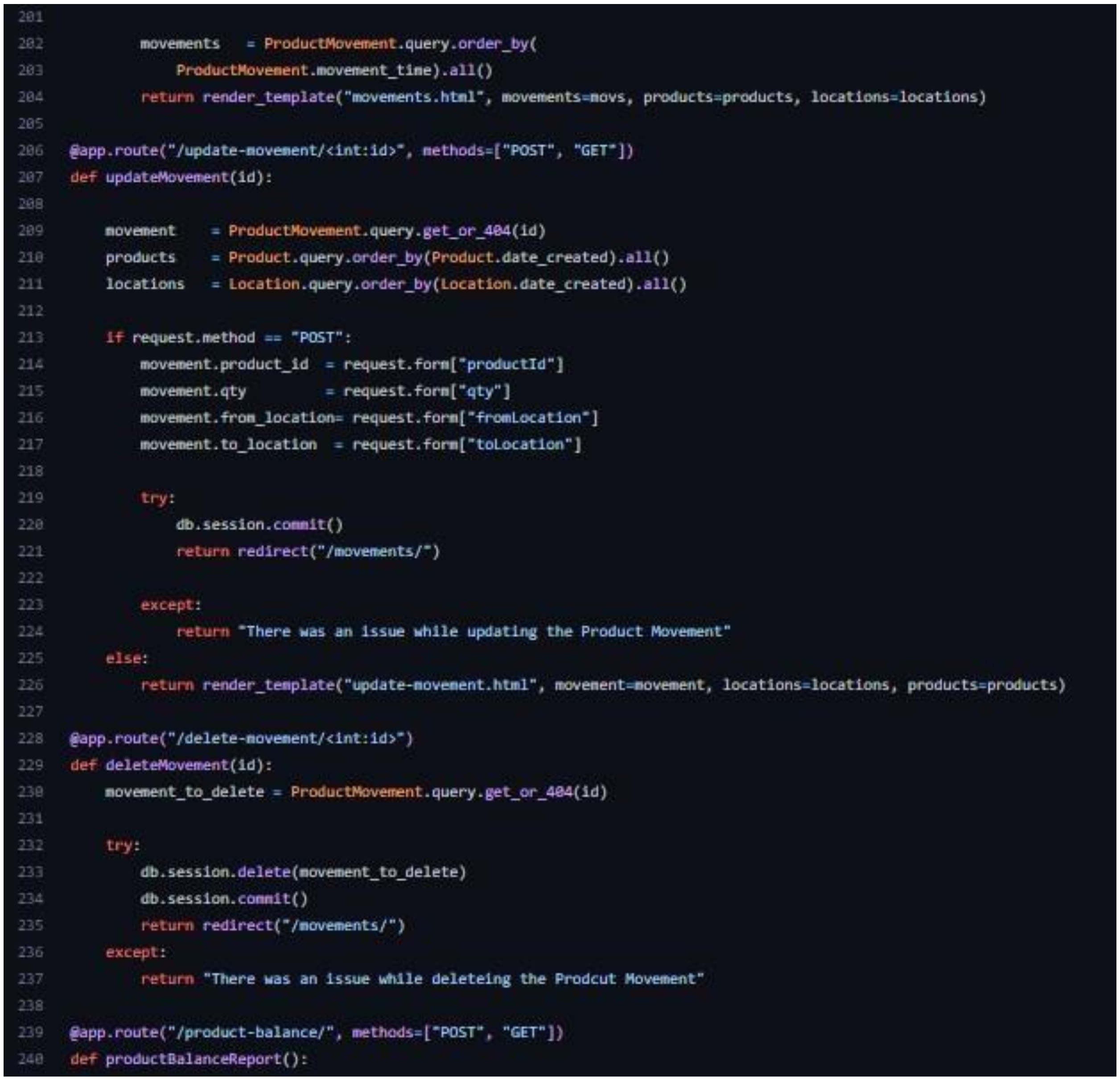
```
        try:
            db.session.add(new_location)
            db.session.commit()
            return redirect("/locations/")

        except:
            locations = Location.query.order_by(Location.date_created).all()
            return "There Was an issue while add a new Location"
    else:
        locations = Location.query.order_by(Location.date_created).all()
        return render_template("locations.html", locations=locations)

@app.route('/products/', methods=["POST", "GET"])
def viewProduct():
    if (request.method == "POST") and ('product_name' in request.form):
        product_name = request.form["product_name"]
        new_product = Product(product_id=product_name)

        try:
            db.session.add(new_product)
            db.session.commit()
            return redirect("/products/")

        except:
            products = Product.query.order_by(Product.date_created).all()
            return "There Was an issue while add a new Product"
    else:
        products = Product.query.order_by(Product.date_created).all()
        return render_template("products.html", products=products)

@app.route("/update-product/<name>", methods=["POST", "GET"])
def updateProduct(name):
    product = Product.query.get_or_404(name)
    old_porduct = product.product_id

    if request.method == "POST":
        product.product_id    = request.form['product_name']

        try:
```



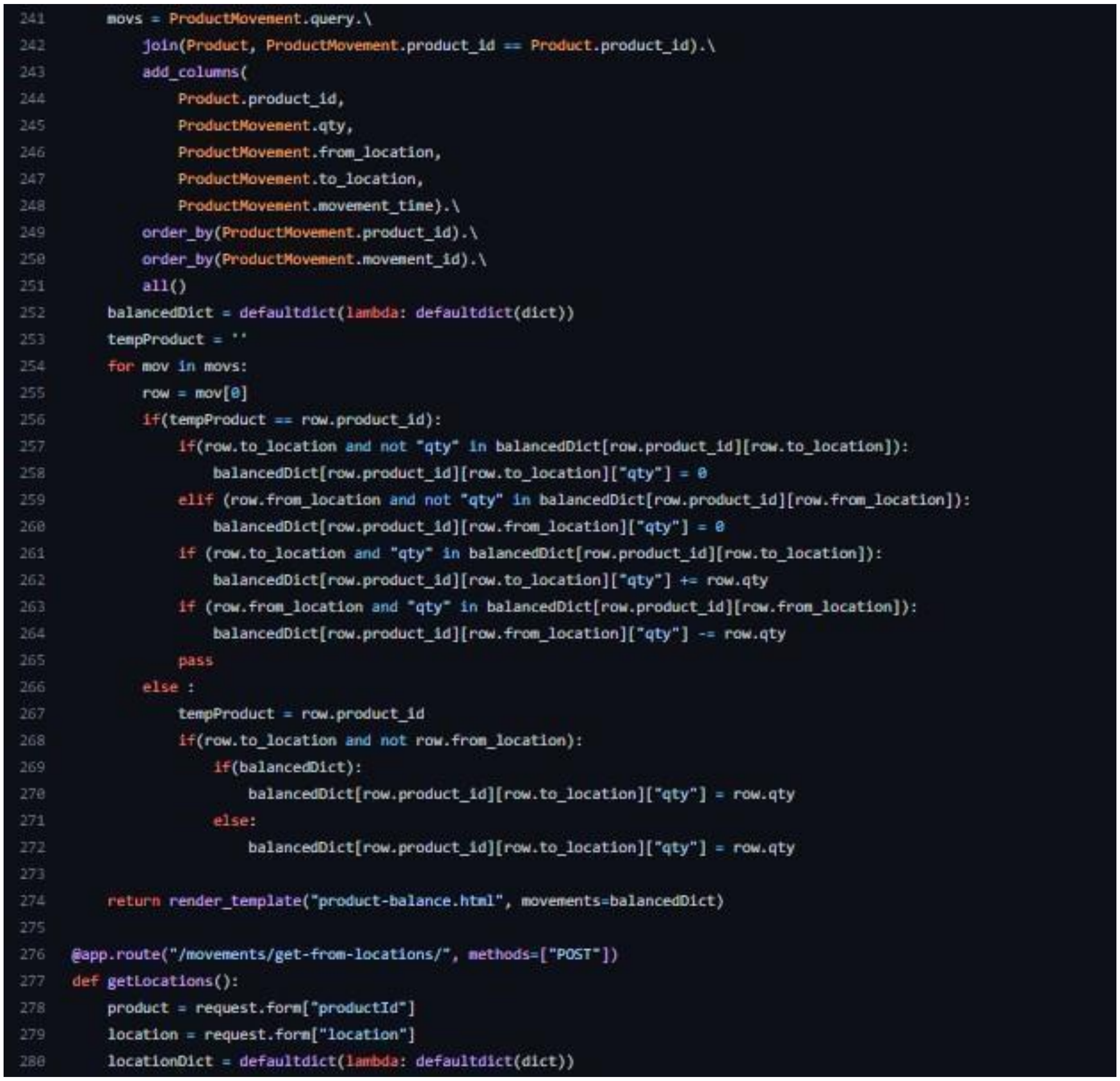
```
            db.session.commit()
            updateProductInMovements(old_porduct, request.form['product_name'])
            return redirect("/products/")

        except:
            return "There was an issue while updating the Product"
    else:
        return render_template("update-product.html", product=product)

@app.route("/delete-product/<name>")
def deleteProduct(name):
    product_to_delete = Product.query.get_or_404(name)

    try:
        db.session.delete(product_to_delete)
        db.session.commit()
        return redirect("/products/")
    except:
        return "There was an issue while deleteing the Product"

@app.route("/update-location/<name>", methods=["POST", "GET"])
def updateLocation(name):
    location = Location.query.get_or_404(name)
    old_location = location.location_id

    if request.method == "POST":
        location.location_id = request.form['location_name']

        try:
            db.session.commit()
            updateLocationInMovements(
                old_location, request.form['location_name'])
            return redirect("/locations/")

        except:
            return "There was an issue while updating the Location"
    else:
        return render_template("update-location.html", location=location)

@app.route("/delete-location/<name>")
```



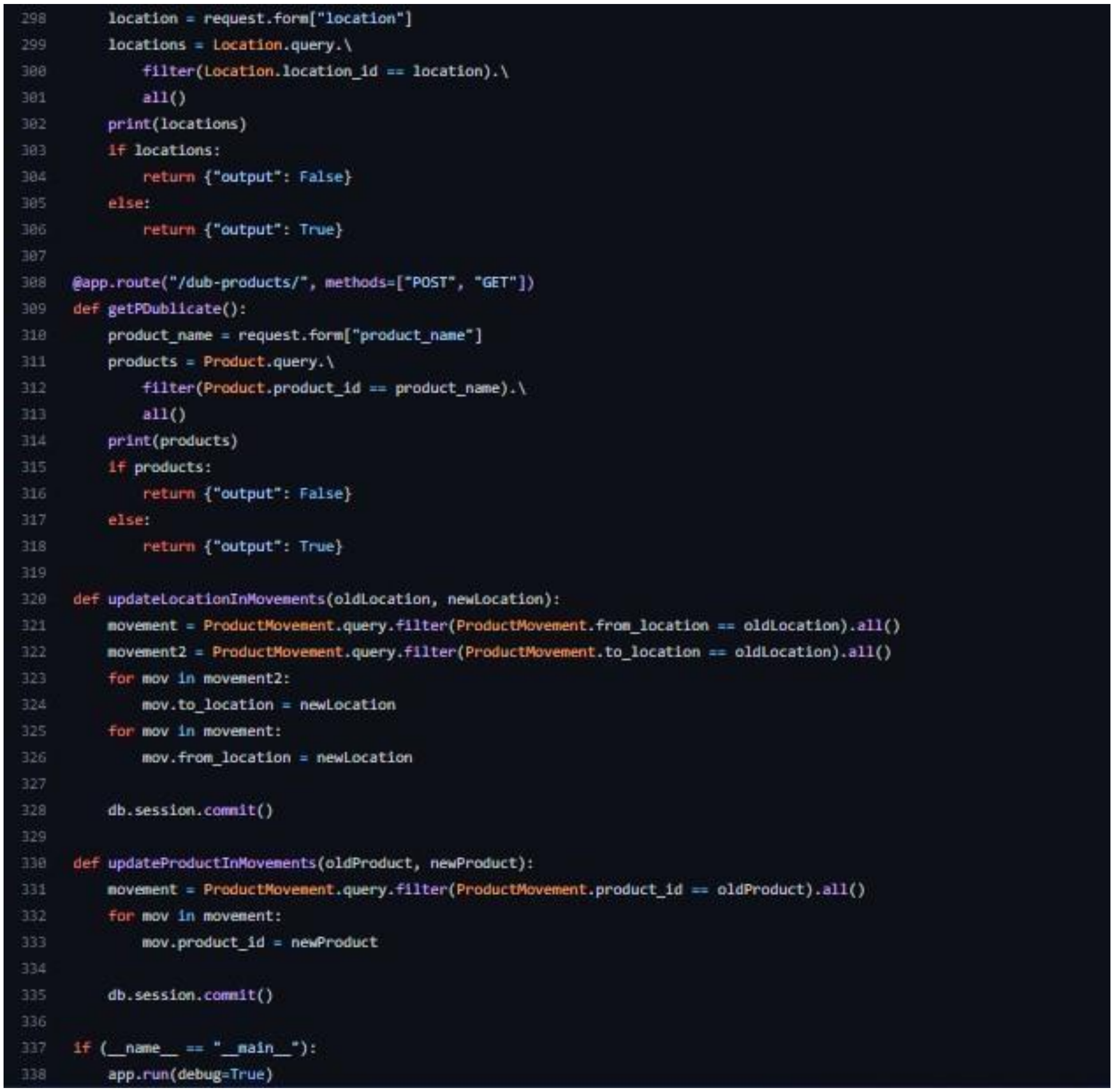
```python
def deleteLocation(id):
    location_to_delete = Location.query.get_or_404(id)

    try:
        db.session.delete(location_to_delete)
        db.session.commit()
        return redirect("/locations/")
    except:
        return "There was an issue while deleteing the Location"

@app.route("/movements/", methods=["POST", "GET"])
def viewMovements():
    if request.method == "POST" :
        product_id      = request.form["productId"]
        qty             = request.form["qty"]
        fromLocation    = request.form["fromLocation"]
        toLocation      = request.form["toLocation"]
        new_movement = ProductMovement(
            product_id=product_id, qty=qty, from_location=fromLocation, to_location=toLocation)

        try:
            db.session.add(new_movement)
            db.session.commit()
            return redirect("/movements/")

        except:
            return "There Was an issue while add a new Movement"
    else:
        products    = Product.query.order_by(Product.date_created).all()
        locations   = Location.query.order_by(Location.date_created).all()
        movs = ProductMovement.query\
        .join(Product, ProductMovement.product_id == Product.product_id)\
        .add_columns(
            ProductMovement.movement_id,
            ProductMovement.qty,
            Product.product_id,
            ProductMovement.from_location,
            ProductMovement.to_location,
            ProductMovement.movement_time)\
        .all()
```



```python

        movements   = ProductMovement.query.order_by(
            ProductMovement.movement_time).all()
        return render_template("movements.html", movements=movs, products=products, locations=locations)

@app.route("/update-movement/<int:id>", methods=["POST", "GET"])
def updateMovement(id):

    movement    = ProductMovement.query.get_or_404(id)
    products    = Product.query.order_by(Product.date_created).all()
    locations   = Location.query.order_by(Location.date_created).all()

    if request.method == "POST":
        movement.product_id  = request.form["productId"]
        movement.qty         = request.form["qty"]
        movement.from_location= request.form["fromLocation"]
        movement.to_location  = request.form["toLocation"]

        try:
            db.session.commit()
            return redirect("/movements/")

        except:
            return "There was an issue while updating the Product Movement"
    else:
        return render_template("update-movement.html", movement=movement, locations=locations, products=products)

@app.route("/delete-movement/<int:id>")
def deleteMovement(id):
    movement_to_delete = ProductMovement.query.get_or_404(id)

    try:
        db.session.delete(movement_to_delete)
        db.session.commit()
        return redirect("/movements/")
    except:
        return "There was an issue while deleteing the Prodcut Movement"

@app.route("/product-balance/", methods=["POST", "GET"])
def productBalanceReport():
```



```
    movs = ProductMovement.query.\
        join(Product, ProductMovement.product_id == Product.product_id).\
        add_columns(
            Product.product_id,
            ProductMovement.qty,
            ProductMovement.from_location,
            ProductMovement.to_location,
            ProductMovement.movement_time).\
        order_by(ProductMovement.product_id).\
        order_by(ProductMovement.movement_id).\
        all()
    balancedDict = defaultdict(lambda: defaultdict(dict))
    tempProduct = ''
    for mov in movs:
        row = mov[0]
        if(tempProduct == row.product_id):
            if(row.to_location and not "qty" in balancedDict[row.product_id][row.to_location]):
                balancedDict[row.product_id][row.to_location]["qty"] = 0
            elif (row.from_location and not "qty" in balancedDict[row.product_id][row.from_location]):
                balancedDict[row.product_id][row.from_location]["qty"] = 0
            if (row.to_location and "qty" in balancedDict[row.product_id][row.to_location]):
                balancedDict[row.product_id][row.to_location]["qty"] += row.qty
            if (row.from_location and "qty" in balancedDict[row.product_id][row.from_location]):
                balancedDict[row.product_id][row.from_location]["qty"] -= row.qty
            pass
        else :
            tempProduct = row.product_id
            if(row.to_location and not row.from_location):
                if(balancedDict):
                    balancedDict[row.product_id][row.to_location]["qty"] = row.qty
                else:
                    balancedDict[row.product_id][row.to_location]["qty"] = row.qty

    return render_template("product-balance.html", movements=balancedDict)

@app.route("/movements/get-from-locations/", methods=["POST"])
def getLocations():
    product = request.form["productId"]
    location = request.form["location"]
    locationDict = defaultdict(lambda: defaultdict(dict))
```



```python
    location = request.form["location"]
    locations = Location.query.\
        filter(Location.location_id == location).\
        all()
    print(locations)
    if locations:
        return {"output": False}
    else:
        return {"output": True}

@app.route("/dub-products/", methods=["POST", "GET"])
def getPDublicate():
    product_name = request.form["product_name"]
    products = Product.query.\
        filter(Product.product_id == product_name).\
        all()
    print(products)
    if products:
        return {"output": False}
    else:
        return {"output": True}

def updateLocationInMovements(oldLocation, newLocation):
    movement = ProductMovement.query.filter(ProductMovement.from_location == oldLocation).all()
    movement2 = ProductMovement.query.filter(ProductMovement.to_location == oldLocation).all()
    for mov in movement2:
        mov.to_location = newLocation
    for mov in movement:
        mov.from_location = newLocation

    db.session.commit()

def updateProductInMovements(oldProduct, newProduct):
    movement = ProductMovement.query.filter(ProductMovement.product_id == oldProduct).all()
    for mov in movement:
        mov.product_id = newProduct

    db.session.commit()

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

8. RESULTS

8.1 Performance Metrics

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. Many Lean practitioners claim that inventory performance is the single best indicator of the overall operational performance of a facility.

9. ADVANTAGES & DISADVANTAGES

- Paper-based retail inventory management can take a lot of time and effort. The retail inventory management software can cut short your in-store inventory process cycles through automation. Automation would give you time to focus on other productive business tasks.
- Inventory management is one of the crucial retail processes. Thus, any discrepancy in the inventory control would impact all other operations in your company. The retail inventory software can streamline the inventory processes, which would, in turn, improve the efficiency of your entire business
- Manual inventory control would increase your labor and process costs. The software would not only help you save time, but it would also help you reduce costs. As a result, the profitability of your business would improve. Also, you can invest the excess funds in activities that promote your business growth.
- One of the biggest problems with any computerized system is the potential for a system crash. A corrupt hard drive, power outages and other technical issues can result in the loss of needed data. At the least, businesses are interrupted when they are unable to access data they need. Business owners should back up data regularly to protect against data loss.
- Hackers look for any way to get company or consumer information. An inventory system connected to point-of-sale devices and accounting is a valuable resource to hack into in search of potential financial information or personal details of owners, vendors or clients. Updating firewalls and anti-virus software can mitigate this potential issue.
- When everything is automated, it is easy to forego time-consuming physical inventory audits. They may no longer seem necessary when the computers are doing their work. However, it is important to continue to do regular audits to identify loss such as spoilage or breakage. Audits also help business owners identify potential internal theft and manipulation of the computerized inventory system.

10. 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.

11. FUTURE SCOPE

In summary, successful companies will embrace the challenges of inventory management in the 21st century by levering the technology that is being offered through the Fourth Industrial Revolution. More important, companies will look at inventory as a strategic asset, that when properly deployed will deliver increased value and competitive advantage. Effective collaboration between supply chain partners will take on increased importance. The intensifying risks inherent with global sourcing in combination with a better appreciation of TCO will motivate companies to rethink their global inventory strategies.

12. REFERENCES

- Aggarwal, S.: A review of current inventory theory and its applications. International Journal of Production Research 12, 443–472 (1974)
- Anily, S., Federgruen, A.: One warehouse multiple retailer systems with vehicle routing costs. Management Science 36, 92–114 (1990)
- Beckmann, M.: An inventory model for arbitrary interval and quantity distributions of demand. Management Science 8, 35–57 (1961)
- Hamann, T., Proth, J.: Inventory control of repairable tools with incomplete information. International Journal of Production Economics 31, 543–550 (1993)

GitHub link **https://github.com/IBM-EPBL/IBM-Project-25316-1659958338**