Date	19- November 2022
Team ID	PNT2022TMID47816
Project Name	Inventory Management System for Retailers
Batch No	B8-2A4E

PROJECT DEVELOPMENT PHASE - SPRINT 4

ManageSales.html

```
<html>
   <head>
      <meta charset="utf-8">
      <title>MyFlaskApp</title>
      <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
   </head>
   <body>
      {% include 'includes/_navbar.html' %}
      <div class="container mt-4">
         {% include 'includes/_messages.html' %}
         {% block body %}{% endblock %}
      </div>
      <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"></script>
   </body>
</html>
```

Addsales.html

```
<html>
   <head>
      <meta charset="utf-8">
```

```
    <title>MyFlaskApp</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
  </head>
  <body>
    {% include 'includes/_navbar.html' %}
    <div class="container mt-4">
      {% include 'includes/_messages.html' %}
      {% block body %}{% endblock %}
    </div>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"></script>
  </body>
</html>
```

edit_product.html

```
{% extends 'layout.html' %}
{% block body %}
<h1>Edit Product</h1>
{% from "includes/_formhelpers.html" import render_field %}
<form action="" method="POST">
  <div class="form-group">
    {{ render_field(form.product_id, class_="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.product_cost, class_="form-control") }}
  </div>
  <div class="form-group">
    {{ render_field(form.product_num, class_="form-control") }}
```

```
    </div>
    <p><input type="submit" value="Update" class="btn btn-primary"></p>
</form>
{% endblock %}
```

product_movement.html

```
{% extends 'layout.html' %}
{% block body %}
    <h1>Product Movements</h1>
    <a class="btn btn-success" href="/add_product_movements">Add Product Movements</a>
    <hr>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Movement ID</th>
                <th>Time</th>
                <th>From Location</th>
                <th>To Location</th>
                <th>Product ID</th>
                <th>Quantity</th>
            </tr>
        </thead>
        <tbody>
                {% for movement in movements %}
               <tr>
                <td>{{movement.MOVEMENT_ID}}</td>
                <td>{{movement.TIME}}</td>
                <td>{{movement.FROM_LOCATION}}</td>
```

```
            <td>{{movement.TO_LOCATION}}</td>
            <td>{{movement.PRODUCT_ID}}</td>
<td>{{movement.QTY}}</td>
            <!--<td><a href="edit_product_movement/{{movement.MOVEMENT_ID}}"
class="btn btn-primary pull-right">Edit</a></td>-->
            <td>
               <form action="{{url_for('delete_product_movements',
id=movement.MOVEMENT_ID)}}" method="POST">
                  <input type="hidden" name="method" value="DELETE">
                  <input type="submit" value="Delete" class="btn btn-danger">
               </form>
            </td>
           </tr>
           {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

app.py

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from flask_mysqldb import MySQL
from wtforms import Form, StringField, TextAreaField, PasswordField,
validators, SelectField, IntegerField import ibm_db from passlib.hash import
sha256_crypt from functools import wraps import win32api from sendgrid
import * #creating an app instance app = Flask(__name__)
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=;PORT=;SECURITY=SSL;S
SL
ServerCertificate=DigiCertGlobalRootCA.crt;UID=;PWD=;","","")
#Index @app.route('/') def index():
return render_template('home.html')
```

```
#Products
@app.route('/products')
def products():
    sql = "SELECT * FROM
products"     stmt =
ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)
products=[]     row =
ibm_db.fetch_assoc(stmt)
while(row):
        products.append(row)         row
= ibm_db.fetch_assoc(stmt)
products=tuple(products)
#print(products)     if result>0:
        return render_template('products.html', products = products)
    else:
        msg='No products found'         return
render_template('products.html', msg=msg)
#Locations
@app.route('/locations')
def locations():
    sql = "SELECT * FROM locations"
stmt = ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)
locations=[]

    row = ibm_db.fetch_assoc(stmt)
while(row):
```

```
        locations.append(row)
row = ibm_db.fetch_assoc(stmt)
locations=tuple(locations)
#print(locations)     if result>0:
        return render_template('locations.html', locations = locations)
    else:
        msg='No locations found'         return
render_template('locations.html', msg=msg)
#Product Movements
@app.route('/product_movements')
def product_movements():
    sql = "SELECT * FROM
productmovements"     stmt =
ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)     movements=[]
row = ibm_db.fetch_assoc(stmt)     while(row):
        movements.append(row)
row = ibm_db.fetch_assoc(stmt)
movements=tuple(movements)
#print(movements)
    if result>0:
        return render_template('product_movements.html', movements = movements)
    else:
        msg='No product movements found'         return
render_template('product_movements.html', msg=msg) #Register
Form Class class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
username = StringField('Username', [validators.Length(min=1, max=25)])
email = StringField('Email', [validators.length(min=6, max=50)])
```

```
password = PasswordField('Password', [ validators.DataRequired(),
validators.EqualTo('confirm', message='Passwords do not match')
   ])
   confirm = PasswordField('Confirm Password')
#user register
@app.route('/register', methods=['GET','POST'])
def register():
   form = RegisterForm(request.form)     if
request.method == 'POST' and form.validate():
      name = form.name.data        email = form.email.data        username =
form.username.data        password =
sha256_crypt.encrypt(str(form.password.data))     sql1="INSERT INTO
users(name, email, username, password) VALUES(?,?,?,?)"        stmt1 =
ibm_db.prepare(conn, sql1)        ibm_db.bind_param(stmt1,1,name)
ibm_db.bind_param(stmt1,2,email)        ibm_db.bind_param(stmt1,3,username)
ibm_db.bind_param(stmt1,4,password)        ibm_db.execute(stmt1)
      #for flash messages taking parameter and the category of message to be flashed
flash("You are now registered and can log in", "success")
        #when registration is successful redirect to home
      return redirect(url_for('login'))     return
render_template('register.html', form = form)
#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():     if request.method == 'POST':
#Get form fields        username =
request.form['username']        password_candidate
= request.form['password']     sql1="Select * from
users where username = ?"        stmt1 =
ibm_db.prepare(conn, sql1)
```

```
ibm_db.bind_param(stmt1,1,username)
result=ibm_db.execute(stmt1)
d=ibm_db.fetch_assoc(stmt1)
      if result > 0:
#Get the stored hash
data = d
          password  =  data['PASSWORD']                #compare
passwords          if sha256_crypt.verify(password_candidate,
password):
             #Passed
session['logged_in'] = True
session['username'] = username
flash("you are now logged in","success")
return redirect(url_for('dashboard'))
          else:
             error = 'Invalid Login'               return
render_template('login.html', error=error)
          #Close connection
          cur.close()
else:
          error = 'Username not found'               return
render_template('login.html',  error=error)         return
render_template('login.html')
#check if user logged in def
is_logged_in(f):     @wraps(f)
def wrap(*args, **kwargs):
if 'logged_in' in session:
return f(*args, **kwargs)
      else:
```

```
        flash('Unauthorized, Please
login','danger')          return
redirect(url_for('login'))    return wrap
#Logout
@app.route('/logout') @is_logged_in def
logout():    session.clear()    flash("You are
now logged out", "success")    return
redirect(url_for('login'))
#Dashboard
@app.route('/dashboard'
) @is_logged_in def
dashboard():
   sql2="SELECT product_id, location_id, qty FROM
product_balance"    sql3="SELECT location_id FROM locations"
stmt2 = ibm_db.prepare(conn, sql2)

   stmt3 = ibm_db.prepare(conn,
sql3)
result=ibm_db.execute(stmt2)
ibm_db.execute(stmt3)
products=[]    row =
ibm_db.fetch_assoc(stmt2)
while(row):
      products.append(row)       row
= ibm_db.fetch_assoc(stmt2)
products=tuple(products)
locations=[]    row2 =
ibm_db.fetch_assoc(stmt3)
while(row2):
```

```
        locations.append(row2)
row2 = ibm_db.fetch_assoc(stmt3)
locations=tuple(locations)     locs = []
for i in locations:
        locs.append(list(i.values())[0])
if result>0:
        return render_template('dashboard.html', products = products, locations = locs)
    else:
        msg='No products found'         return
render_template('dashboard.html', msg=msg)
#Product Form Class
class
ProductForm(Form):
    product_id = StringField('Product ID', [validators.Length(min=1, max=200)])
product_cost = StringField('Product Cost', [validators.Length(min=1, max=200)])
product_num = StringField('Product Num', [validators.Length(min=1, max=200)])
#Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
    form = ProductForm(request.form)     if
request.method == 'POST' and form.validate():
        product_id = form.product_id.data         product_cost = form.product_cost.data
product_num = form.product_num.data       sql1="INSERT INTO products(product_id,
product_cost, product_num) VALUES(?,?,?)"         stmt1 = ibm_db.prepare(conn, sql1)
ibm_db.bind_param(stmt1,1,product_id)         ibm_db.bind_param(stmt1,2,product_cost)
ibm_db.bind_param(stmt1,3,product_num)           ibm_db.execute(stmt1)
flash("Product Added", "success")    return redirect(url_for('products'))     return
render_template('add_product.html', form=form)
```

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

 #populate product form fields     form.product_id.data =
product['PRODUCT_ID']     form.product_cost.data =
str(product['PRODUCT_COST'])     form.product_num.data
= str(product['PRODUCT_NUM'])     if request.method ==
'POST' and form.validate():         product_id =
request.form['product_id']         product_cost =
request.form['product_cost']         product_num =
request.form['product_num']
       sql2="UPDATE products SET product_id=?,product_cost=?,product_num=?
WHERE product_id=?"         stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,product_id)         ibm_db.bind_param(stmt2,2,product_cost)
ibm_db.bind_param(stmt2,3,product_num)         ibm_db.bind_param(stmt2,4,id)
ibm_db.execute(stmt2)         flash("Product Updated", "success")         return
redirect(url_for('products'))     return render_template('edit_product.html', form=form)
#Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in def
delete_product(id):
```

```
    sql2="DELETE FROM products WHERE product_id=?"
    stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)
ibm_db.execute(stmt2)
flash("Product   Deleted",   "success")
return       redirect(url_for('products'))
#Location      Form      Class      class
LocationForm(Form):
    location_id = StringField('Location ID', [validators.Length(min=1, max=200)])
#Add Location
@app.route('/add_location', methods=['GET', 'POST'])
@is_logged_in
def
add_location():
    form = LocationForm(request.form)     if
request.method == 'POST' and form.validate():
location_id = form.location_id.data         sql2="INSERT
into locations VALUES(?)"         stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,location_id)
ibm_db.execute(stmt2)         flash("Location Added",
"success")         return redirect(url_for('locations'))
return render_template('add_location.html', form=form)
#Edit Location
@app.route('/edit_location/<string:id>', methods=['GET', 'POST'])
@is_logged_in def
edit_location(id):
        sql2="SELECT * FROM locations where location_id = ?"
stmt2 = ibm_db.prepare(conn, sql2)
```

```
ibm_db.bind_param(stmt2,1,id)
result=ibm_db.execute(stmt2)
location=ibm_db.fetch_assoc(stmt2)
   #Get form     form =
LocationForm(request.form)
print(location)

#populate article form fields     form.location_id.data
= location['LOCATION_ID']     if request.method
== 'POST' and form.validate():
       location_id = request.form['location_id']         sql2="UPDATE
locations SET location_id=? WHERE location_id=?"         stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,location_id)
ibm_db.bind_param(stmt2,2,id)         ibm_db.execute(stmt2)
flash("Location Updated", "success")         return
redirect(url_for('locations'))     return
render_template('edit_location.html', form=form)
#Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in def
delete_location(id):
   sql2="DELETE FROM locations WHERE
location_id=?"     stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)     ibm_db.execute(stmt2)
flash("Location Deleted", "success")     return
redirect(url_for('locations'))
#Product Movement Form Class
class ProductMovementForm(Form):
```

```
    from_location = SelectField('From Location',
choices=[])     to_location = SelectField('To Location',
choices=[])     product_id = SelectField('Product ID',
choices=[])     qty = IntegerField('Quantity') class
CustomError(Exception):

    pass
#Add Product Movement
@app.route('/add_product_movements', methods=['GET', 'POST'])
@is_logged_in def
add_product_movements():
    form =
ProductMovementForm(request.form)
sql2="SELECT product_id FROM products"
sql3="SELECT location_id FROM locations"
stmt2 = ibm_db.prepare(conn, sql2)     stmt3 =
ibm_db.prepare(conn, sql3)
result=ibm_db.execute(stmt2)
ibm_db.execute(stmt3)     products=[]     row =
ibm_db.fetch_assoc(stmt2)     while(row):
      products.append(row)       row =
ibm_db.fetch_assoc(stmt2)
products=tuple(products)
locations=[]     row2 =
ibm_db.fetch_assoc(stmt3)
while(row2):
        locations.append(row2)
row2 = ibm_db.fetch_assoc(stmt3)
```

```
locations=tuple(locations)     prods = []
for p in products:

prods.append(list(p.values())[0])
locs = []     for i in locations:
locs.append(list(i.values())[0])
form.from_location.choices = [(l,l)
for l in locs]
form.from_location.choices.append(
("Main Inventory","Main
Inventory"))
form.to_location.choices = [(l,l) for l
in locs]
form.to_location.choices.append(("
Main Inventory","Main Inventory"))
form.product_id.choices = [(p,p) for
p in prods]     if request.method ==
'POST' and form.validate():
       from_location =
form.from_location.data         to_location =
form.to_location.data         product_id =
form.product_id.data         qty =
form.qty.data         if
from_location==to_location:
           raise CustomError("Please Give different From and To Locations!!")         elif
from_location=="Main Inventory":             sql2="SELECT * from product_balance
where location_id=? and product_id=?"             stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,to_location)
ibm_db.bind_param(stmt2,2,product_id)             result=ibm_db.execute(stmt2)
```

```
result=ibm_db.fetch_assoc(stmt2)          print("-----------------")          print(result)
print("-----------------")          app.logger.info(result)          if result!=False:
if(len(result))>0:
                Quantity = result["QTY"]
q = Quantity + qty
sql2="UPDATE product_balance set
qty=? where location_id=? and
product_id=?"

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

```
ibm_db.bind_param(stmt2,3,product_id)  ibm_db.bind_param(stmt2,4,qty)
ibm_db.execute(stmt2)

        sql = "select product_num from products where product_id=?"
stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,product_id)
current_num=ibm_db.execute(stmt)  current_num =
ibm_db.fetch_assoc(stmt)  sql2="Update products set
product_num=? where product_id=?"  stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-qty)
ibm_db.bind_param(stmt2,2,product_id)  ibm_db.execute(stmt2)
alert_num=current_num['PRODUCT_NUM']-qty
      if(alert_num<=0):
        alert("Please update the quantity of the product {}, Atleast {} number of
pieces must be added to finish the pending Product Movements!".format(product_id,-
alert_num))  elif to_location=="Main Inventory":  sql2="SELECT *
from product_balance where location_id=? and product_id=?"  stmt2 =
ibm_db.prepare(conn, sql2)  ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,product_id)  result=ibm_db.execute(stmt2)
result=ibm_db.fetch_assoc(stmt2)  app.logger.info(result)  if result!=False:
if(len(result))>0:
        Quantity = result["QTY"]
q = Quantity - qty
        sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
        stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)  ibm_db.execute(stmt2)
```

```
sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)" stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id) ibm_db.bind_param(stmt2,4,qty)
ibm_db.execute(stmt2) flash("Product Movement Added", "success")
sql = "select product_num from products where product_id=?" stmt =
ibm_db.prepare(conn, sql) ibm_db.bind_param(stmt,1,product_id)
current_num=ibm_db.execute(stmt) current_num = ibm_db.fetch_assoc(stmt)
sql2="Update products set product_num=? where product_id=?" stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']+qty)
ibm_db.bind_param(stmt2,2,product_id) ibm_db.execute(stmt2)
alert_num=q if(alert_num<=0):
    alert("Please Add {} number of {} to {}
warehouse!".format(q,product_id,from_location))
  else:
    raise CustomError("There is no product named {} in
{}.".format(product_id,from_location)) else: #will be executed if both
from_location and to_location are specified f=0

  sql = "SELECT * from product_balance where location_id=? and
product_id=?" stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,from_location)
ibm_db.bind_param(stmt,2,product_id) result=ibm_db.execute(stmt)
result = ibm_db.fetch_assoc(stmt) if result!=False:
    if(len(result))>0:
      Quantity = result["QTY"]
q = Quantity - qty
      sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
```

```
                stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,from_location)
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.execute(stmt2)
                f=1
alert_num=q
if(alert_num<=0):
                    alert("Please Add {} number of {} to {}
warehouse!".format(q,product_id,from_location))
        else:
            raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))
        if(f==1):
            sql = "SELECT * from product_balance where location_id=? and
product_id=?"               stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,to_location)


ibm_db.bind_param(stmt,2,product_id)
result=ibm_db.execute(stmt)               result =
ibm_db.fetch_assoc(stmt)               if
result!=False:                  if(len(result))>0:
                    Quantity = result["QTY"]
q = Quantity + qty
                    sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,to_location)
```

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

   flash("Product Movement Added", "success")
render_template('products.html',form=form)        return
redirect(url_for('product_movements'))     return
render_template('add_product_movements.html', form=form)
#Delete Product Movements
@app.route('/delete_product_movements/<string:id>', methods=['POST'])
@is_logged_in def
delete_product_movements(id):
   sql2="DELETE FROM productmovements WHERE movement_id=?"
   stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)
ibm_db.execute(stmt2)     flash("Product
Movement Deleted", "success")     return
redirect(url_for('product_movements')) if
```

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

config.py

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from flask_mysqldb import MySQL
from wtforms import Form, StringField, TextAreaField, PasswordField,
validators, SelectField, IntegerField import ibm_db from passlib.hash import
sha256_crypt from functools import wraps import win32api
from sendgrid import *
#creating an app instance
app = Flask(__name__)
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=;PORT=;SECURITY=SSL;S
SL
ServerCertificate=DigiCertGlobalRootCA.crt;UID=;PWD=;","","")
#Index @app.route('/') def index():
return render_template('home.html')
#Products
@app.route('/products')
def products():
   sql = "SELECT * FROM
products"     stmt =
ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)
products=[]     row =
```

```
ibm_db.fetch_assoc(stmt)
while(row):
        products.append(row)         row
= ibm_db.fetch_assoc(stmt)
products=tuple(products)
#print(products)     if result>0:
        return render_template('products.html', products = products)
    else:
        msg='No products found'         return
render_template('products.html', msg=msg)
#Locations
@app.route('/locations')
def locations():
    sql = "SELECT * FROM
locations"     stmt =
ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)
locations=[]     row =
ibm_db.fetch_assoc(stmt)
while(row):
        locations.append(row)
row = ibm_db.fetch_assoc(stmt)
locations=tuple(locations)
#print(locations)     if result>0:
        return render_template('locations.html', locations = locations)
    else:
        msg='No locations found'         return
render_template('locations.html', msg=msg)
```

```
#Product Movements
@app.route('/product_movements')
def product_movements():
    sql = "SELECT * FROM
productmovements"     stmt =
ibm_db.prepare(conn, sql)
result=ibm_db.execute(stmt)     movements=[]
row = ibm_db.fetch_assoc(stmt)     while(row):
        movements.append(row)
row = ibm_db.fetch_assoc(stmt)
movements=tuple(movements)
    #print(movements)
    if result>0:
        return render_template('product_movements.html', movements = movements)
    else:
        msg='No product movements found'         return
render_template('product_movements.html', msg=msg)
#Register Form Class
class
RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
username = StringField('Username', [validators.Length(min=1, max=25)])
email = StringField('Email', [validators.length(min=6, max=50)])
password = PasswordField('Password', [         validators.DataRequired(),
validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')
#user register
```

```
@app.route('/register', methods=['GET','POST'])
def register():
   form = RegisterForm(request.form)     if
request.method == 'POST' and form.validate():
       name = form.name.data         email = form.email.data         username =
form.username.data         password = sha256_crypt.encrypt(str(form.password.data))
sql1="INSERT INTO users(name, email, username, password) VALUES(?,?,?,?)"
stmt1 = ibm_db.prepare(conn, sql1)         ibm_db.bind_param(stmt1,1,name)
ibm_db.bind_param(stmt1,2,email)         ibm_db.bind_param(stmt1,3,username)

       ibm_db.bind_param(stmt1,4,password)
ibm_db.execute(stmt1)
       #for flash messages taking parameter and the category of message to be
flashed         flash("You are now registered and can log in", "success")
#when registration is successful redirect to home         return
redirect(url_for('login'))     return render_template('register.html', form = form)
#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():
   if request.method == 'POST':         #Get form
fields         username = request.form['username']
password_candidate = request.form['password']
sql1="Select * from users where username = ?"
stmt1 = ibm_db.prepare(conn, sql1)
ibm_db.bind_param(stmt1,1,username)
result=ibm_db.execute(stmt1)
d=ibm_db.fetch_assoc(stmt1)
       if result > 0:
```

```
        #Get the stored hash
data = d
        password = data['PASSWORD']
        #compare passwords            if
sha256_crypt.verify(password_candidate, password):
            #Passed                session['logged_in']
= True                session['username'] = username
flash("you are now logged in","success")
return redirect(url_for('dashboard'))

        else:
            error = 'Invalid Login'                return
render_template('login.html', error=error)
        #Close
connection
cur.close()        else:
        error = 'Username not found'            return
render_template('login.html', error=error)     return
render_template('login.html') #check if user logged in
def is_logged_in(f):     @wraps(f)     def wrap(*args,
**kwargs):         if 'logged_in' in session:
return f(*args, **kwargs)
       else:
        flash('Unauthorized, Please
login','danger')            return
redirect(url_for('login'))     return wrap
#Logout
@app.route('/logout') @is_logged_in def
logout():     session.clear()     flash("You are
```

```
now logged out", "success")     return
redirect(url_for('login'))
#Dashboard
@app.route('/dashboard')
@is_logged_in
def dashboard():
   sql2="SELECT product_id, location_id, qty FROM
product_balance"     sql3="SELECT location_id FROM locations"
stmt2 = ibm_db.prepare(conn, sql2)     stmt3 = ibm_db.prepare(conn,
sql3)     result=ibm_db.execute(stmt2)     ibm_db.execute(stmt3)
products=[]     row = ibm_db.fetch_assoc(stmt2)     while(row):
       products.append(row)         row
= ibm_db.fetch_assoc(stmt2)
products=tuple(products)
locations=[]     row2 =
ibm_db.fetch_assoc(stmt3)
while(row2):
       locations.append(row2)
row2 = ibm_db.fetch_assoc(stmt3)
locations=tuple(locations)     locs = []
for i in locations:
       locs.append(list(i.values())[0])
if result>0:
       return render_template('dashboard.html', products = products, locations = locs)
   else:
       msg='No products found'         return
render_template('dashboard.html', msg=msg)
#Product Form Class
class ProductForm(Form):
```

```
   product_id = StringField('Product ID', [validators.Length(min=1, max=200)])
product_cost = StringField('Product Cost', [validators.Length(min=1, max=200)])
product_num = StringField('Product Num', [validators.Length(min=1, max=200)])
#Add Product
@app.route('/add_product', methods=['GET', 'POST'])
@is_logged_in
def add_product():
   form = ProductForm(request.form)     if
request.method == 'POST' and form.validate():
       product_id = form.product_id.data
product_cost = form.product_cost.data
product_num = form.product_num.data
       sql1="INSERT INTO products(product_id, product_cost, product_num)
VALUES(?,?,?)"
       stmt1 = ibm_db.prepare(conn, sql1)
ibm_db.bind_param(stmt1,1,product_id)
ibm_db.bind_param(stmt1,2,product_cost)
ibm_db.bind_param(stmt1,3,product_num)
ibm_db.execute(stmt1)        flash("Product Added",
"success")        return redirect(url_for('products'))
return render_template('add_product.html', form=form)
#Edit Product
@app.route('/edit_product/<string:id>', methods=['GET', 'POST'])
@is_logged_in def
edit_product(id):
   sql1="Select * from products where product_id = ?"
stmt1 = ibm_db.prepare(conn, sql1)
ibm_db.bind_param(stmt1,1,id)
```

```
result=ibm_db.execute(stmt1)
product=ibm_db.fetch_assoc(stmt1)
      print(product)     #Get form     form =
ProductForm(request.form) #populate product form fields
form.product_id.data = product['PRODUCT_ID']
form.product_cost.data = str(product['PRODUCT_COST'])
form.product_num.data = str(product['PRODUCT_NUM'])
if request.method == 'POST' and form.validate():
product_id = request.form['product_id']        product_cost =
request.form['product_cost']        product_num =
request.form['product_num']
      sql2="UPDATE products SET product_id=?,product_cost=?,product_num=?
WHERE product_id=?"        stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,product_id)        ibm_db.bind_param(stmt2,2,product_cost)
ibm_db.bind_param(stmt2,3,product_num)        ibm_db.bind_param(stmt2,4,id)
ibm_db.execute(stmt2)        flash("Product Updated", "success")        return
redirect(url_for('products'))     return render_template('edit_product.html', form=form)
#Delete Product
@app.route('/delete_product/<string:id>', methods=['POST'])
@is_logged_in def
delete_product(id):
sql2="DELETE
FROM products
WHERE
product_id=?"

   stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)
ibm_db.execute(stmt2)
```

```
flash("Product   Deleted",   "success")
return redirect(url_for('products'))
#Location Form Class
class
LocationForm(Form):
   location_id = StringField('Location ID', [validators.Length(min=1, max=200)])
#Add Location
@app.route('/add_location', methods=['GET', 'POST'])
@is_logged_in
def
add_location():
   form = LocationForm(request.form)     if
request.method == 'POST' and form.validate():
location_id = form.location_id.data         sql2="INSERT
into locations VALUES(?)"         stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,location_id)
ibm_db.execute(stmt2)         flash("Location Added",
"success")         return redirect(url_for('locations'))
return render_template('add_location.html', form=form)
#Edit Location
@app.route('/edit_location/<string:id>', methods=['GET', 'POST'])
@is_logged_in def
edit_location(id):
       sql2="SELECT * FROM locations where location_id = ?"
stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)     result=ibm_db.execute(stmt2)
location=ibm_db.fetch_assoc(stmt2)
```

```
    #Get form     form =
LocationForm(request.form)
print(location)
    #populate article form fields     form.location_id.data =
location['LOCATION_ID'] if request.method == 'POST' and
form.validate():         location_id = request.form['location_id']
sql2="UPDATE locations SET location_id=? WHERE location_id=?"
stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,location_id)
ibm_db.bind_param(stmt2,2,id)         ibm_db.execute(stmt2)
flash("Location Updated", "success")         return
redirect(url_for('locations'))     return
render_template('edit_location.html', form=form)
#Delete Location
@app.route('/delete_location/<string:id>', methods=['POST'])
@is_logged_in def
delete_location(id):
    sql2="DELETE FROM locations WHERE
location_id=?"     stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)     ibm_db.execute(stmt2)
flash("Location Deleted", "success")     return
redirect(url_for('locations'))
#Product Movement Form Class
class ProductMovementForm(Form):
    from_location = SelectField('From Location',
choices=[])     to_location = SelectField('To Location',
choices=[])     product_id = SelectField('Product ID',
choices=[])     qty = IntegerField('Quantity') class
CustomError(Exception):
```

```
    pass
#Add Product Movement
@app.route('/add_product_movements', methods=['GET', 'POST'])
@is_logged_in def
add_product_movements():
    form =
ProductMovementForm(request.form)
sql2="SELECT product_id FROM products"
sql3="SELECT location_id FROM locations"
stmt2 = ibm_db.prepare(conn, sql2)     stmt3 =
ibm_db.prepare(conn, sql3)
result=ibm_db.execute(stmt2)
ibm_db.execute(stmt3)     products=[]     row =
ibm_db.fetch_assoc(stmt2)     while(row):
        products.append(row)         row
= ibm_db.fetch_assoc(stmt2)
products=tuple(products)
locations=[]     row2 =
ibm_db.fetch_assoc(stmt3)
while(row2):
        locations.append(row2)
row2 = ibm_db.fetch_assoc(stmt3)
locations=tuple(locations)     prods = []
for p in products:

prods.append(list(p.values())[0])
locs = []     for i in locations:
        locs.append(list(i.values())[0]) form.from_location.choices = [(l,l) for l
in locs]     form.from_location.choices.append(("Main Inventory","Main
```

```
Inventory"))     form.to_location.choices = [(l,l) for l in locs]
form.to_location.choices.append(("Main Inventory","Main Inventory"))
form.product_id.choices = [(p,p) for p in prods]     if request.method ==
'POST' and form.validate():
       from_location =
form.from_location.data         to_location =
form.to_location.data         product_id =
form.product_id.data         qty =
form.qty.data         if
from_location==to_location:
           raise CustomError("Please Give different From and To Locations!!")


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

           print("------------------")
app.logger.info(result)
if result!=False:
if(len(result))>0:
                   Quantity = result["QTY"]
q = Quantity + qty
                   sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                   stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,to_location)
```

```
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.execute(stmt2)
                sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"                    stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)                    ibm_db.bind_param(stmt2,4,qty)
ibm_db.execute(stmt2)
          else:
             sql2="INSERT into product_balance(product_id, location_id, qty) values(?, ?,
?)"                stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,product_id)
ibm_db.bind_param(stmt2,2,to_location)               ibm_db.bind_param(stmt2,3,qty)
ibm_db.execute(stmt2)
             sql2="INSERT into productmovements(from_location, to_location, product_id,
qty) VALUES(?, ?, ?, ?)"
             stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.bind_param(stmt2,4,qty)                ibm_db.execute(stmt2)
sql = "select product_num from products where product_id=?"
stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,product_id)
current_num=ibm_db.execute(stmt)            current_num =
ibm_db.fetch_assoc(stmt)        sql2="Update products set
product_num=? where product_id=?"            stmt2 =
ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,current_num['PRODUCT_NUM']-qty)
```

```
ibm_db.bind_param(stmt2,2,product_id)  ibm_db.execute(stmt2)
alert_num=current_num['PRODUCT_NUM']-qty
    if(alert_num<=0):
        alert("Please update the quantity of the product {}, Atleast {} number of pieces must be added to finish the pending Product Movements!".format(product_id,-alert_num))  elif to_location=="Main Inventory":  sql2="SELECT * from product_balance where location_id=? and product_id=?"  stmt2 = ibm_db.prepare(conn, sql2)  ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,product_id)  result=ibm_db.execute(stmt2)
result=ibm_db.fetch_assoc(stmt2)  app.logger.info(result)  if result!=False:
        if(len(result))>0:
            Quantity = result["QTY"]
q = Quantity - qty
            sql2="UPDATE product_balance set qty=? where location_id=? and product_id=?"
            stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.execute(stmt2)
            sql2="INSERT into productmovements(from_location, to_location, product_id, qty) VALUES(?, ?, ?, ?)"  stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)  ibm_db.bind_param(stmt2,4,qty)
ibm_db.execute(stmt2)  flash("Product Movement Added", "success")
sql = "select product_num from products where product_id=?"  stmt = ibm_db.prepare(conn, sql)  ibm_db.bind_param(stmt,1,product_id)
current_num=ibm_db.execute(stmt)  current_num = ibm_db.fetch_assoc(stmt)
```

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

            if(alert_num<=0):
                alert("Please Add {} number of {} to {}
warehouse!".format(q,product_id,from_location))
        else:
            raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))                else: #will be executed if both
from_location and to_location are specified
        f=0
        sql = "SELECT * from product_balance where location_id=? and
product_id=?"          stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,from_location)
ibm_db.bind_param(stmt,2,product_id)          result=ibm_db.execute(stmt)
result = ibm_db.fetch_assoc(stmt) if result!=False:
            if(len(result))>0:
                Quantity = result["QTY"]
q = Quantity - qty
                sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,from_location)
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.execute(stmt2)
```

```
                f=1
alert_num=q
if(alert_num<=0):
                    alert("Please Add {} number of {} to {}
warehouse!".format(q,product_id,from_location))
          else:
              raise CustomError("There is no product named {} in
{}.".format(product_id,from_location))
          if(f==1):
              sql = "SELECT * from product_balance where location_id=? and
product_id=?"               stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,to_location)
ibm_db.bind_param(stmt,2,product_id)              result=ibm_db.execute(stmt)
result = ibm_db.fetch_assoc(stmt)              if result!=False:
if(len(result))>0:
                    Quantity = result["QTY"]
q = Quantity + qty
                    sql2="UPDATE product_balance set qty=? where location_id=? and
product_id=?"
                    stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,q)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)
ibm_db.execute(stmt2)
              else:
                        sql2="INSERT into product_balance(product_id, location_id,
qty) values(?, ?, ?)"                  stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,product_id)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,qty)                  ibm_db.execute(stmt2)
```

```
            sql2="INSERT into productmovements(from_location, to_location,
product_id, qty) VALUES(?, ?, ?, ?)"                stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,from_location)
ibm_db.bind_param(stmt2,2,to_location)
ibm_db.bind_param(stmt2,3,product_id)                ibm_db.bind_param(stmt2,4,qty)
ibm_db.execute(stmt2)     flash("Product Movement Added", "success")
render_template('products.html',form=form)         return
redirect(url_for('product_movements'))     return
render_template('add_product_movements.html', form=form)
#Delete Product Movements
@app.route('/delete_product_movements/<string:id>', methods=['POST'])
@is_logged_in def
delete_product_movements(id):
   sql2="DELETE FROM productmovements WHERE movement_id=?"
   stmt2 = ibm_db.prepare(conn, sql2)
ibm_db.bind_param(stmt2,1,id)
ibm_db.execute(stmt2)

   flash("Product Movement Deleted",
"success")     return
redirect(url_for('product_movements')) if
__name__ == '__main__':     app.secret_key =
"secret123"
   #when the debug mode is on, we do not need to restart the server again and again
app.run(debug=True)
```