PROJECT REPORT

DATE:	**19.11.2022**
PROJECT NAME:	**INVENTORY MANAGEMENT SYSTEM FOR RETAILERS**
TEAM ID:	**PNT2022TMID02997**

SUBMITTED BY

SANJAY KANNAN K – TEAM LEADER

ROHIT PRIYAN M – TEAM MEMBER 1

SIVABALAN G -- TEAM MEMBER 2

SHYAM SUNDAR V -- TEAM MEMBER 3

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

NOV 2022

CONTENTS:

1. INTRODUCTION

1.1 PROJECT OVERVIEW

The project Inventory Management System is a complete desktop based application designed on full stack technology using Visual Studio Software. The main aim of the project is to develop Inventory Management System Model software in which all the information regarding the stock of the organization will be presented. It is an intranet based desktop application which has admin component to manage the inventory and maintenance of the inventory system. This desktop application is based on the management of stock of an organization. The application contains general organization profile, sales details, Purchase details and the remaining stock that are presented in the organization. There is a provision of updating the inventory also. This application also provides the remaining balance of the stock as well as the details of the balance of transaction. Each new stock is created and entitled with the named and the entry date of that stock and it can also be update any time when required as per the transaction or the sales is returned in case. Here the login page is created in order to protect the management of the stock of organization in order to prevent it from the threads and misuse of the inventory.

1.2 PURPOSE

Inventory Management System is an online software application which fulfills the requirement of a typical Stock Analysis in various godowns. It provides the interface to users in a graphical way to manage the daily transactions as well as historical data. it also provides the management reports like monthly inwards, monthly deliveries and monthly returns. This application maintains the centralized database so that any changes done at a location reflects immediately. This is an online tool so more than one user can login into system and use the tool simultaneously. The aim of this application is to reduce the manual effort needed to manage transactions and historical data used in various go downs. It also this application provides an interface to users to view the details like the daily Stock Statements of all godowns.

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

2. Literature Review

Products are considered as the business resources for the organization. This includes managing the product with appropriate way to review any time as per the requirement. Therefore it is important to have a computer based IMS which has the ability to generate reports, maintain the balance of the stock, details about the purchase and sales in the organization. Before developing this application we came up with 2Inventory Management System existing in the market, which helps to give the knowledge for the development of our project. These application software are only used by the large organization but so we came up with the application which can be used by the small company for the management of their stock in the production houses. After analysing the other inventory management system we decided to include some of common and key features that should be included in every inventory management system. So we decided to include those things that help the small organization in away or other.

2.1 EXISTING PROBLEM

Current system is a manual one in which users are maintaining ledgers, books etc to store the information like suppliers details, inwards, deliveries and returns of items in all go downs, customer details as well as employee details. It is very difficult to maintain historical data. Also regular investments need to purchase stationary every year In the existing system, the inventory management is handled manually, which is highly tedious. Some of the important business operations are estimating the requirement of new raw material, dealing in the production of Purchase order, purchase invoice, sales invoice and debit note. All these operations are performed by a team of skilled members which are prompt in financial calculations and have a sharp memory. The operations are handled in an effective way, but the process is time taking and subjected to human errors. The present system is not is exception consultant encountering all the above problems.

1. Time Consuming.
2. It is very tedious.
3. All information is not placed separately
4. Lot of paper work
5. Slow data processing

2.2 REFERENCES

[1] Joshni S Pasaribu, “Development of web based inventory information system” Vol-1, No. 2 (2021) pp 24-31, eISSN: 2775-2674.

[2] Trupti Shirsat, “Online inventory management system” Vol-2 No. 6 (2019), pp 118-119, ISSN: 2581- 7175.

[3] Varalakshmi GS, “A review of inventory management system” Vol-10 No. 6 (2021), pp 421-423, ISSN: 2278-1021.

[4] Anas M. Atieh, “Performance improvement of inventory management system process by an automated warehouse management system”, (2016) pp 568-572, ISSN: 2212-8271.

[5] E S Soegoto, “Web Based Online Inventory Information System”, IOP-879 012125, (2020) Rashmi Mishra, “AN INFORMATIVE LITERATURE REVIEW ON INVENTORY CONTROL SYSTEM” Vol-5 No. 8, (2018) pp 614-618, ISSN: 2349-5162

[6] Rashmi Mishra, “AN INFORMATIVE LITERATURE REVIEW ON INVENTORY CONTROL SYSTEM” Vol-5 No. 8, (2018) pp 614-618, ISSN: 2349-5162

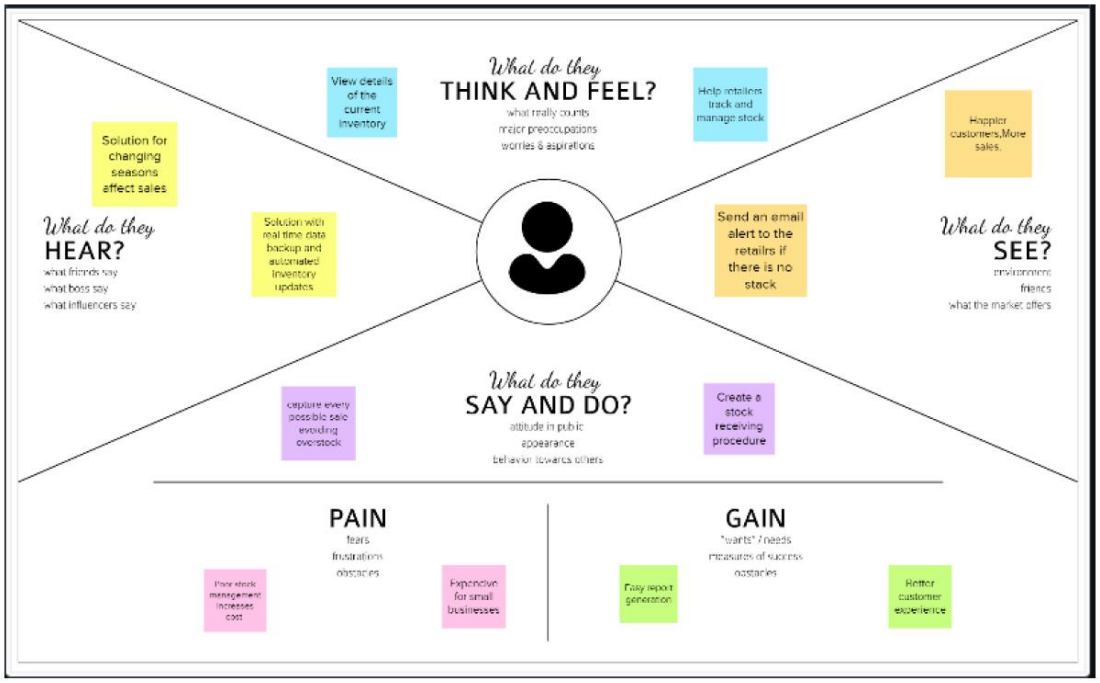
[7] PRATAP CHANDRAKUMAR. R, “A STUDY ON INVENTORY MANAGEMENT AND CONTROL” Vol-3 No 5 (2017) pp 1524-1532, ISSN: 2395-4396

2.3 PROBLEM STATEMENT DEFINITION

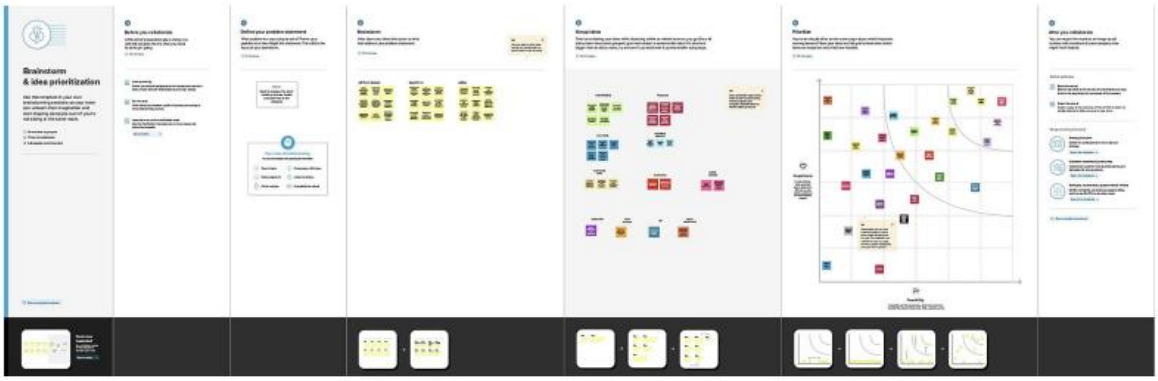
Inventory management is the process of ensuring you carry merchandise that shoppers want, with neither too little nor too much on hand. By managing inventory, retailers meet customer demand without running out of stock or carrying excess supply. In practice, effective retail inventory management results in lower costs and a better understanding of sales patterns. Retail inventory management tools and methods give retailers more information on which to run their businesses. Applications have been developed to help retailers track and manage stocks related to their own products. The System will ask retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION & BRAINSTORMING



3.3 PROPOSED SOLUTION

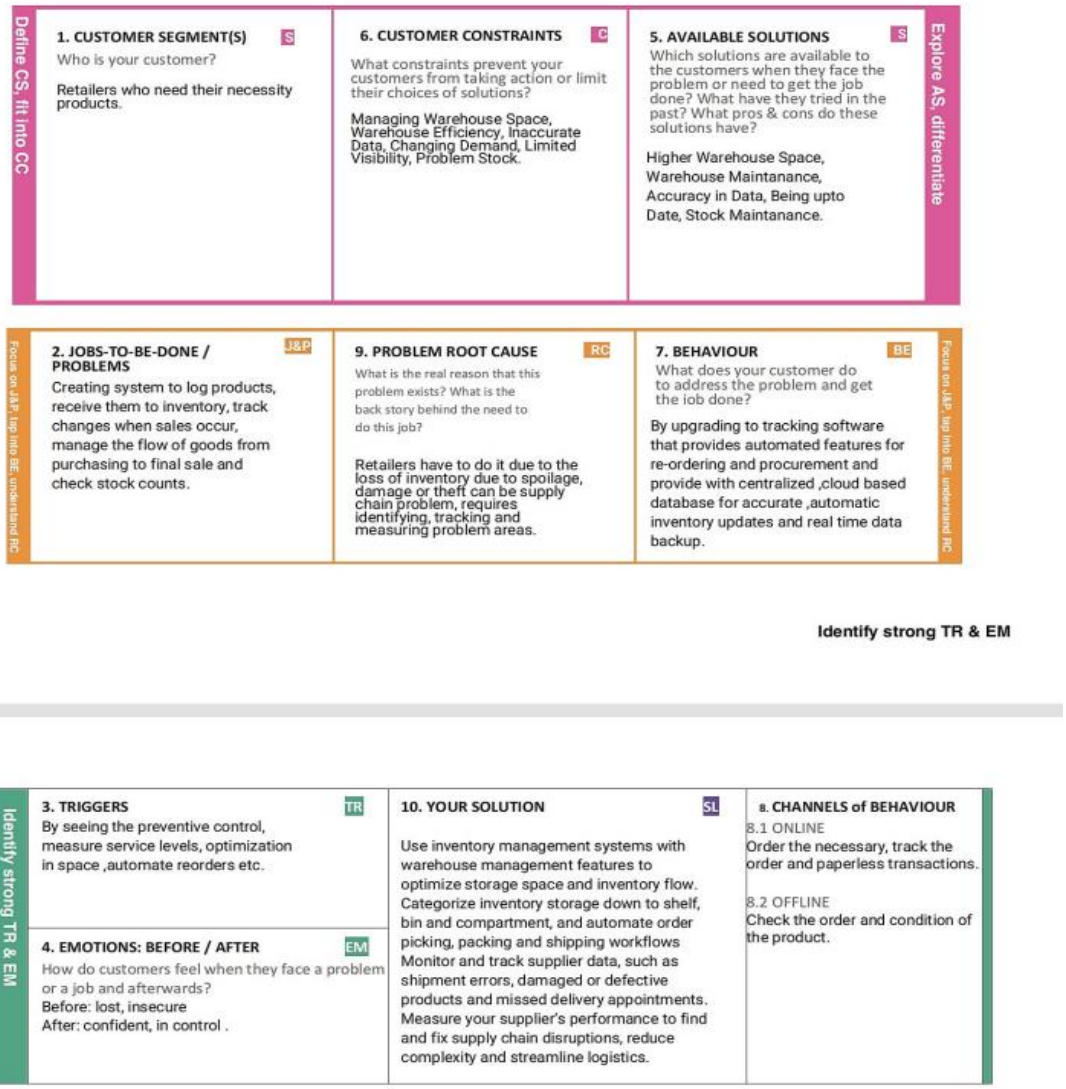
Home: This first module manages Home Screen Which is Provide a Home Page of my Software. After clicking home butten. Butten will provide Welcome Screen of the Software etc

Sales: This is Provide Sales information And Sales Page it is contain sales_id, Product_code , Product_name , Quantity, Revenue, Sold by etc.

Products:. It is hold the details of product with product code, product name, cost price selling price brand etc.

Purchase: this is contain detail about purchase . It will provide purchase screen which is hold some value like purchase id ,product code ,product name ,quantity ,total cost

3.4 PROBLEM SOLUTION FIT



4 REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

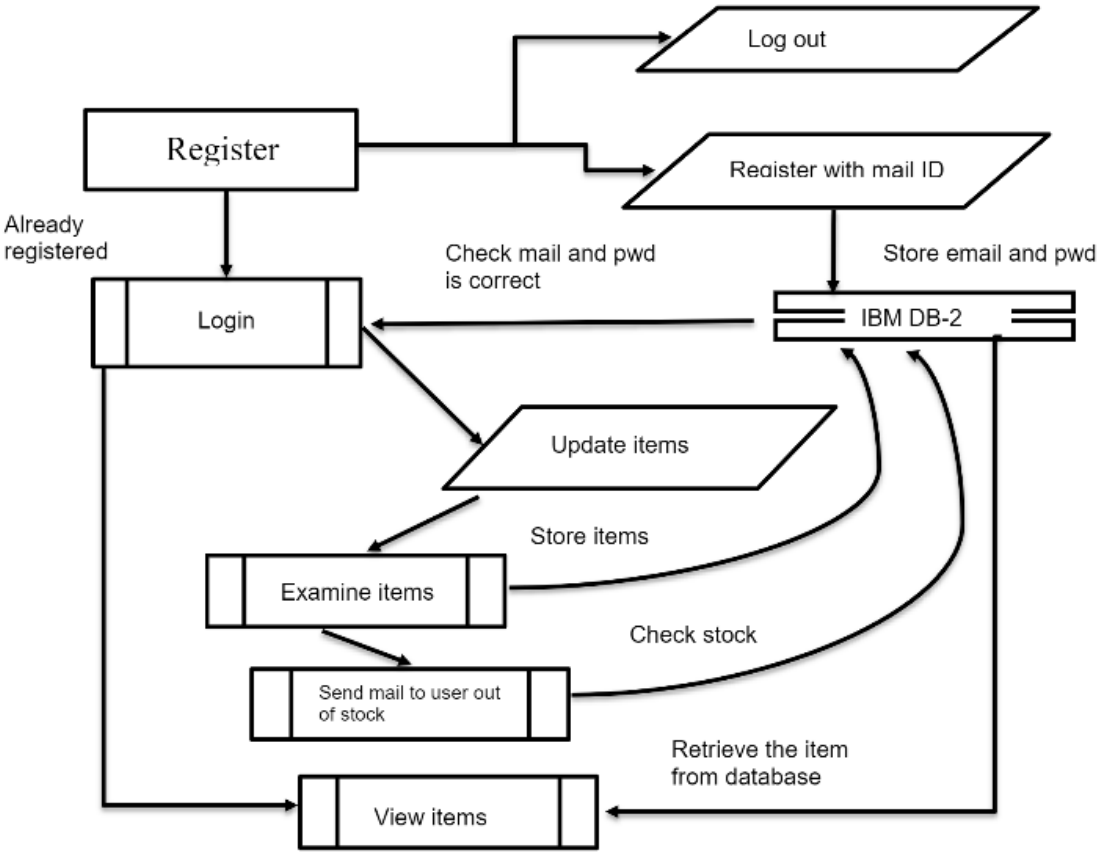
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through User Form Registration through Gmail/ Phone Number Registration through Facebook
FR-2	User Confirmation	Confirmation via Email, Phone Number
FR-3	Adding,updating,deleting the products	Add or update or delete products using the options given in dashboard
FR-4	High demand products	User can view the high demand products by using the previous sales details
FR-5	Barcode scanning	User can check or update products by scanning the Barcode
FR-6	Invoice generation	User can generate invoice by their email for future reference

4.2 NON-FUNCTIONAL REQUIREMENTS

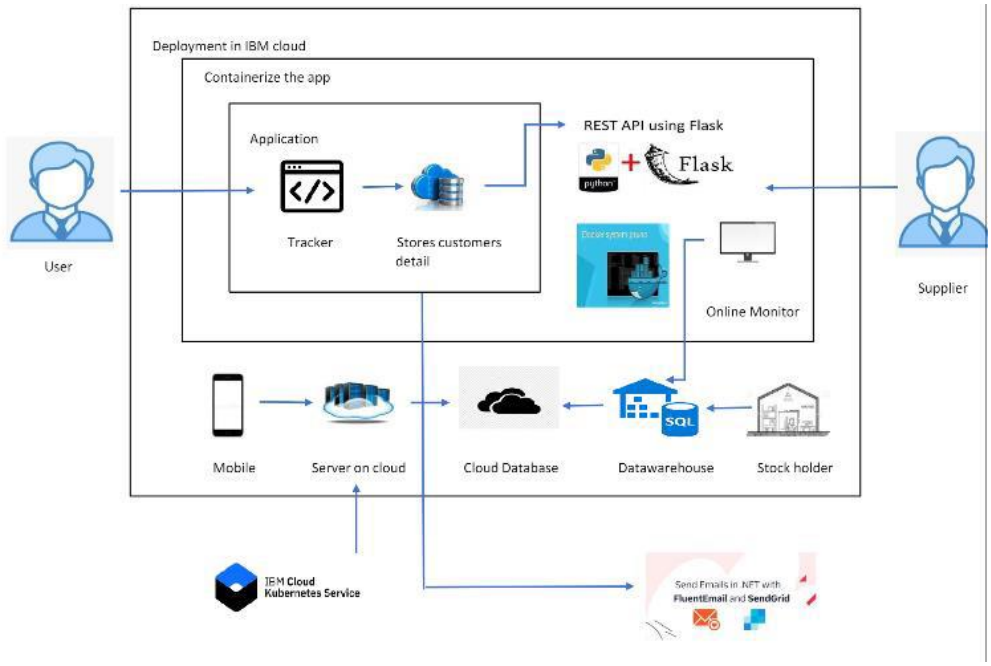
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Stocks are managed and tracked easily
NFR-2	Security	Two step verification
NFR-3	Reliability	Huge number of stock products can be easily calculated
NFR-4	Performance	The website’s loading time should be less than 5 seconds
NFR-5	Availability	All kind of retailers and wholesale business man can use .
NFR-6	Scalability	Starting from retailer shop owners to department store owners can use our product

5 PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 SOLUTION & TECHNICAL ARCHITECTURE



5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Retailer(Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I will be redirected to login page	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 verify the OTP number	Medium	Sprint-1
Login	USN-5		As a user, I can log into the application by entering email & password	I can access my account / dashboard	High	Sprint-1
Dashboard	USN-6		As a user,I can update stock in & out count details	Updation can be made through barcode scanning	High	Sprint -2
Dashboard	USN-7		As a user,I can check the low stock details through alert message	Alert message can be received by registered mail	High	Sprint -1

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Gathering information by referring technical papers, research publications describes	15 SEPTEMBER 2022

	literature survey.	
Prepare Empathy Map	To establish users pros and cons, prepare the empathy map canvas on problem statement.	19 SEPTEMBER 2022
Ideation	Establishing brainstorm sessions and emphasize the top ideas based on the importance of scalability and feasibility.	19 SEPTEMBER 2022
Proposed Solution	Prepare the proposed solution which describes idea, uniqueness, customer satisfaction, business model and scalability of solution.	23 SEPTEMBER 2022
Problem Solution Fit	Prepare problem - solution fit which describes the existence of a problem.	26 SEPTEMBER 2022
Solution Architecture	Defining process of developing solution based on pre defined processes.	26 SEPTEMBER 2022
Customer Journey	Prepare a customer journey map which understand the customers on user interaction and experiences from scratch to finding solution.	04 OCTOBER 2022
Functional Requirement	Prepare the functional requirement document.	10 OCTOBER 2022
Data Flow Diagrams	Draw the data flow diagrams based on problem statement.	12 OCTOBER 2022
Technology Architecture	Prepare a technology architecture diagram.	13 OCTOBER 2022
Prepare Milestone & Activity List	Prepare the milestones & activity list foe the project.	18 OCTOBER 2022
Project Development - Delivery of Sprint-1, 2, 3 & 4	Develop & submit the developed code by testing it.	17 NOVEMBER 2022

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Functional Requirements (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Login	USN-1	As a user, I can login to user dashboard and see the information about the products in inventory	3	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V
Sprint-2	Dashboard	USN-2	As a user, I can add or update or delete products using the options given in dashboard	4	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V
Sprint-2	Dashboard	USN-3	As a user, I can track the product and manage the Product.	2	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V
Sprint-3	Dashboard	USN-4	As a user, I can track the movements of the product in the inventory	3	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V
Sprint-3	Dashboard	USN-5	As a user, I can track the product balance report in the inventory	4	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V
Sprint-4	Software Testing and Deployment	USN-6	As a user want to access the Application without any drawbacks, We need test the application before release	2	High	Sanjay Kannan K Rohit Priyan M Sivabalan G Shyam Sundar V

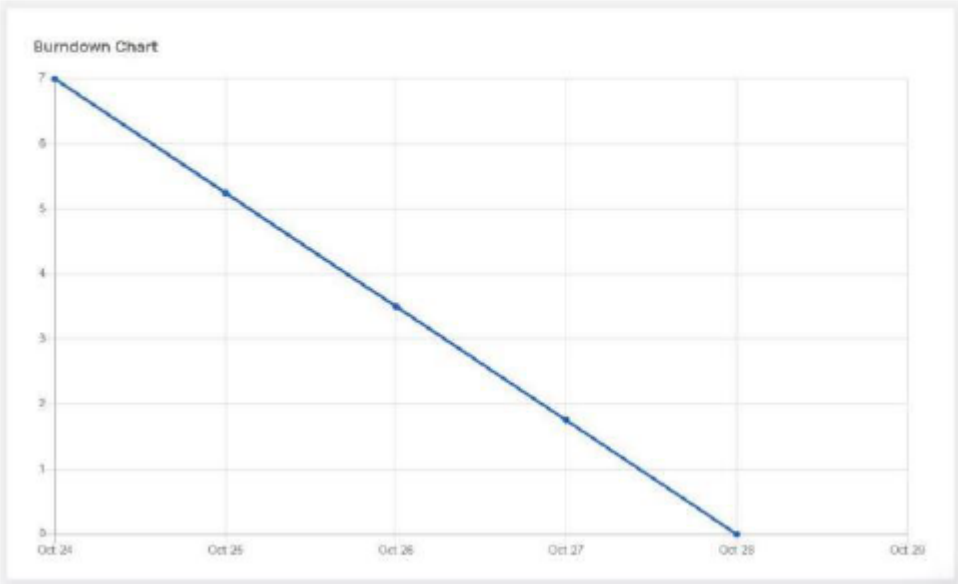
Project Tracker, Velocity & Burndown Chart: (4 Marks)

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

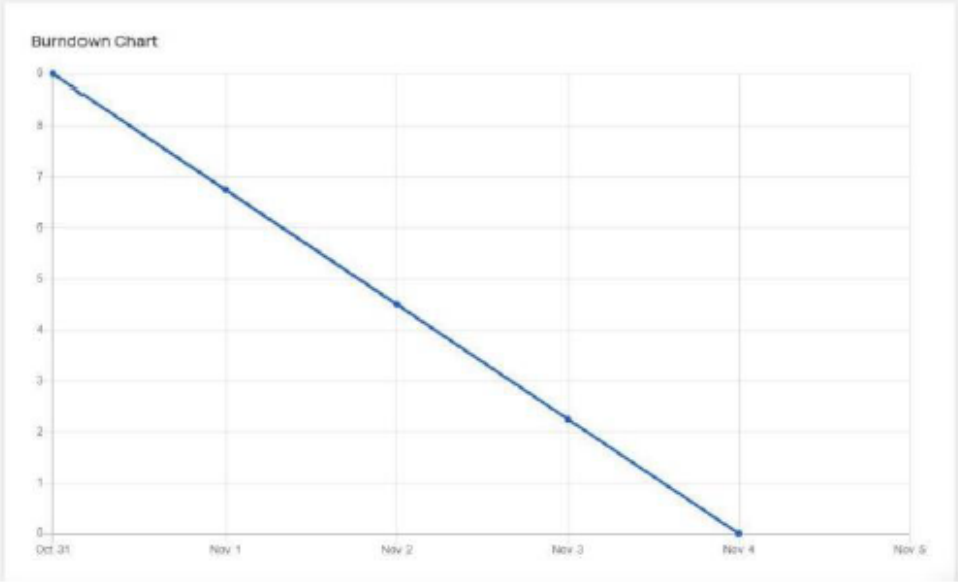
Velocity:
Sprint 1(AV) =3.34
Sprint 2(AV) =3.34
Sprint 3(AV) =3.34
Sprint 4(AV) =3.34

BURNDOWN CHART:

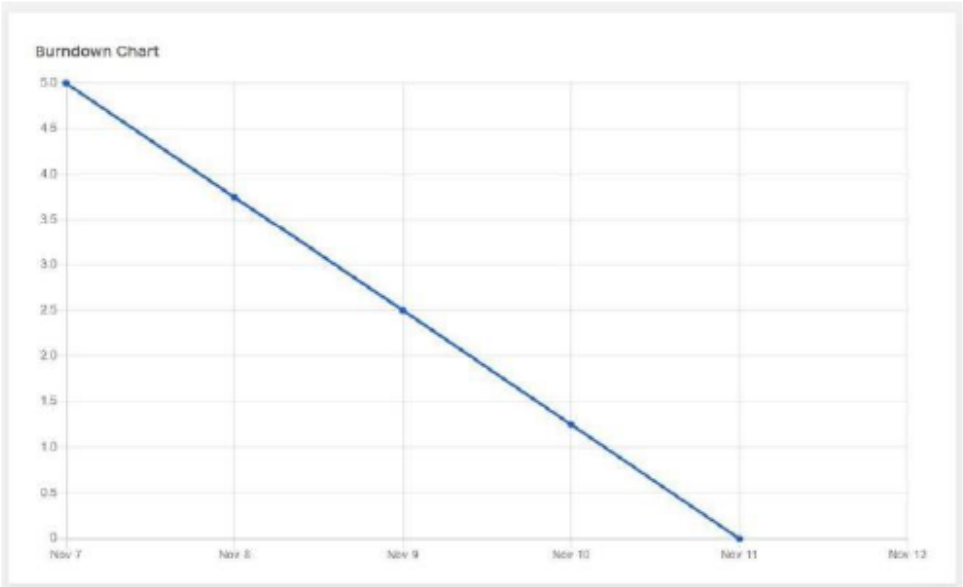
SPRINT – I:



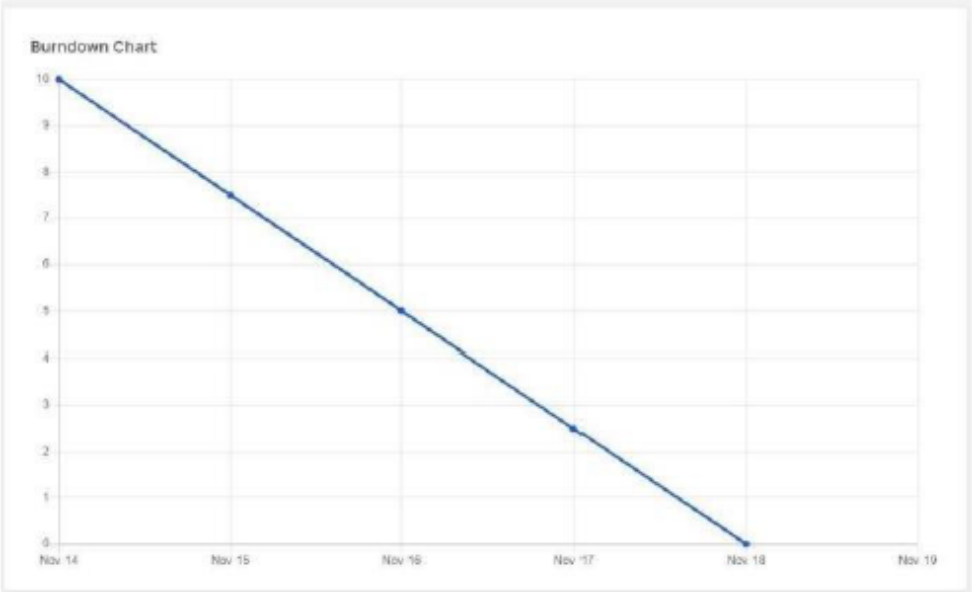
SPRINT – II



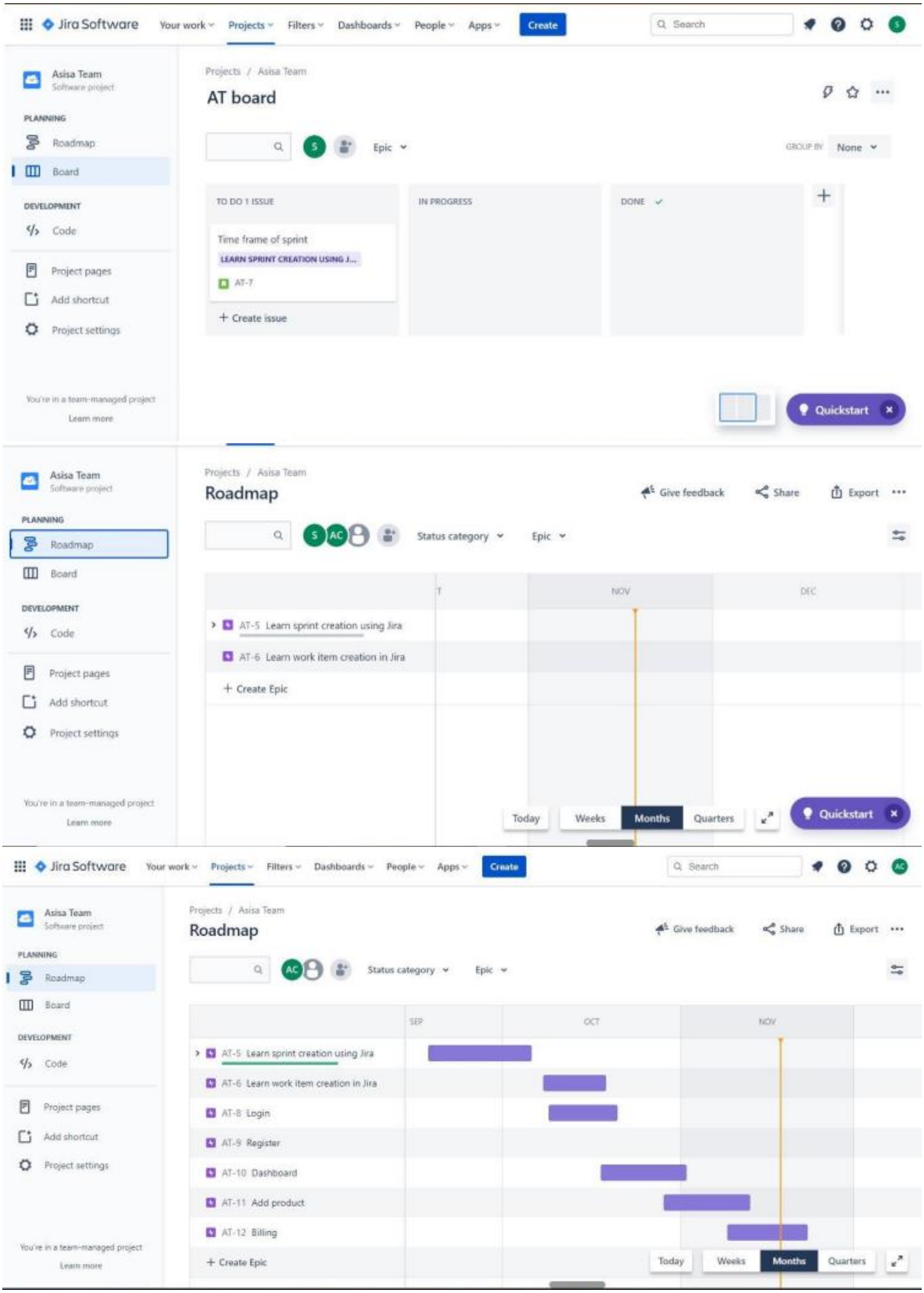
SPRINT – III:



SPRINT – IV:



6.3 REPORTS FROM JIRA



7. CODING AND SOLUTIONING

7.1 FEATURE 1

APP.PY

```
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 login():
    try:
        return render_template("login.html")
    except:
        return "error"


@app.route('/index', 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)
```

```python
@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(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()

        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}
```

```python
@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)
```

LOGIN

```
<!DOCTYPE HTML>
<html lang="en" >
<html>
<style>
  @import url(https://fonts.googleapis.com/css?family=Roboto:300);

.body{
  background-image: url('https://ordinaryfaith.net/wp-
content/uploads/2016/03/Gray-plain-website-background.jpg');
  background-repeat: no-repeat;
  background-size: cover;
}

.login-page {
  width: 360px;
  padding: 8% 0 0;
  margin: auto;

}
.form {
  position: relative;
  z-index: 1;
  background: #48c9b0;
  max-width: 360px;
  margin: 0 auto 100px;
  padding: 45px;
  text-align: center;
  box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.2), 0 5px 5px 0 rgba(0, 0, 0, 0.24);

}
.form input {
  font-family: FontAwesome, "Roboto", sans-serif;
  outline: 0;
  background: #f2f2f2;
  width: 100%;
  border: 0;
  margin: 0 0 15px;
  padding: 15px;
  box-sizing: border-box;
  font-size: 14px;
border-radius:10px;

}
.form button {
  font-family: "Titillium Web", sans-serif;
  font-size: 14px;
```

```
  font-weight: bold;
  letter-spacing: .1em;
  outline: 0;
  background: #17a589;
  width: 100%;
  border: 0;
border-radius:30px;
  margin: 0px 0px 8px;
  padding: 15px;
  color: #FFFFFF;
  -webkit-transition: all 0.3 ease;
  transition: all 0.3 ease;
  cursor: pointer;
  transition: all 0.2s;


}
.form button:hover,.form button:focus {
  background: #148f77;
    box-shadow: 0 5px 10px rgba(0, 0, 0, 0.2);
  transform: translateY(-4px);
}
.form button:active {
  transform: translateY(2px);
  box-shadow: 0 2.5px 5px rgba(0, 0, 0, 0.2);
}

.form .message {

  margin: 6px 6px;
  color: #808080;
  font-size: 11px;
  text-align: center;
  font-weight: bold;
  font-style: normal;



}
.form .message a {
  color: #FFFFFF;
  text-decoration: none;
  font-size: 13px;
}
.form .register-form {
  display: none;
}
.container {
  position: relative;
```

```
  z-index: 1;
  max-width: 300px;
  margin: 0 auto;
}
.container:before, .container:after {
  content: "";
  display: block;
  clear: both;
}
.container .info {
  margin: 50px auto;
  text-align: center;
}
.container .info h1 {
  margin: 0 0 15px;
  padding: 0;
  font-size: 36px;
  font-weight: 300;
  color: #1a1a1a;
}
.container .info span {
  color: #4d4d4d;
  font-size: 12px;
}
.container .info span a {
  color: #000000;
  text-decoration: none;
}
.container .info span .fa {
  color: #EF3B3A;
}
body {
  background: #76b852; /* fallback for old browsers */
  background: -webkit-linear-gradient(right, #76b852, #8DC26F);
  background: -moz-linear-gradient(right, #76b852, #8DC26F);
  background: -o-linear-gradient(right, #76b852, #8DC26F);
  background: linear-gradient(to left, #76b852, #8DC26F);
  font-family: "Roboto", sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
</style>
<head>
  <title>Login</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="login_style.css">
```

```
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <link href='https://fonts.googleapis.com/css?family=Titillium+Web:400,300,600' rel='stylesheet' type='text/css'>
  <link href='https://fonts.googleapis.com/css?family=Titillium+Web:400,300,600' rel='stylesheet' type='text/css'>
  <script src="https://unpkg.com/@lottiefiles/lottie-player@latest/dist/lottie-player.js"></script>
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.1/css/all.css" integrity="sha384-vp86vTRFVJgpjF9jiIGPEEqYqlDwgyBgEF109VFjmqGmIY/Y4HV4d3Gp2irVfcrp" crossorigin="anonymous">
</head>

<body class="body">
<div class="login-page">
  <div class="form">

    <form action="/index">
      <lottie-player src="https://assets4.lottiefiles.com/datafiles/XRVoUu3IX4sGWtiC3MPpFnJvZNq7lVWDCa8LSqgS/profile.json"  background="transparent"  speed="1"  style="justify-content: center;" loop  autoplay></lottie-player>
      <input type="text" placeholder="&#xf007;  username"/>
      <input type="password" id="password" placeholder="&#xf023;  password"/>
      <i class="fas fa-eye" onclick="show()"></i>
      <br>
      <br>
      <button type="submit" onclick="index()">LOGIN</button>
      <p class="message"></p>
    </form>
  </div>
</div>

  <script>
    function show(){
      var password = document.getElementById("password");
      var icon = document.querySelector(".fas")

      // ========== Checking type of password ==========
      if(password.type === "password"){
        password.type = "text";
      }
      else {
        password.type = "password";
```

```
      }
    };

  </script>
</body>
</html>
```

DASHBOARD

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/bootstrap/styles.css')}}">
    <link
href="https://cdn.datatables.net/1.10.20/css/dataTables.bootstrap4.min.css"
rel="stylesheet" crossorigin="anonymous" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.13.0/js/all.min.js" crossorigin="anonymous"></script>
    {% block title%}{% endblock %}
</head>
<body class="sb-nav-fixed">
    <nav class="sb-topnav navbar navbar-expand navbar-dark bg-dark">
        <button class="btn btn-link btn-sm order-1 order-lg-0"
id="sidebarToggle" href="#"><i class="fas fa-bars"></i></button>
        <a class="navbar-brand" href="/index">Inventory Managment App</a>

    </nav>
    <div id="layoutSidenav">
        <div id="layoutSidenav_nav">
            <nav class="sb-sidenav accordion sb-sidenav-dark"
id="sidenavAccordion">
                <div class="sb-sidenav-menu">
                    <div class="nav">
                        <div class="sb-sidenav-menu-heading">Core</div>
                        <a class="nav-link" href="/index">
                            <div class="sb-nav-link-icon"><i class="fas fa-
tachometer-alt"></i></div>
                            Dashboard
                        </a>
                        <div class="sb-sidenav-menu-heading">Sections</div>
                        <a class="nav-link" href="/products/">
                            <div class="sb-nav-link-icon"><i class="fas fa-
table"></i></div>
                            Products
```

```
                        </a>
                        <a class="nav-link" href="/locations/">
                            <div class="sb-nav-link-icon"><i class="fas fa-
table"></i></div>
                            Locations
                        </a>
                        <a class="nav-link" href="/movements/">
                            <div class="sb-nav-link-icon"><i class="fas fa-
chart-area"></i></div>
                            Movements
                        </a>
                        <a class="nav-link" href="/product-balance/">
                            <div class="sb-nav-link-icon"><i class="fas fa-
table"></i></div>
                            Product Balance Report
                        </a>
                        <a class="nav-link" href="/">
                            <div class="sb-nav-link-icon"><i style="font-
size:18px" class="fa">&#xf08b;</i>
                            </div>
                           Log Out
                        </a>
                    </div>
                </div>
            </nav>
        </div>
        <div id="layoutSidenav_content">
            {% block content %}
            {% endblock %}
        </div>
    </div>
            <script src="https://code.jquery.com/jquery-3.5.1.min.js"
crossorigin="anonymous"></script>
    <script src="{{ url_for('static',
filename='js/bootstrap/scripts.js')}}"></script>
    <script
src="https://cdn.datatables.net/1.10.20/js/jquery.dataTables.min.js"
crossorigin="anonymous"></script>
    <script
src="https://cdn.datatables.net/1.10.20/js/dataTables.bootstrap4.min.js"
crossorigin="anonymous"></script>
    <script src="{{ url_for('static', filename='js/datatables-
demo.js')}}"></script>
    <script src="{{ url_for('static', filename='js/script.js')}}"></script>
</body>
</html>
```

PRODUCT

```
{% extends 'base.html' %}

{% block title %}
<title>Inventory Managmnet App</title>
{% endblock %}


{% block content %}
<main>
    <div class="container-fluid">
        <h1 class="mt-4">Dashboard</h1>
        <ol class="breadcrumb mb-4">
            <li class="breadcrumb-item"><a href="/products">Dashboard</a></li>
            <li class="breadcrumb-item active">Products</li>
        </ol>
        <div class="card mb-4">
            <div class="card-header">Products</div>
            <div class="card-body">
                <div class="card mb-4">
                    <div class="card-header">New Product</div>
                    <div class="card-body">
                        <form action="/products/" method="POST"
id="product_form">
                            <label for="product_name" class="col-form-
label">Product Name</label>
                            <input type="text" name="product_name"
id="product_name">
                            <input id="submitProduct" value="Add Product"
class="btn btn-primary">
                        </form>
                    </div>
                </div>
                <div class="card mb-4">
                <div class="card-header">
                    <i class="fas fa-table mr-1"></i>
                    Products Table
                </div>
                <div class="card-body">
                    <div class="table-responsive">
                        {% if products|length < 1 %}
                        <h4>There are no Products, add one above</h4>
                        {% else %}
                        <table class="table table-bordered" id="dataTable"
width="100%" cellspacing="0">
                            <thead>
                                <tr>
                                    <th>Product Name</th>
```

```
                                    <th>Date</th>
                                    <th>Actions</th>
                                </tr>
                            </thead>
                            <tfoot>
                                <tr>
                                    <th>Product Name</th>
                                    <th>Date</th>
                                    <th>Actions</th>
                                </tr>
                            </tfoot>
                            <tbody>
                                {% for product in products %}
                                <tr>
                                    <td>{{ product.product_id }}</td>
                                    <td>{{ product.date_created.date() }}</td>
                                    <td>
                                        <a href="/delete-product/{{
product.product_id }}">Delete</a>
                                        <br>
                                        <a href="/update-product/{{
product.product_id }}">Update</a>
                                    </td>
                                </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                        {% endif %}
                    </div>
                </div>
                    </div>
                </div>
            </div>
        </div>
</main>
{% endblock %}
```

LOCATION

```
{% extends 'base.html' %}

{% block title %}
<title>Inventory Managmnet App</title>
{% endblock %}


{% block content %}
<main>
```

```
    <div class="container-fluid">
        <h1 class="mt-4">Dashboard</h1>
        <ol class="breadcrumb mb-4">
            <li class="breadcrumb-item"><a href="/">Dashboard</a></li>
            <li class="breadcrumb-item active">Locations</li>
        </ol>
        <div class="card mb-4">
            <div class="card-header">Locations</div>
            <div class="card-body">
                <div class="card mb-4">
                    <div class="card-header">New Location</div>
                    <div class="card-body">
                        <form action="/locations/" method="POST"
id="location_form">
                            <label for="location_name" class="col-form-
label">Location Name</label>
                            <input type="text" name="location_name"
id="location_name">
                            <input id="submitLocation" value="Add Location"
class="btn btn-primary">
                        </form>
                    </div>
                </div>
                <div class="card mb-4">
                <div class="card-header">
                    <i class="fas fa-table mr-1"></i>
                    Locations Table
                </div>
                <div class="card-body">
                    <div class="table-responsive">
                        {% if locations|length < 1 %}
                        <h4>There are no Locations, add one above</h4>
                        {% else %}
                        <table class="table table-bordered" id="dataTable"
width="100%" cellspacing="0">
                            <thead>
                                <tr>
                                    <th>Location Name</th>
                                    <th>Date</th>
                                    <th>Actions</th>
                                </tr>
                            </thead>
                            <tfoot>
                                <tr>
                                    <th>Location Name</th>
                                    <th>Date</th>
                                    <th>Actions</th>
                                </tr>
```

```
                            </tfoot>
                            <tbody>
                                {% for location in locations %}
                                <tr>
                                    <td>{{ location.location_id }}</td>
                                    <td>{{ location.date_created.date()
}}</td>
                                    <td>
                                        <br>
                                        <a href="/update-location/{{
location.location_id }}">Update</a>
                                    </td>
                                </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                        {% endif %}
                    </div>
                </div>
                    </div>
                </div>
            </div>
        </div>
</main>
{% endblock %}
```

8. TESTING

The purpose of software testing is to access or evaluate the capabilities or attributes of a software program's ability to adequately meet the applicable standards and application need. Testing does not ensure quality and the purpose of testing is not to find bugs. Testing can be verification and validation or reliability estimation. The primary objective if testing includes:

- To identifying defects in the application.
- The most important role of testing is simply to provide information to check the proper working of the application while inserting updating and deleting the entry of the products.

8.1 TYPE OF TESTING

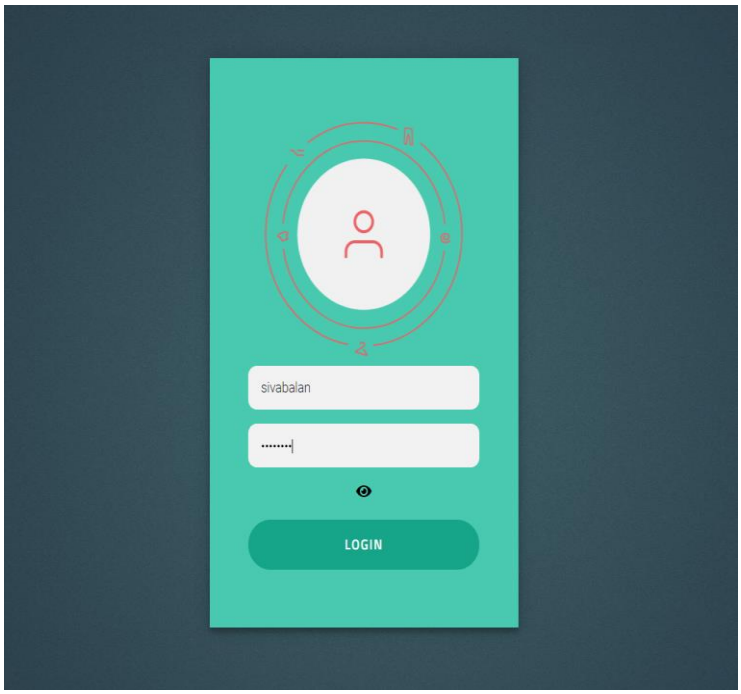
We have used one type of testing to ensure the error free features of our software application:

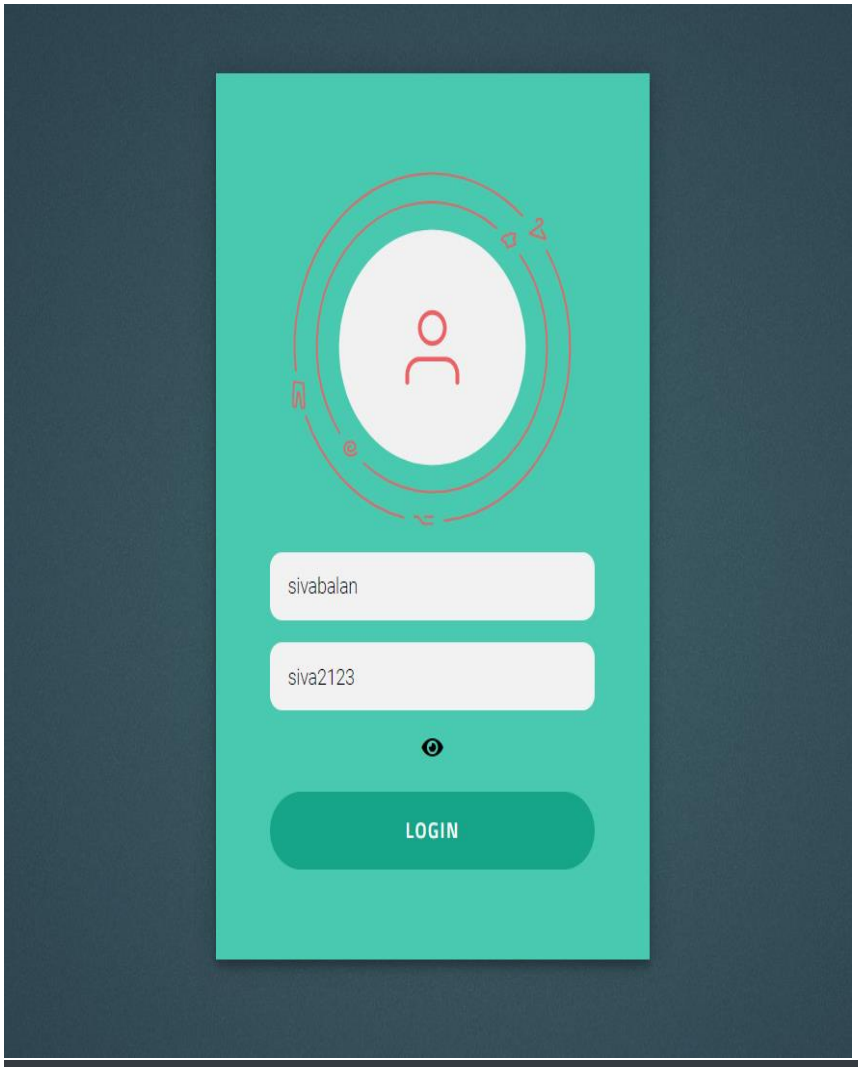
8.2 USER ACCEPTANCE TESTING

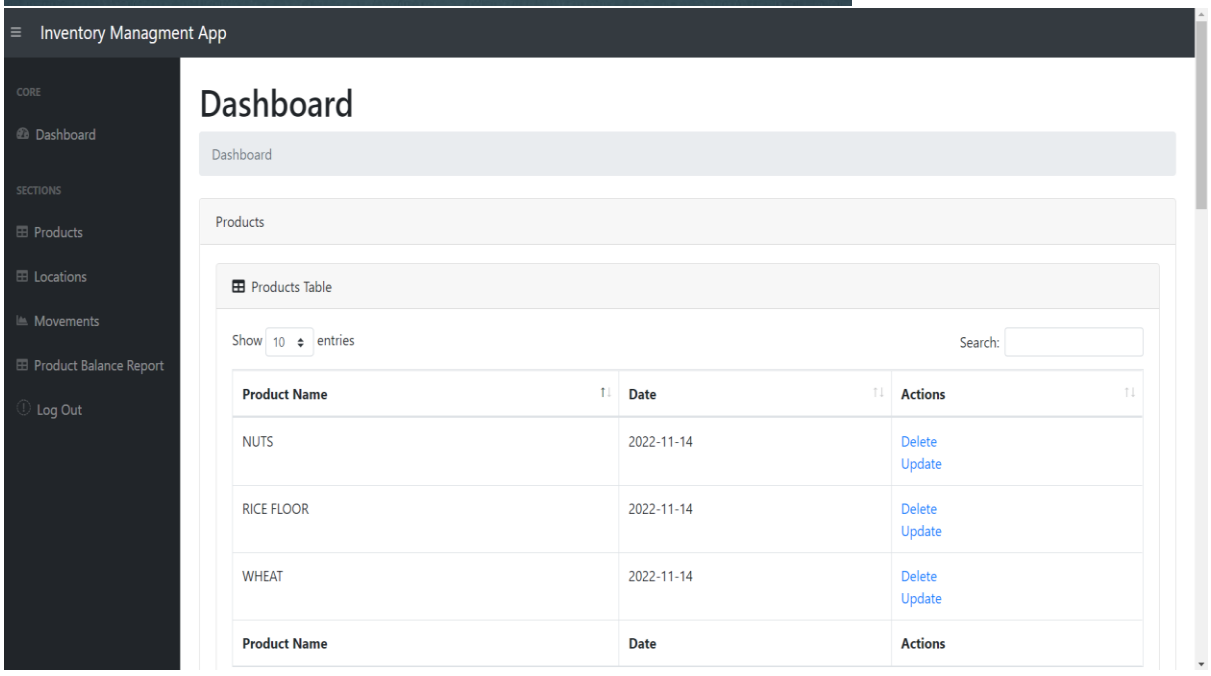
This type of testing is the testing of individual software components. It is typically done by the programmer and not by the testers. It requires details information and knowledge about the internal program design and code to perform this. During unit testing, we carried out various testing task such as the reflection of the unit data on database and its interface. Various types of bugs associated with the component were identified and fixed. We use various functional keys to test our software. In our software unit testing is concerned with the stock units, opening stock units and product units validation as well as the validation of product units.

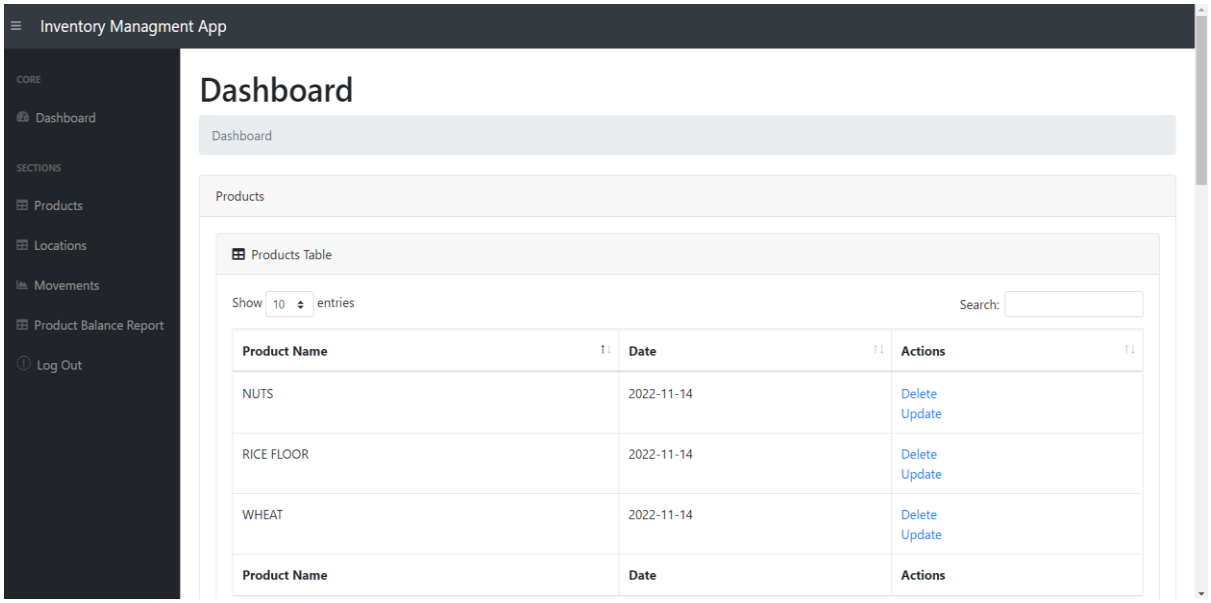
9. RESULTS:

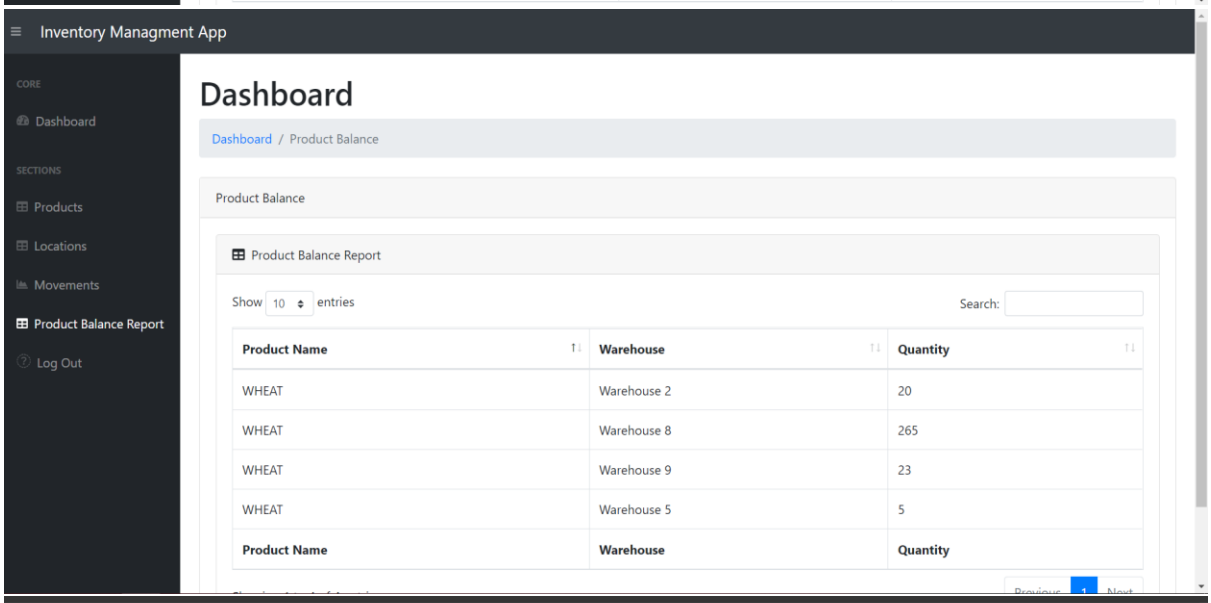
OUTPUT

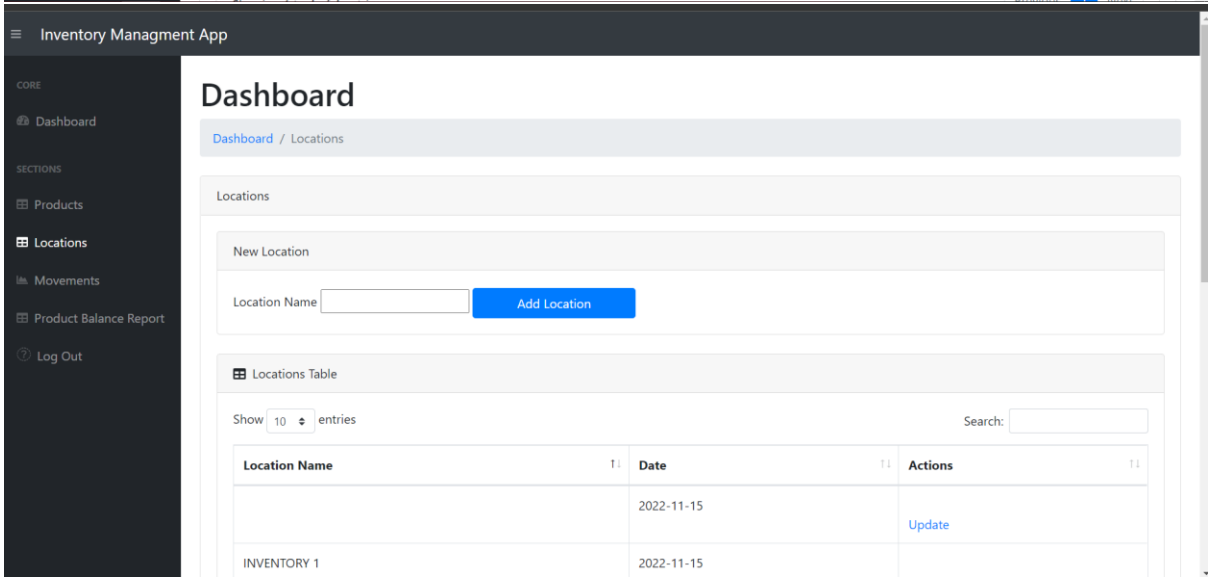


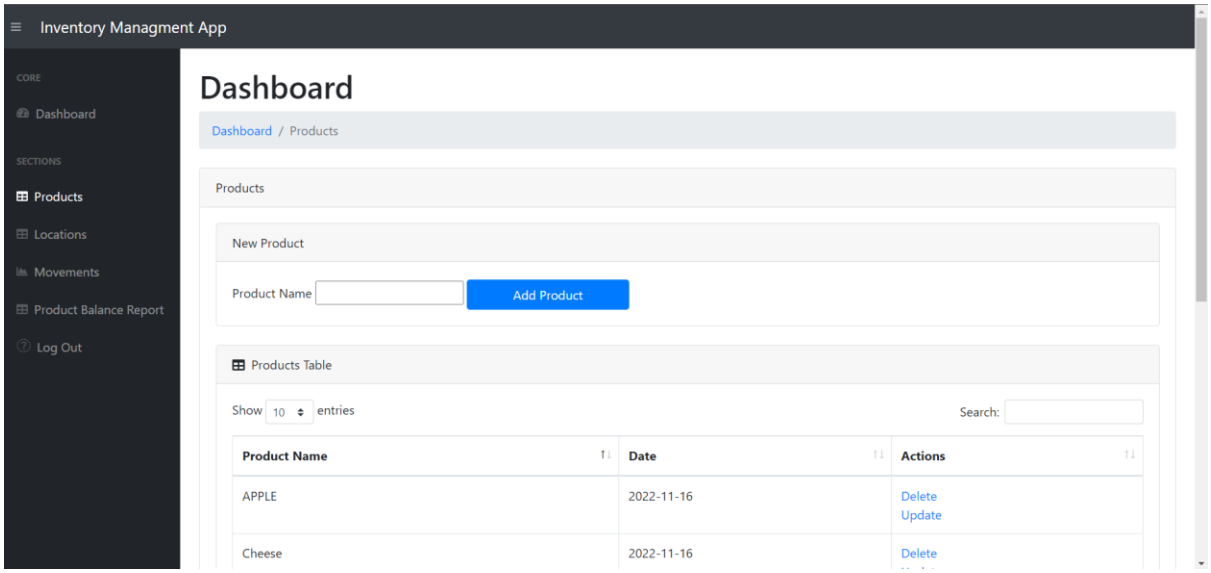

sivabalan
siva2123
LOGIN

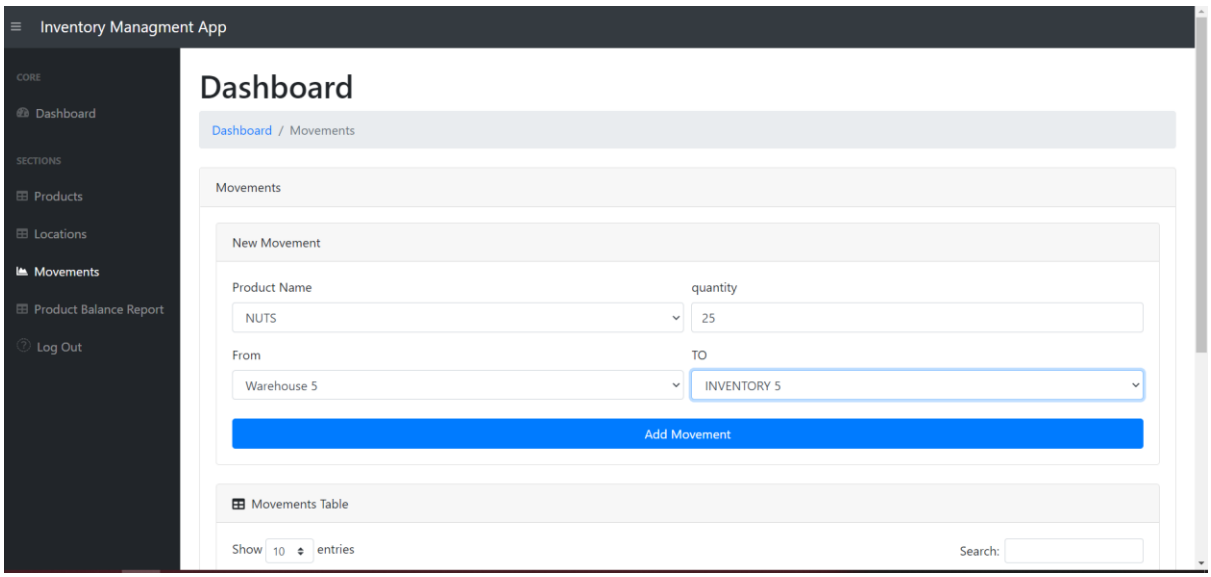

Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard
Products
Products Table
Show 10 entries
Search:
Product Name
Date
Actions
NUTS
2022-11-14
Delete
Update
RICE FLOOR
2022-11-14
Delete
Update
WHEAT
2022-11-14
Delete
Update
Product Name
Date
Actions


Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard
Products
Products Table
Show 10 entries
Search:
Product Name
Date
Actions
NUTS
2022-11-14
Delete
Update
RICE FLOOR
2022-11-14
Delete
Update
WHEAT
2022-11-14
Delete
Update
Product Name
Date
Actions


Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard / Product Balance
Product Balance
Product Balance Report
Show 10 entries
Search:
Product Name
Warehouse
Quantity
WHEAT
Warehouse 2
20
WHEAT
Warehouse 8
265
WHEAT
Warehouse 9
23
WHEAT
Warehouse 5
5
Product Name
Warehouse
Quantity


Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard / Locations
Locations
New Location
Location Name
Add Location
Locations Table
Show 10 entries
Search:
Location Name
Date
Actions
2022-11-15
Update
INVENTORY 1
2022-11-15


Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard / Products
Products
New Product
Product Name
Add Product
Products Table
Show 10 entries
Search:
Product Name
Date
Actions
APPLE
2022-11-16
Delete
Update
Cheese
2022-11-16
Delete


Inventory Managment App
CORE
Dashboard
SECTIONS
Products
Locations
Movements
Product Balance Report
Log Out
Dashboard
Dashboard / Movements
Movements
New Movement
Product Name
quantity
NUTS
25
From
TO
Warehouse 5
INVENTORY 5
Add Movement
Movements Table
Show 10 entries
Search:

10. ADVANTAGES AND DISADVANTAGES

ADVANTAGES OF INVENTORY MANAGEMENT

1. It helps to maintain the right amount of stocks: It seeks to maintain an equilibrium point where your inventory is working at a maximum efficiency and you do not have to have many stocks or too few stocks at hand at any particular point in time.

2.Increased information transparency: a good inventory management helps to keep the flow of information transparent.

3. It leads to a more organized warehouse: with the aid of a good inventory management system, you can easily organize your warehouse.

4. It saves time and money: an effective inventory management system can translate to time and money saved on the part of the business.

5. Improves efficiency and productivity: inventory management devices like bar code scanners and inventory management software can help to greatly increase the efficiency and productivity of a business.

6. Schedule maintenance: once you get hold of a new appliance, you can begin to schedule routine and preventative maintenance, issue work order to your staff and track that the maintenance was actually carried out.

7. Flexibility: a good inventory management strategy will allow the manager to be flexible and adapt to situations as they arise.

DISADVANTAGE:

System crash

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.

Malicious Hacks

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.

Reduced Physical Audits

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.

11. CONCLUSION

To conclude, Inventory Management System is a simple desktop based application basically suitable for small organization. It has every basic items which are used for the small organization. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also provides a simple report on daily basis to know the daily sales and purchase details. This application matches for small organization where there small limited if godowns. Through it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organization.

12. FUTURE SCOPE:

The scope of an inventory system can cover many needs, including valuing the inventory, measuring the change in inventory and planning for future inventory levels. The value of the inventory at the end of each period provides a basis for financial reporting on the balance sheet. Measuring the change in inventory allows the company to determine the cost of inventory sold during the period. This allows the company to plan for future inventory needs.

13. APPENDIX

SOURCE CODE :

```
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 login():
    try:
        return render_template("login.html")
    except:
        return "error"


@app.route('/index', 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>")
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()

        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)
```

GITHUB LINK :

https://github.com/IBM-EPBL/IBM-Project-25022-1659952430

DEMO VIDEO LINK:

https://drive.google.com/file/d/1S_cBZ2_NfjlblzlK6MUvWADK4PzB5xln/view?usp=sharing