1.INTRODUCTION

1.1 Project Overview

Inventory is the stock of physical items such as materials, components, work-in-progress, finished goods, etc., held at a specific location at a specific time. Inventory is the merchandise that is purchased and/or produced and stored for eventual sale. Inventory is a list of what you have. In company accounts, inventory usually refers to the value of stocks, as distinct from fixed assets. An inventory would include items which are held for sale in the ordinary course of business or which are in the process of production for the purpose of sale, or which are to be used in the production of goods or services which will be for sale. Inventory is a list of names, quantities and/or monitory values of all or any group of items. Any quantifiable item that you can handle, buy, sell, store, consume, produce, or track can be considered inventory. This covers everything from office and maintenance supplies, to raw material used for manufacturing, to semi-finished and finished goods, to fuel used to power equipment used in the business.

1.2 Purpose

An inventory management system project that allows user to manage and maintain his/her inventory with ease. The inventory management system has been developed to allow users to add an inventory, delete an inventory, enter inventory quantity and other details, update inventory status and more. The inventory management system has its own intelligently managed support system that allows user to view and manage various inventories added in the system.

2. LITERATURE SURVEY

2.1 Existing Problem

Inconsistent Tracking:

Using manual inventory tracking procedures across different software and spreadsheets is time-consuming, redundant and vulnerable to errors. Even small businesses can benefit from a centralized inventory tracking system that includes accounting features.

Warehouse Efficiency:

Inventory management controls at the warehouse is labor-intensive and involves several steps, including receiving and put away, picking, packing and shipping. The challenge is to perform all these tasks in the most efficient way possible.

Inaccurate Data:

You need to know, at any given moment, exactly what inventory you have. Gone are the days when inventory could be counted once a year with an all-hands-on-deck approach.

Changing Demand:

Customer demand is constantly shifting. Keeping too much could result in obsolete inventory you’re unable to sell, while keeping too little could leave you unable to fulfill customer orders. Order strategies for core items, as well as technology to create and execute an inventory plan, can help compensate for changing demand.

Limited Visibility:

When your inventory is hard to identify or locate in the warehouse, it leads to incomplete, inaccurate or delayed shipments. Receiving and finding the right stock is vital to efficient warehouse operations and positive customer experiences

2.2 Reference

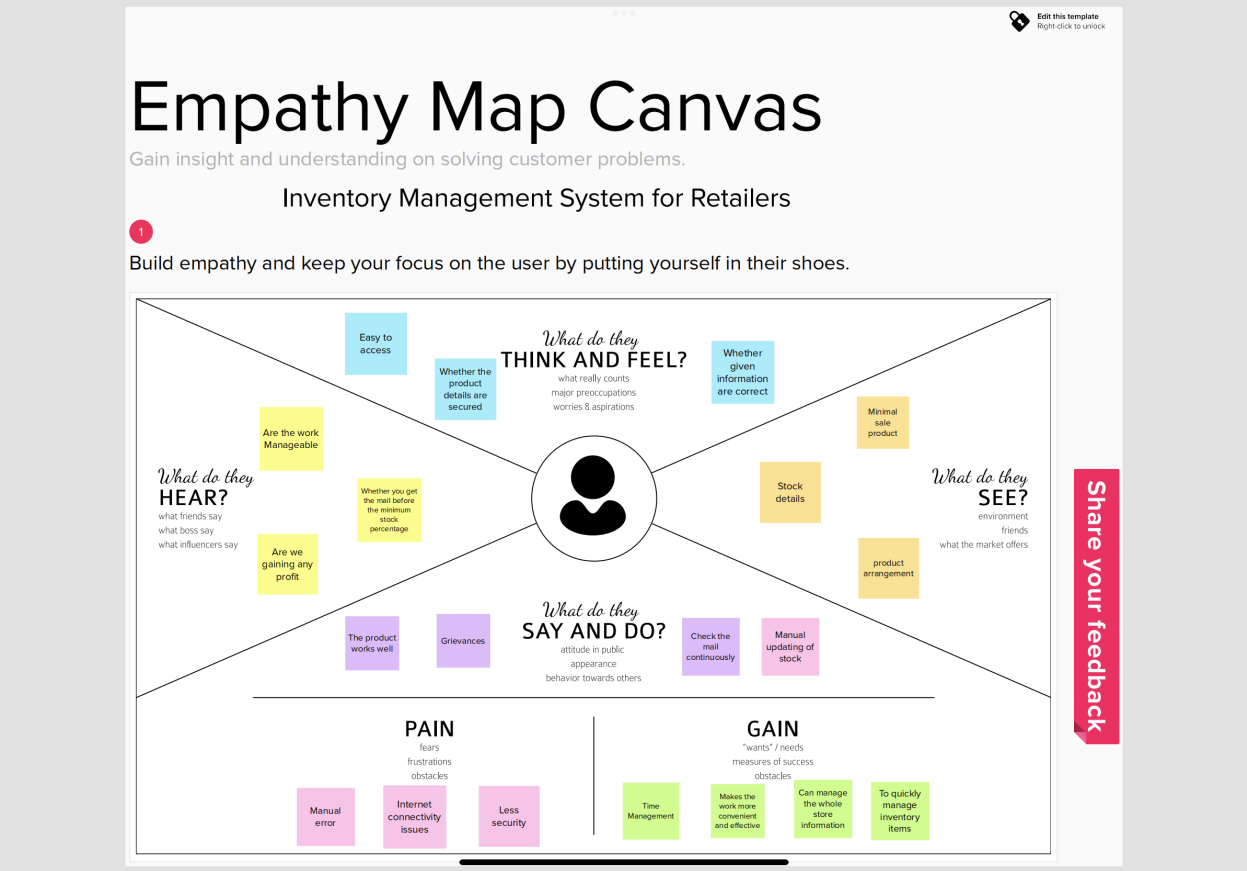
https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management-challenges.shtml

2.3 Problem Statement Definition

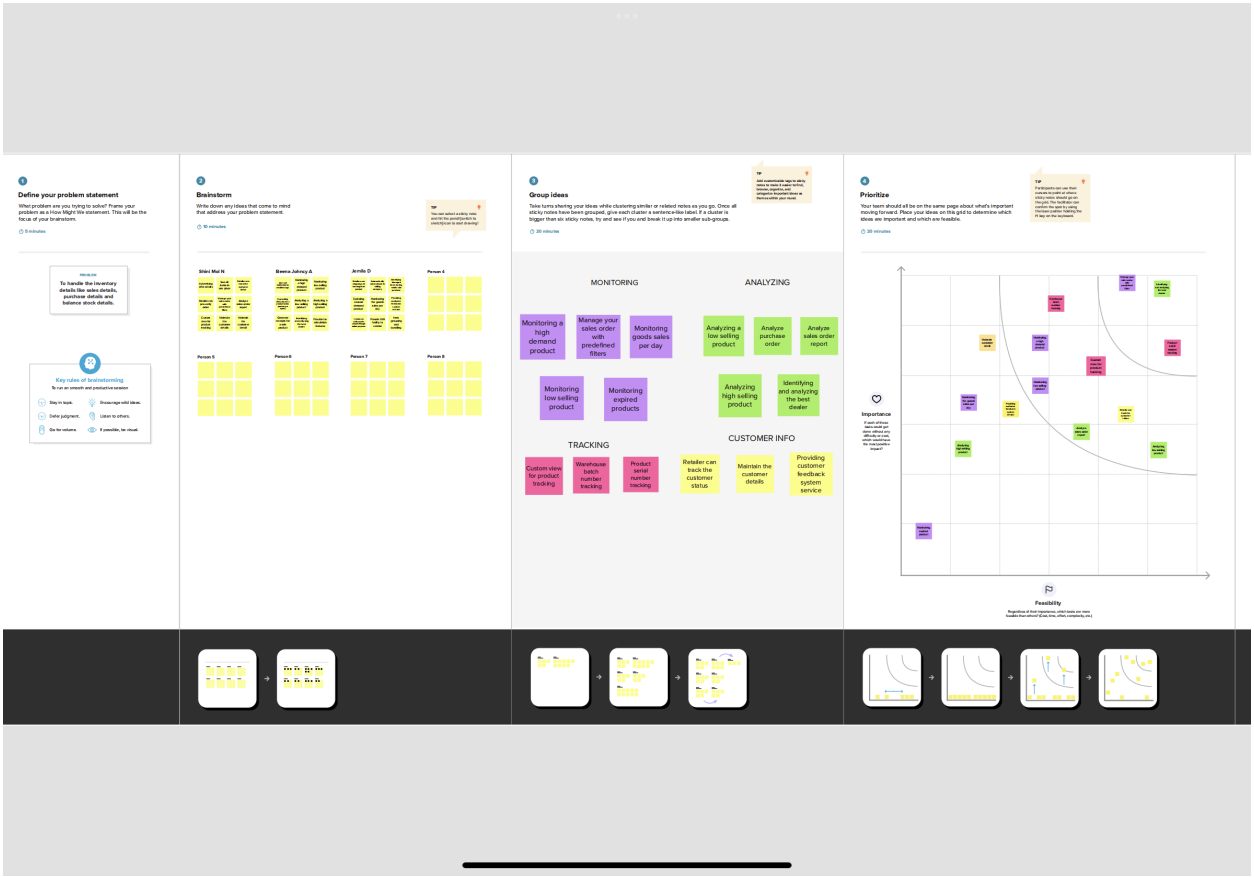
Demand is frequently unpredictable in inventory systems, and lead times can often vary. Managers frequently keep a safety supply to minimize shortages. In such cases, it's difficult to say what order amounts and reorder points will result in the lowest total inventory cost. The inventory issue refers to the general issue of deciding how much inventory to keep on hand in expectation of possible demand. Loss occurs when a business is unable to meet demand (for example, when a store loses sales or when soldiers in a war run out of ammunition) or when commodities are stocked for which there is no demand.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



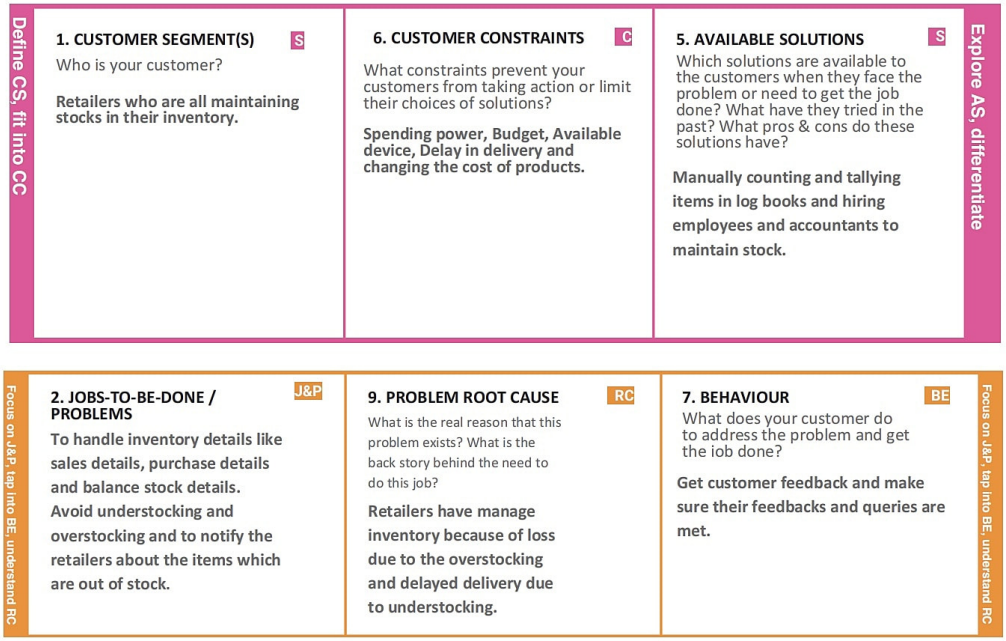
3.2 Ideation & Brainstorming



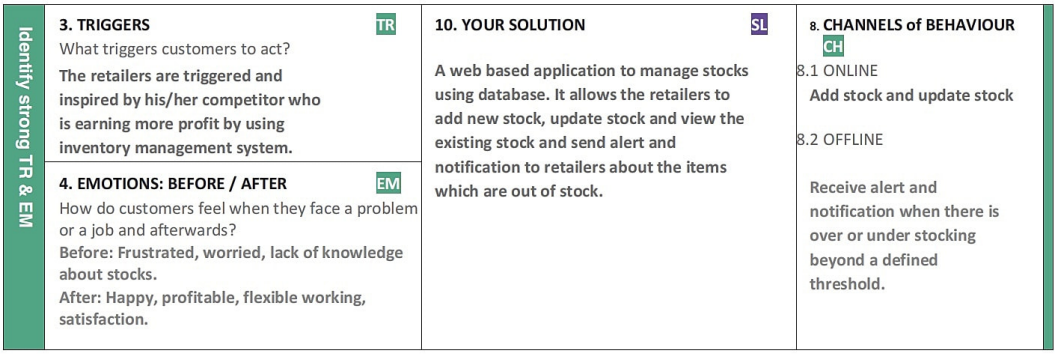
3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To handle the inventory details like sales details, purchase details and balance stock details and make the stock manageable and simplify the use of inventory in the retail store.
2.	Idea / Solution description	Inventory management's major goal is to make ordering, stocking, storing, and using inventory as simple and efficient as possible for firms. An inventory system's main function is to keep track of products and supplies.
3.	Novelty / Uniqueness	Inventory management software integrates with barcode scanners for instant product identification. Receive alerts and notifications when there is over or under stocking beyond a defined threshold.
4.	Social Impact / Customer Satisfaction	Return and replace facility available for the damaged products. There’s always enough stock to fulfil customer orders and proper warning of a shortage.
5.	Business Model (Revenue Model)	With proper inventory management, we spend money on inventory that sells, so cash is always moving through our business.
6.	Scalability of the Solution	Small and large scale retailers can use this system and it can be collaborated with multiple retailers and have clear stock information.

3.4 Problem Solution Fit



Identify strong TR & EM



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Gmail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Username/Email-ID Login with Password
FR-4	Monitors stock of the product	Monitors the stock of the product and updates the stock of the product continuously after selling each product.
FR-5	Low stock products are shown	Low stock products have been highlighted by red colour.
FR-6	Alert notification	By monitoring stock of the product, notification or message will be send to the retailer if the stock is under beyond the threshold.

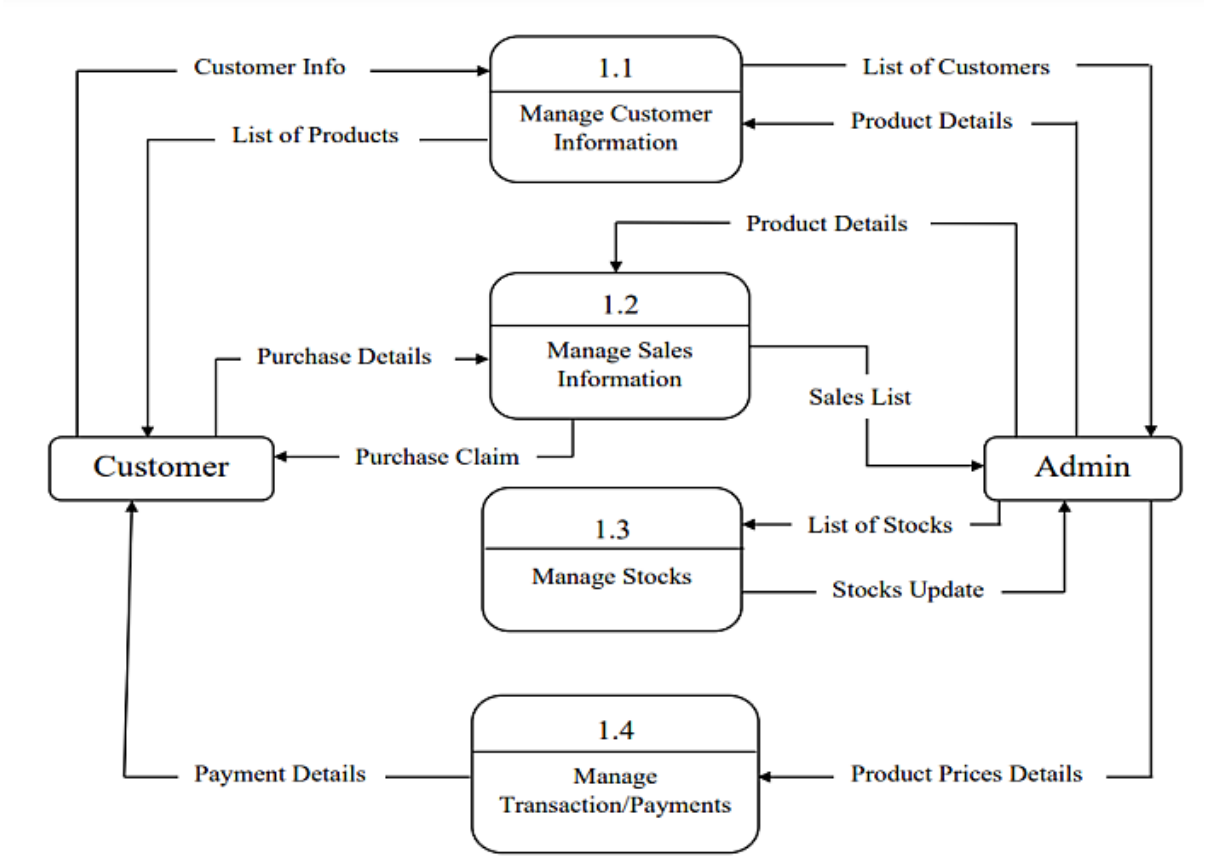
4.2 Non-Functional requirement

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The system must be easy to use for retailers such that they do not need to read an extensive amount of manuals. The system must be quickly accessible by retailers. The system must be intuitive and simple in the way it displays all relevant data and relationships. The menus of the system must be easily navigable by the users with buttons that are easy to understand.
NFR-2	**Security**	Inventory security aims to prevent inventory losses – for example, due to incorrect storage, theft, or incorrect incoming goods inspection – so that the correct stock is always available.
NFR-3	**Reliability**	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 product given by the user and provide estimations and inventory status in relevance with the newly updated entities. The system must provide a password enabled login to the user to avoid any foreign entity changing the data in the system. The system should provide the user updates on completion of requested processes and if the requested processes fail, it should provide the user the reason for the failure. The system should not update the data in any database for any failed processes.
NFR-4	**Performance**	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 products 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.
NFR-5	**Availability**	The software will be available only to the administrator of the organization and the product, as well as customer details, will be recorded by him. He can manage the inventory.
NFR-6	**Scalability**	Scalability is an aspect or rather a functional quality of a system, software or solution. This proposed system for inventory management system can accommodate expansion without restricting the existing workflow and ensure an increase in the output or efficiency of the process. It is scalable that we are going to use data in kilobytes so that the quite amount of storage is satisfied.

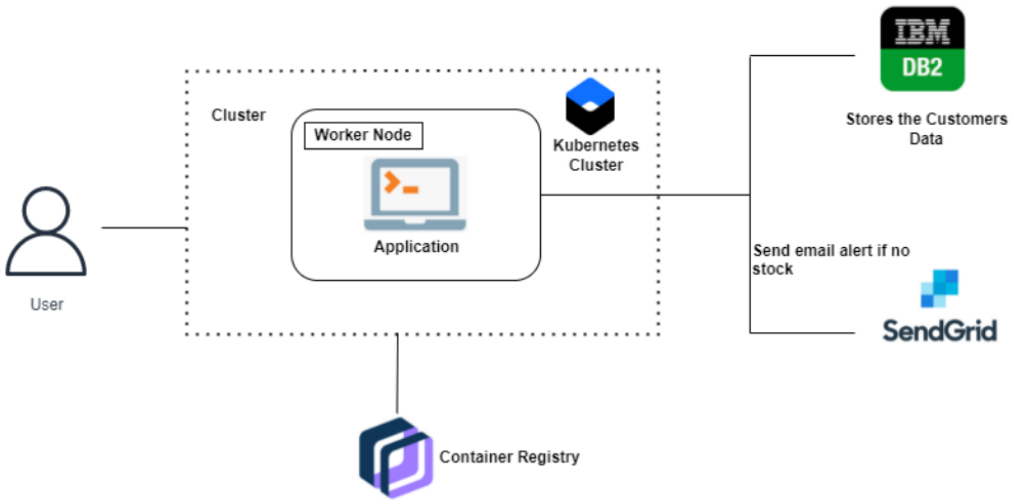
5. PROJECT DESIGN

5.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering 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	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can login by entering Gmail and password	High	Sprint-1
	Dashboard	USN-6	As a user, I can track data of sales of products and inventory levels	I can track data of sales of products and inventory levels	High	Sprint-1
Customer (Web user)	Registration	USN-7	As a user, I will receive for the application by entering my email, password and confirming my password	I can access my account/dashboard	High	Sprint-1
Customer Care Executive		USN-8	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Spirnt-1
Administrator		USN-9	As a user, I can register for the application through facebook	I can register & access the dashboard with facebook login	Low	Sprint-3
		USN-10	As a user, I can register for the application through Gmail	I can register for the application through Gmail	Medium	Sprint-2

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Shini Mol. N Beena Johncy Jemila
Sprint-1	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Shini Mol. N Beena Johncy Jemila
Sprint-2	Registration	USN-3	As a user, I can register for the application through Facebook	2	Low	Shini Mol. N Beena Johncy Jemila
Sprint-1	Registration	USN-4	As a user, I can register for the application through Gmail	2	Medium	Shini Mol. N Beena Johncy Jemila
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	1	High	Shini Mol. N Beena Johncy Jemila

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Dashboard	USN-6	As a user, I must be able to see my details on the dashboard	2	Medium	Shini Mol. N Beena Johncy Jemila
Sprint-3	Inventory	USN-7	As a retailer, I should be able to alter product details in the app	2	High	Shini Mol. N Beena Johncy Jemila
Sprint-3	Inventory	USN-8	As a retailer, I should be able to add or remove quantity of products in the app.	2	Medium	Shini Mol. N Beena Johncy Jemila
Sprint-3	Inventory	USN-9	As a retailer, I should get alert on stock shortage or unavailability.	1	Low	Shini Mol. N Beena Johncy Jemila
Sprint-4	Maintenance	USN-10	As a administrator, I should be able to edit details of the users of the app.	2	High	Shini Mol. N Beena Johncy Jemila
Sprint-4	Feedback	USN-11	As a customer care team member, I should be able to get feedback from the users	2	Low	Shini Mol. N Beena Johncy Jemila

7. CODING & SOLUTIONING

```
from flask import Flask, render_template, url_for, request, redirect
from flask_sqlalchemy import SQLAlchemy
from collections import default dict
from date time import datetime

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


class Product(db.Model):

    __table name__ = '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):
    __table name__    = '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('/login/', methods=["POST", "GET"])
def login():
    if (request.method == "POST") and ('email_name' in request.form) and
('user_password' in request.form):
        email_name = request.form["email_name"]
        user_password = request.form["user_password"]

        if (email_name=="admin@gmail.com") and (user_password=="123456"):

            try:
                products    =
Product.query.order_by(Product.date_created).all()
                locations   =
Location.query.order_by(Location.date_created).all()
                return render_template("home.html", products = products,
locations = locations)
            except:
                return "There Was an issue while Login to Portal"

        else:
            return "There Was an issue while Login to Portal..Enter Proper
Credientials"
```

```
    else:
        return render_template("index.html")

@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("home.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>")
def deleteLocation(name):
    location_to_delete = Location.query.get_or_404(name)

    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()

        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))
    locations = ProductMovement.query.\
        filter( ProductMovement.product_id == product).\
        filter(ProductMovement.to_location != '').\
        add_columns(ProductMovement.from_location,
ProductMovement.to_location, ProductMovement.qty).\
        all()

    for key, location in enumerate(locations):
        if(locationDict[location.to_location] and
locationDict[location.to_location]["qty"]):
            locationDict[location.to_location]["qty"] += location.qty
        else:
            locationDict[location.to_location]["qty"] = location.qty

    return locationDict


@app.route("/dub-locations/", methods=["POST", "GET"])
def getDublicate():
    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. TESTING

8.1 Test Cases

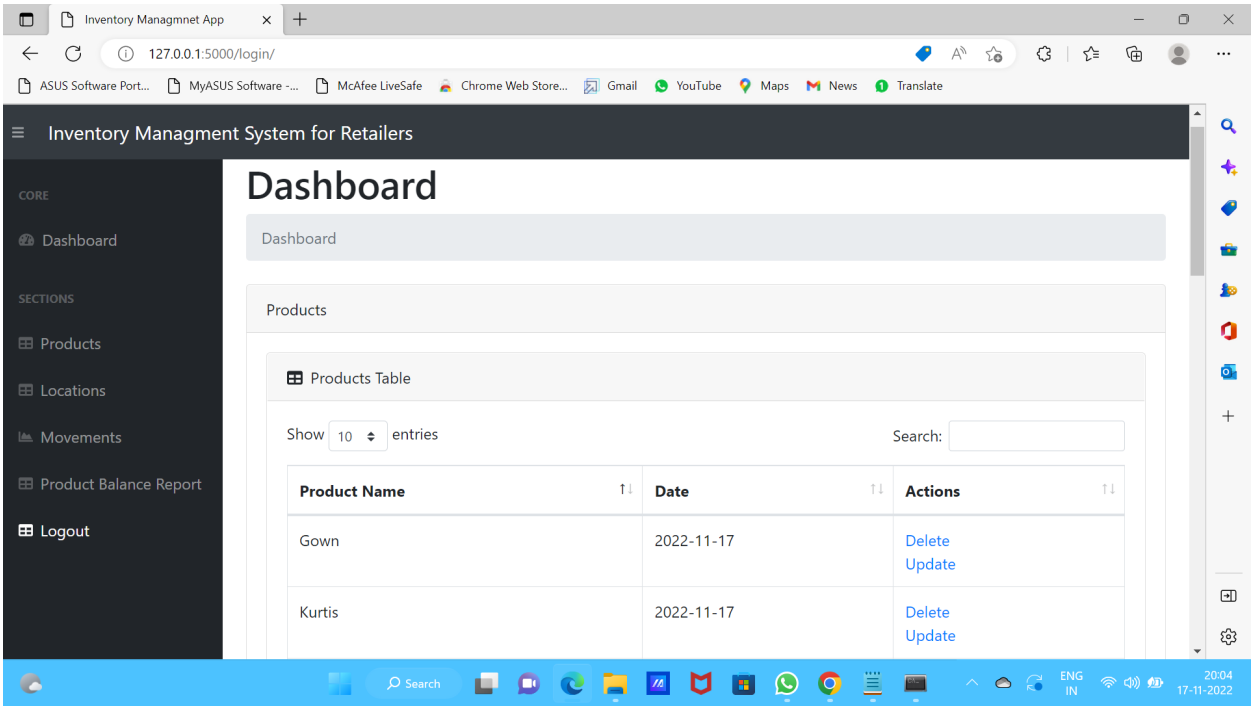
A test case is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

8.2 User Acceptance Testing

User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

9. RESULT


Inventory Managmnet App
127.0.0.1:5000/login/
ASUS Software Port...
MyASUS Software -...
McAfee LiveSafe
Chrome Web Store...
Gmail
YouTube
Maps
News
Translate
Inventory Managment System for Retailers
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Logout
Dashboard
Dashboard
Products
Products Table
Show 10 entries
Search:
Product Name
Date
Actions
Gown
2022-11-17
Delete
Update
Kurtis
2022-11-17
Delete
Update
Search
ENG
IN
20:04
17-11-2022

10. ADVANTAGES & DISADVANTAGES

Advantages

There are many advantages of the inventory management system. Thus, summarized below which can avoid the company from suffering from big economical losses and other problems that may occur during the everyday operations of the firm that can be observed as the materials being out of stock or machine failures and many other operations happenings on a day-to-day basis.

Disadvantages

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.

11. CONCLUSION

Inventory management has to do with keeping accurate records of goods that are ready for shipment. This often means having enough stock of goods to the inventory totals as well as subtracting the most recent shipments of finished goods to buyers. When the company has a return policy in place, there is usually a sub-category contained in the finished goods inventory to account for any returned goods that are reclassified or second grade quality. Accurately maintaining figures on the finished goods inventory makes it possible to quickly convey information to sales personnel as to what is available and ready for shipment at any given time by buyer.

Inventory management is important for keeping costs down, while meeting regulation. Supply and demand is a delicate balance, and inventory management hopes to ensure that the balance is undisturbed. Highly trained Inventory management and high-quality software will help make Inventory management a success. The ROI of Inventory management will be seen in the forms of increased revenue and profits, positive employee atmosphere, and on overall increase of customer satisfaction.

12. FUTURE SCOPE

This project helps us to access and manage the information easily. And also helps to verify the stock currently available with them and to update the stock when necessary. This project reduce the time to search the product from the current available stock. The role of an inventory system is to track your products and supplies. Inventory management System.

13. APPENDIX

Github Link - https://github.com/IBM-EPBL/IBM-Project-40423-1660629304

Project Demo Link -

https://drive.google.com/file/d/1ZBwP0h_PPRIeQ157LxDvYO5fEvU-hFfP/view?usp=drivesdk