Name: Sarthak Shandilya

Case study 6 (Ecommerce)

Submitted to: Karthika

Hexaware Technologies

Schema Design:

Create database Ecommerce.

```
mysql> create database if not exists Ecommerce_App
    -> ;
Query OK, 1 row affected (0.01 sec)
```

1. customers table:
   - customer_id (Primary Key)
   - name
   - email
   - password

```
mysql> use Ecommerce_App;
Database changed
mysql> create table customers(
    -> customer_id int primary key auto_increment,
    -> name varchar(40),
    -> email varchar(50),
    -> password varchar(30));
Query OK, 0 rows affected (0.08 sec)
```

2. products table:
   - product_id (Primary Key)
   - name
   - price
   - description
   - stockQuantity

```
mysql> create table products(
    -> product_id int primary key auto_increment,
    -> name varchar(30),
    -> price float,
    -> category varchar(20),
    -> stockquantity int);
Query OK, 0 rows affected (0.07 sec)
```

3. cart table:
   - cart_id (Primary Key)
   - customer_id (Foreign Key)
   - product_id (Foreign Key)
   - quantity

```
mysql> create table cart(
    -> cart_id int primary key auto_increment,
    -> customer_id int,
    -> product_id int,
    -> quantity int,
    -> foreign key(customer_id) references customers(customer_id) on update cascade on delete cascade,
    -> foreign key(product_id) references products(product_id) on update cascade on delete cascade);
Query OK, 0 rows affected (0.11 sec)
```

4. orders table:
   - order_id (Primary Key)
   - customer_id (Foreign Key)
   - order_date
   - total_price
   - shipping_address

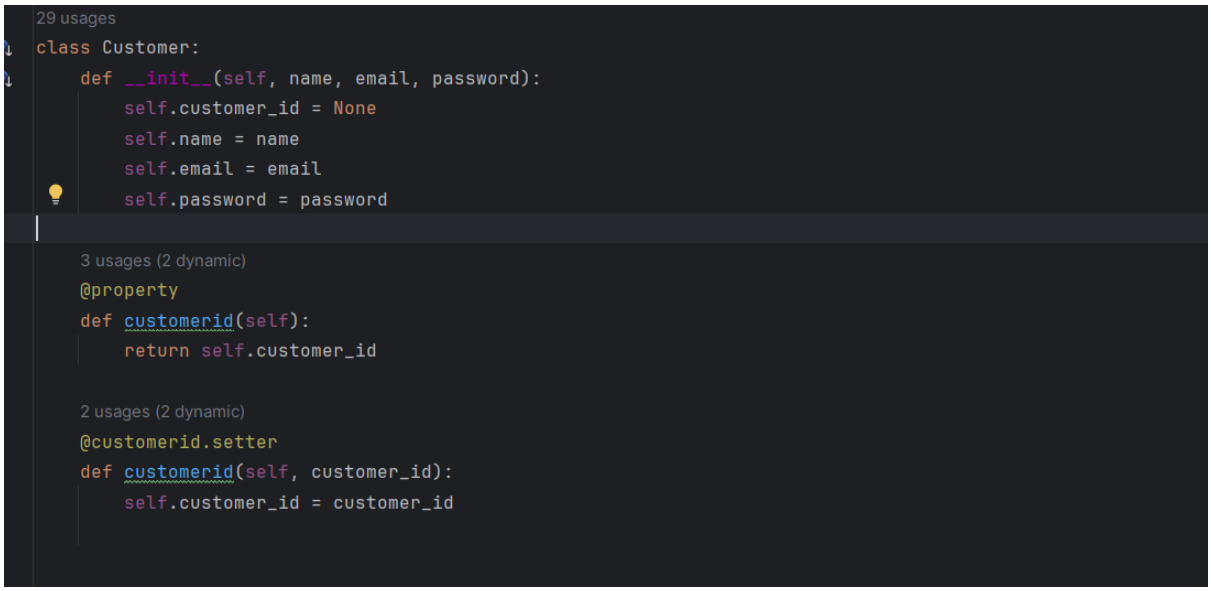
```
mysql> create table orders(
    -> order_id int primary key auto_increment,
    -> customer_id int,
    -> orderdate date,
    -> total_price float,
    -> shipping_address varchar(80),
    -> foreign key(customer_id) references customers(customer_id) on update cascade on delete cascade);
Query OK, 0 rows affected (0.14 sec)
```

5. order_items table (to store order details):
    - order_item_id (Primary Key)
    - order_id (Foreign Key)
    - product_id (Foreign Key)
    - quantity

```
mysql> create table order_items(
    -> order_item_id int primary key auto_increment,
    -> order_id int,
    -> product_id int,
    -> quantity int,
    -> foreign key(order_id) references orders(order_id) on delete cascade on update cascade,
    -> foreign key(product_id) references products(product_id) on delete cascade on update cascade);
Query OK, 0 rows affected (0.17 sec)
```

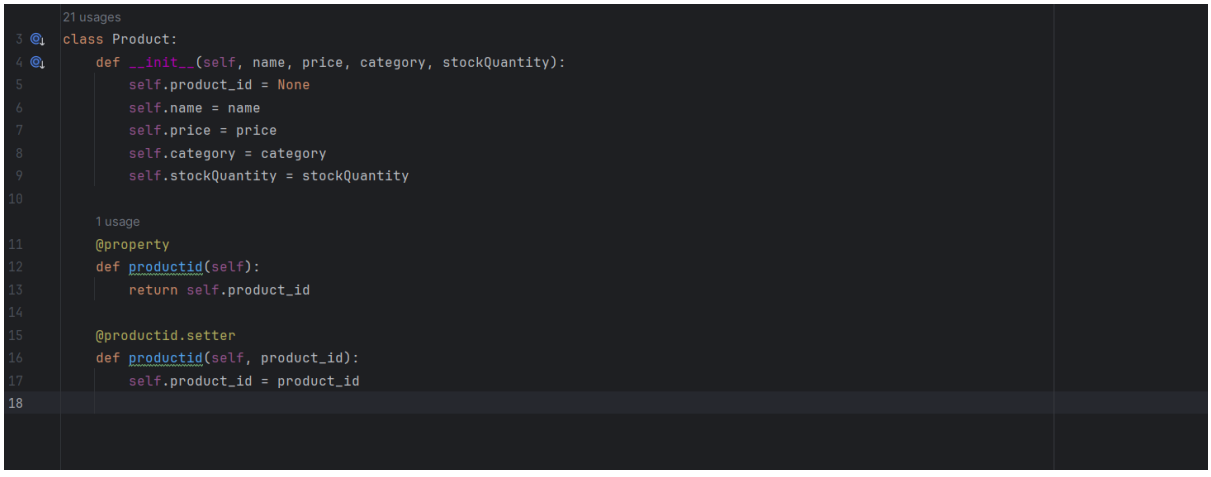
Classes:

Customer:



```
29 usages
class Customer:
    def __init__(self, name, email, password):
        self.customer_id = None
        self.name = name
        self.email = email
        self.password = password

    3 usages (2 dynamic)
    @property
    def customerid(self):
        return self.customer_id

    2 usages (2 dynamic)
    @customerid.setter
    def customerid(self, customer_id):
        self.customer_id = customer_id
```

Products:



```
21 usages
class Product:
    def __init__(self, name, price, category, stockQuantity):
        self.product_id = None
        self.name = name
        self.price = price
        self.category = category
        self.stockQuantity = stockQuantity

    1 usage
    @property
    def productid(self):
        return self.product_id

    @productid.setter
    def productid(self, product_id):
        self.product_id = product_id
```

Orders:

```python
from entity.Customers import Customer


2 usages
class Order(Customer):
    def __init__(self, customer, order_date, total_price, shipping_address):
        self.order_id = None
        self.customer_id = customer.customerid
        self.order_date = order_date
        self.total_price = total_price
        self.address = shipping_address

    1 usage
    @property
    def orderid(self):
        return self.order_id

    @orderid.setter
    def orderid(self, order_id):
        self.order_id = order_id
```

OrderItems:

```python
from Orders import Order
from Products import Product


class OrderItem(Order, Product):
    def __init__(self, order, product, quantity):
        self.order_item_id = None
        self.order_id = order.order_id
        self.product_id = product.product_id
        self.quantity = quantity

    1 usage
    @property
    def orderItemId(self):
        return self.order_item_id

    @orderItemId.setter
    def orderItemId(self, order_item_id):
        self.order_item_id = order_item_id
```

Cart:

```python
from entity.Customers import Customer
from Products import Product


class Cart(Customer, Product):
    def __init__(self, customer, product, quantity):
        self.cart_id = None
        self.customer_id = customer.customerid
        self.product_id = product.product_id
        self.quantity = quantity

    1 usage
    @property
    def cartid(self):
        return self.cart_id

    @cartid.setter
    def cartid(self, cart_id):
        self.cart_id = cart_id
```

Service Provider Interface/Abstract class:

Keep the interfaces and implementation classes in package dao

• Define an OrderProcessorRepository interface/abstract class with methods for adding/removing products to/from the cart and placing orders.

The following methods will interact with database.

1. createProduct()

parameter: Product product return type: boolean

2. createCustomer()

parameter: Customer customer

return type: boolean

3. deleteProduct() parameter: productId return type: boolean

4. deleteCustomer(customerId) parameter: customerId return type: boolean

5. addToCart(): insert the product in cart. parameter: Customer customer, Product product, int quantity

return type: boolean

6. removeFromCart(): delete the product in cart. parameter: Customer customer, Product product

return type: boolean

7. getAllFromCart(Customer customer): list the product in cart for a customer.

parameter: Customer customer

return type: list of product

8. placeOrder(Customer customer, List<Map>, string shippingAddress): should update order table and orderItems table.

parameter: Customer customer, list of product and quantity

return type: boolean

9. getOrdersByCustomer()

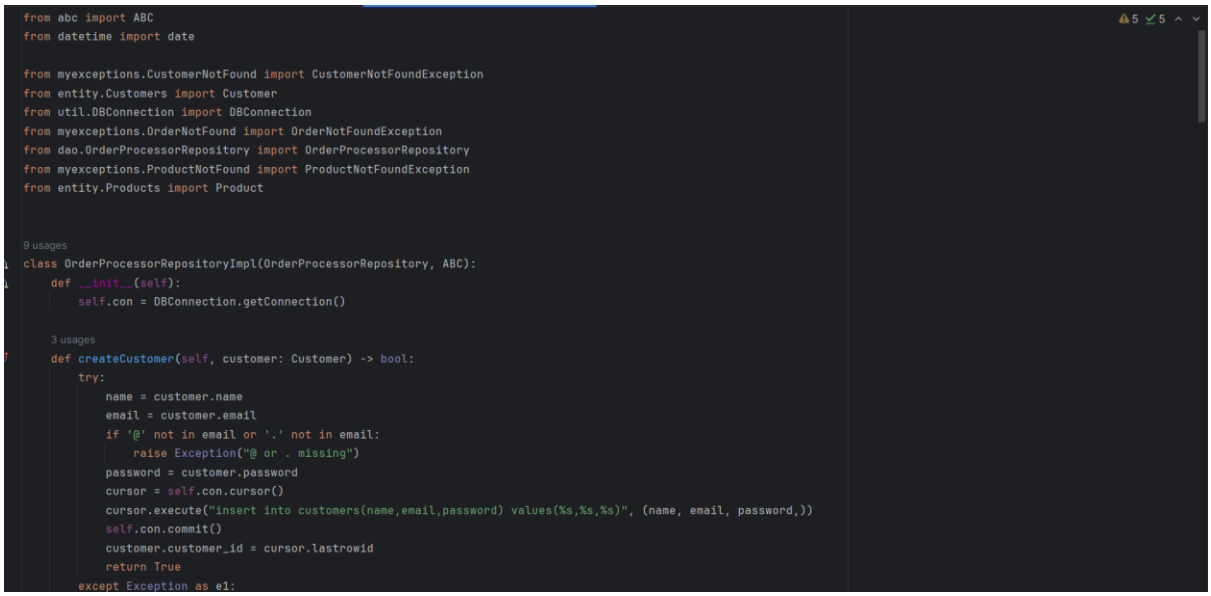
parameter: customerid

return type: list of product and quantity

```python
from abc import ABC,abstractmethod
from entity.Products import Product
from entity.Customers import Customer


2 usages
class OrderProcessorRepository(ABC):
    @abstractmethod
    def createProduct(self, product: Product) -> bool:
        pass

    @abstractmethod
    def createCustomer(self, customer: Customer) -> bool:
        pass

    @abstractmethod
    def deleteProduct(self, product_id) -> bool:
        pass

    @abstractmethod
    def deleteCustomer(self, customer_id) -> bool:
        pass

    @abstractmethod
    def addToCart(self, customer: Customer, product: Product, quantity) -> bool:
        pass

    @abstractmethod
    def removeFromCart(self, customer: Customer, product: Product) -> bool:
        pass
```

```python
        pass

    @abstractmethod
    def removeFromCart(self, customer: Customer, product: Product) -> bool:
        pass

    @abstractmethod
    def get_customer_by_id(self, customer_id):
        pass

    def get_product_by_id(self, product_id):
        pass

    @abstractmethod
    def getAllFromCart(self, customer: Customer) -> list:
        pass

    @abstractmethod
    def placeOrder(self, customer: Customer, product_quantities: list, shipping_address) -> bool:
        pass

    @abstractmethod
    def getOrdersByCustomers(self, customer: Customer) -> list:
        pass
```

Implement the above interface in a class called OrderProcessorRepositoryImpl in package dao.



```python
from abc import ABC
from datetime import date

from myexceptions.CustomerNotFound import CustomerNotFoundException
from entity.Customers import Customer
from util.DBConnection import DBConnection
from myexceptions.OrderNotFound import OrderNotFoundException
from dao.OrderProcessorRepository import OrderProcessorRepository
from myexceptions.ProductNotFound import ProductNotFoundException
from entity.Products import Product


9 usages
class OrderProcessorRepositoryImpl(OrderProcessorRepository, ABC):
    def __init__(self):
        self.con = DBConnection.getConnection()

    3 usages
    def createCustomer(self, customer: Customer) -> bool:
        try:
            name = customer.name
            email = customer.email
            if '@' not in email or '.' not in email:
                raise Exception("@ or . missing")
            password = customer.password
            cursor = self.con.cursor()
            cursor.execute("insert into customers(name,email,password) values(%s,%s,%s)", (name, email, password,))
            self.con.commit()
            customer.customer_id = cursor.lastrowid
            return True
        except Exception as e1:
```

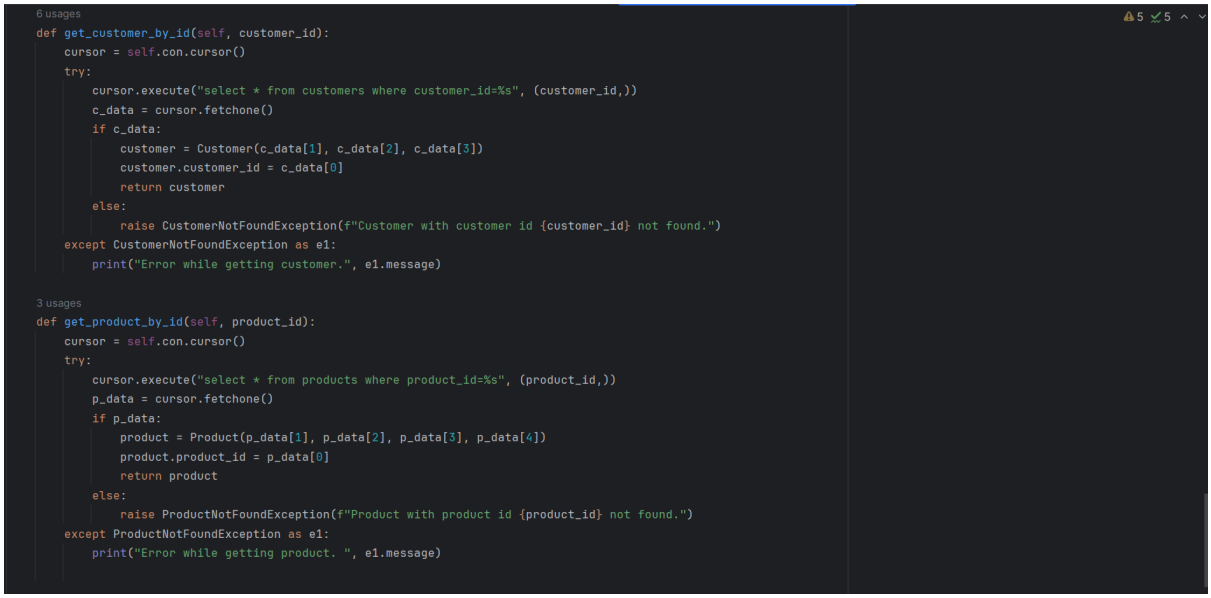
```python
            return True
        except Exception as e1:
            print("Error while registering customer. ", e1)
            return False

    3 usages
    def createProduct(self, product: Product) -> bool:
        name = product.name
        price = product.price
        category = product.category
        quantity = product.stockQuantity
        cursor = self.con.cursor()
        flag = 0
        try:
            query = "insert into products(name,price,category,stockquantity) values(%s,%s,%s,%s)"
            cursor.execute(query, (name, price, category, quantity,))
            self.con.commit()
            product.product_id = cursor.lastrowid
            print(f"Your product id is {product.product_id}. Remember it for future reference")
            flag = 1
            return True
        except Exception as e1:
            print("Error while adding product. ", e1)
            return False
```

```python
    1 usage
    def deleteProduct(self, product_id) -> bool:
        cursor = self.con.cursor()
        try:
            cursor.execute("delete from products where product_id=%s", (product_id,))
            self.con.commit()
            return True
        except Exception as e1:
            print(f"Sorry. We can't find product id {product_id}", e1)
            return False
```

```python
def placeOrder(self, customer: Customer, product_quantities: list, shipping_address) -> bool:
    customer_id = customer.customer_id
    cursor = self.con.cursor()
    try:
        for product, quantity in product_quantities:
            product_id = product.product_id
            date_today = date.today()
            total_price = product.price * quantity
            address = shipping_address
            cursor.execute("select stockquantity from products where product_id = %s", (product_id, ))
            quat = cursor.fetchone()
            if quat[0] >= quantity:
                q_updateOrder = "update products set stockquantity=%s where product_id=%s"
                cursor.execute(q_updateOrder, (quat[0]-quantity, product_id, ))
                self.con.commit()
                q_placeOrder = "insert into orders(customer_id,orderdate,total_price,shipping_address) values(%s,%s,%s,%s)"
                cursor.execute(q_placeOrder, (customer_id, date_today, total_price, address))
                self.con.commit()
                order_id = cursor.lastrowid
                print(f"This order product with product id {product_id} has order id {order_id}. Your total order value is worth {total_price}.")
                q_orderItem = "insert into order_items(order_id, product_id, quantity) values(%s,%s,%s)"
                cursor.execute(q_orderItem, (order_id, product_id, quantity,))
                self.con.commit()
                order_item_id = cursor.lastrowid
                print(f"Your order item id is {order_item_id}")
                return True
            else:
                print("Not enough stock available.")
    except Exception as e1:
        print("Something happened while placing order try again.", e1)
        return False
```

```python
1 usage
def addToCart(self, customer: Customer, product: Product, quantity) -> bool:
    customer_id = customer.customer_id
    product_id = product.product_id
    cursor = self.con.cursor()
    query = "insert into cart(customer_id, product_id, quantity) values(%s,%s,%s)"
    try:
        cursor.execute(query, (customer_id, product_id, quantity,))
        self.con.commit()
        cart_id = cursor.lastrowid
        print(f"Your item is added to cart successfully. Your cart id is {cart_id}")
        return True
    except Exception as e1:
        print("There was some issue adding item to cart.", e1)
        return False
```

```python
1 usage
def getAllFromCart(self, customer: Customer) -> list:
    try:
        cursor = self.con.cursor()
        q = "select products.* from cart join products on cart.product_id=products.product_id where customer_id=%s"
        c_id = customer.customer_id
        cursor.execute(q, (c_id,))
        return cursor.fetchall()
    except Exception as e1:
        print("Problem while fetching cart details. ", e1)

1 usage
def getOrdersByCustomers(self, customer: Customer) -> list:
    customer_id = customer.customer_id
    cursor = self.con.cursor()
    query = ("select products.*,order_items.quantity from order_items join products on "
             "products.product_id=order_items.product_id join orders on orders.order_id=order_items.order_id where "
             "customer_id=%s")
    cursor.execute(query, (customer_id, ))
    product_details = cursor.fetchall()
    try:
        if product_details:
            return product_details
        else:
            raise OrderNotFoundException(f"No orders found for customer id {customer_id}")
    except OrderNotFoundException as o1:
        print("Error while fetching order details", o1.message)
```



```
6 usages
def get_customer_by_id(self, customer_id):
    cursor = self.con.cursor()
    try:
        cursor.execute("select * from customers where customer_id=%s", (customer_id,))
        c_data = cursor.fetchone()
        if c_data:
            customer = Customer(c_data[1], c_data[2], c_data[3])
            customer.customer_id = c_data[0]
            return customer
        else:
            raise CustomerNotFoundException(f"Customer with customer id {customer_id} not found.")
    except CustomerNotFoundException as e1:
        print("Error while getting customer.", e1.message)

3 usages
def get_product_by_id(self, product_id):
    cursor = self.con.cursor()
    try:
        cursor.execute("select * from products where product_id=%s", (product_id,))
        p_data = cursor.fetchone()
        if p_data:
            product = Product(p_data[1], p_data[2], p_data[3], p_data[4])
            product.product_id = p_data[0]
            return product
        else:
            raise ProductNotFoundException(f"Product with product id {product_id} not found.")
    except ProductNotFoundException as e1:
        print("Error while getting product. ", e1.message)
```

These are the implementations of every abstract methods.

Write code to establish a connection to your SQL database.

• Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.

• Connection properties supplied in the connection string should be read from a property file.

• Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```
import mysql
from mysql import connector

from util.PropertyUtil import PropertyUtil


7 usages
class DBConnection:
    con = None

    2 usages
    @staticmethod
    def getConnection():
        if DBConnection.con is None:
            con_string = PropertyUtil.getPropertyString()
            DBConnection.con = mysql.connector.connect(**con_string)
            return DBConnection.con
```

```python
2 usages
class PropertyUtil:
    1 usage
    @staticmethod
    def getPropertyString():
        connection = {
            'host': 'localhost',
            'database': 'ecommerce_app',
            'user': 'root',
            'password': 'root'
        }
        return connection
```

Create the exceptions in package myexceptions and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method

• CustomerNotFoundException: throw this exception when user enters an invalid customer id which doesn't exist in db

```python
9 usages
class CustomerNotFoundException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

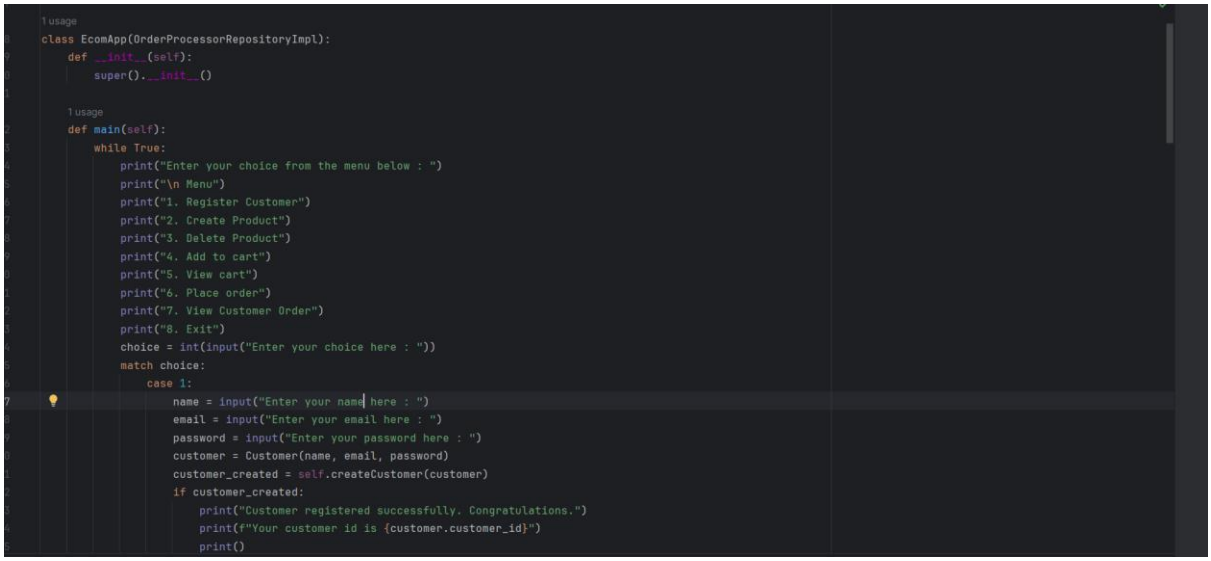
• ProductNotFoundException: throw this exception when user enters an invalid product id which doesn't exist in db

```python
6 usages
class ProductNotFoundException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

• OrderNotFoundException: throw this exception when user enters an invalid order id which doesn't exist in db

```python
3 usages
class OrderNotFoundException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

Create class named EcomApp with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.



```
1 usage
class EcomApp(OrderProcessorRepositoryImpl):
    def __init__(self):
        super().__init__()

    1 usage
    def main(self):
        while True:
            print("Enter your choice from the menu below : ")
            print("\n Menu")
            print("1. Register Customer")
            print("2. Create Product")
            print("3. Delete Product")
            print("4. Add to cart")
            print("5. View cart")
            print("6. Place order")
            print("7. View Customer Order")
            print("8. Exit")
            choice = int(input("Enter your choice here : "))
            match choice:
                case 1:
                    name = input("Enter your name here : ")
                    email = input("Enter your email here : ")
                    password = input("Enter your password here : ")
                    customer = Customer(name, email, password)
                    customer_created = self.createCustomer(customer)
                    if customer_created:
                        print("Customer registered successfully. Congratulations.")
                        print(f"Your customer id is {customer.customer_id}")
                        print()
```

1. Register Customer.

```
match choice:
    case 1:
        name = input("Enter your name here : ")
        email = input("Enter your email here : ")
        password = input("Enter your password here : ")
        customer = Customer(name, email, password)
        customer_created = self.createCustomer(customer)
        if customer_created:
            print("Customer registered successfully. Congratulations.")
            print(f"Your customer id is {customer.customer_id}")
            print()
```

Output:

```
E:\HexawareAssignments\Python\Ecommerce\.venv\Scripts\python.exe E:\HexawareAssignments\Python\Ecommerce\main\EcomApp.py
Enter your choice from the menu below :

 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 1
Enter your name here : Aayushi Singh
Enter your email here : saayushi3@outlook.com
Enter your password here : Aayushi1@
Customer registered successfully. Congratulations.
Your customer id is 9

Enter your choice from the menu below :

 Menu
1. Register Customer
2. Create Product
```

```
mysql> select * from customers;
+-------------+---------------+---------------------------+-------------+
| customer_id | name          | email                     | password    |
+-------------+---------------+---------------------------+-------------+
|           1 | John Doe      | john.doe@yahoo.com        | John1@      |
|           3 | Jenifer       | laurance.jenifer@yahoo.com | Jenifer@123 |
|           4 | Mayank Kumar  | mayank.kumar@yahoo.com    | Mayank@123  |
|           9 | Aayushi Singh | saayushi3@outlook.com     | Aayushi1@   |
+-------------+---------------+---------------------------+-------------+
4 rows in set (0.01 sec)
```

2. Create Product.

```
        case 2:
            name = input("Enter product name : ")
            price = float(input("Enter price of the product here : "))
            category = input("Enter category of product : ")
            quantity = int(input("Enter number of item : "))
            product = Product(name, price, category, quantity)
            product_created = self.createProduct(product)
            if product_created:
                print("Product added successfully. Congratulations.")
                print(f"Your product id is {product.product_id}")
                print()
```

Output:

```
 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 2
Enter product name : iPhone
Enter price of the product here : 48000
Enter category of product : Electornics Gadget
Enter number of item : 15
Your product id is 16. Remember it for future reference
Product added successfully. Congratulations.
Your product id is 16
```

```
mysql> select * from products;
+------------+--------+---------+--------------------+---------------+
| product_id | name   | price   | category           | stockquantity |
+------------+--------+---------+--------------------+---------------+
|          1 | Laptop | 35000.5 | Electronics Gadget |            20 |
|          3 | Laptop | 35000.5 | Electronics Gadget |            17 |
|         16 | iPhone |   48000 | Electornics Gadget |            15 |
+------------+--------+---------+--------------------+---------------+
3 rows in set (0.00 sec)
```

3. Delete Product.

```
    case 3:
        product_id = int(input("Enter your product id here : "))
        deleted_product = self.deleteProduct(product_id)
        if deleted_product:
            print("Product deleted successfully.")
            print()
```

```
Enter your choice from the menu below :

 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 3
Enter your product id here : 15
Product deleted successfully.
```

Output:


```
Enter your choice from the menu below :

 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 3
Enter your product id here : 15
Product deleted successfully.
```


```
mysql> select * from products;
+------------+-----------------+---------+--------------------+---------------+
| product_id | name            | price   | category           | stockquantity |
+------------+-----------------+---------+--------------------+---------------+
|          1 | Laptop          | 35000.5 | Electronics Gadget |            20 |
|          3 | Laptop          | 35000.5 | Electronics Gadget |            17 |
|         15 | Product to test |    2000 | Testing            |            20 |
+------------+-----------------+---------+--------------------+---------------+
3 rows in set (0.00 sec)

mysql> select * from products;
+------------+--------+---------+--------------------+---------------+
| product_id | name   | price   | category           | stockquantity |
+------------+--------+---------+--------------------+---------------+
|          1 | Laptop | 35000.5 | Electronics Gadget |            20 |
|          3 | Laptop | 35000.5 | Electronics Gadget |            17 |
|         16 | iPhone |   48000 | Electornics Gadget |            15 |
+------------+--------+---------+--------------------+---------------+
3 rows in set (0.00 sec)
```

4. Add to cart.

```
            print()
        case 4:
            customer_id = int(input("Enter customer id : "))
            product_id = int(input("Enter product id : "))
            quantity = int(input("Enter number of items : "))
            customer = self.get_customer_by_id(customer_id)
            product = self.get_product_by_id(product_id)
            if product and customer:
                self.addToCart(customer, product, quantity)
                print("Product added to cart successfully.")
            else:
                print("Product or customer not found.")
            print()
```

Output :

```
 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 4
Enter customer id : 3
Enter product id : 16
Enter number of items : 2
Your item is added to cart successfully. Your cart id is 6
Product added to cart successfully.
```

```
mysql> select * from cart;
+---------+-------------+------------+----------+
| cart_id | customer_id | product_id | quantity |
+---------+-------------+------------+----------+
|       5 |           1 |          1 |        3 |
|       6 |           3 |         16 |        2 |
+---------+-------------+------------+----------+
2 rows in set (0.01 sec)
```

5. View cart.

```
    print()
case 5:
    customer_id = int(input("Enter your customer id : "))
    customer = self.get_customer_by_id(customer_id)
    try:
        if customer:
            cart_items = self.getAllFromCart(customer)
            for cart in cart_items:
                print("Cart Id : ", cart[0])
                print("Customer Id : ", cart[1])
                print("Product Id : ", cart[2])
                print("Quantity : ", cart[3])
                print("------------------")
        else:
            raise CustomerNotFoundException("Customer not found.")
    except CustomerNotFoundException as e1:
        print("Error. ", e1)
```

Output :

```
Enter your choice from the menu below :
----- Menu -----
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 5
Enter your customer id : 3
Cart Id :  16
Customer Id :  iPhone
Product Id :  48000.0
Quantity :  Electornics Gadget
------------------
```

```
mysql> select * from cart;
+---------+-------------+------------+----------+
| cart_id | customer_id | product_id | quantity |
+---------+-------------+------------+----------+
|       5 |           1 |          1 |        3 |
|       6 |           3 |         16 |        2 |
+---------+-------------+------------+----------+
2 rows in set (0.01 sec)
```

6. Place order.

```
case 6:
    customer_id = int(input("Enter your customer id : "))
    customer = self.get_customer_by_id(customer_id)
    products_quantities = []
    while True:
        product_id = int(input("Please enter product_id(0 to stop) : "))
        if product_id == 0:
            break
        quantity = int(input("Enter the quantity for this item : "))
        product = self.get_product_by_id(product_id)
        try:
            if product:
                products_quantities.append((product, quantity))
            else:
                raise ProductNotFoundException(f"Product not found for product id {product_id}")
        except ProductNotFoundException as p1:
            print("Error while getting product.", p1)
    shipping_address = input("Enter your address where you want the orders : ")
    orders_placed = self.placeOrder(customer, products_quantities, shipping_address)
    if orders_placed:
        print("Orders placed successfully! ")
        print("We're heading you to main menu.")
        print()
    else:
        print("Error while placing order.")
        print()
```

Output:

```
Enter your choice from the menu below :

 Menu
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 6
Enter your customer id : 9
Please enter product_id(0 to stop) : 16
Enter the quantity for this item : 2
Please enter product_id(0 to stop) : 0
Enter your address where you want the orders : Pune, Hinjewadi
This order product with product id 16 has order id 3. Your total order value is worth 96000.0.
Your order item id is 3
Orders placed successfully!
We're heading you to main menu.
```

```
mysql> select * from orders;
+----------+-------------+------------+-------------+--------------------------+
| order_id | customer_id | orderdate  | total_price | shipping_address         |
+----------+-------------+------------+-------------+--------------------------+
|        1 |           3 | 2024-02-05 |       70001 | New york city, USA       |
|        2 |           1 | 2024-02-07 |      105002 | RDS college, muzaffarpur |
|        3 |           9 | 2024-02-07 |       96000 | Pune, Hinjewadi          |
+----------+-------------+------------+-------------+--------------------------+
3 rows in set (0.00 sec)
```

7. View Customer Order

```
    case 7:
        customer_id = int(input("Enter your customer id : "))
        customer = self.get_customer_by_id(customer_id)
        try:
            if customer:
                products_quantities = self.getOrdersByCustomers(customer)
                for product_quantity in products_quantities:
                    print("Product id : ", product_quantity[0])
                    print("Product Name : ", product_quantity[1])
                    print("Price : ", product_quantity[2])
                    print("Category : ", product_quantity[3])
                    print("Quantity : ", product_quantity[5])
                    print("------------------------")
                print("Order details fetched successfully.")
                print()
            else:
                raise CustomerNotFoundException("Customer not present in database.")
        except CustomerNotFoundException as c1:
            print("Error fetching customer. ", c1)
            print()
```

Output :

```
Enter your choice from the menu below :
----- Menu -----
1. Register Customer
2. Create Product
3. Delete Product
4. Add to cart
5. View cart
6. Place order
7. View Customer Order
8. Exit
Enter your choice here : 7
Enter your customer id : 3
Product id :  1
Product Name :  Laptop
Price :  35000.5
Category :  Electronics Gadget
Quantity :  2
------------------------
Order details fetched successfully.
```

8. Exit

```
            print()
    case 8:
        print("Thanks for visiting our Ecommerce app. Please visit again.")
        break
    case _:
        print("Invalid input. Please try again.")
```

Unit Testing

Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

- Write test case to test Product created successfully or not.

```
import pytest
from dao.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
from entity.Customers import Customer
from entity.Products import Product


def test_for_create_product(mocker):
    creating_product = OrderProcessorRepositoryImpl()
    mock_create_product = mocker.patch('dao.OrderProcessorRepositoryImpl.OrderProcessorRepositoryImpl.createProduct')
    test_cases = [
        (Product( name: "Product To Test",  price: 400.5,  category: "demo_category",  stockQuantity: 50), True)
    ]
    for product, expected_value in test_cases:
        mock_create_product.return_value = expected_value
        result = creating_product.createProduct(product)
        assert result == expected_value
```

- Write test case to test product is added to cart successfully or not.

```
def test_for_create_customer(mocker):
    registration_process = OrderProcessorRepositoryImpl()
    mock_create_customer = mocker.patch('dao.OrderProcessorRepositoryImpl.OrderProcessorRepositoryImpl.createCustomer')
    test_cases = [
        (Customer( name: "Customer to test",  email: "testemailcom",  password: "Testing@1"), False),
        (Customer( name: "Customer2 to test",  email: "test.email@gmail.com",  password: "Testin1@"), True)
    ]
    for customer, expected_value in test_cases:
        mock_create_customer.return_value = expected_value
        result = registration_process.createCustomer(customer)
        assert result == expected_value
```

- Write test case to test product is ordered successfully or not.

```
def test_for_place_order(mocker):
    placing_order = OrderProcessorRepositoryImpl()
    mock_place_order = mocker.patch('dao.OrderProcessorRepositoryImpl.OrderProcessorRepositoryImpl.placeOrder')
    test_cases = [
        (Customer( name: "TestCustomer",  email: "test@email.test",  password: "Test@1"),
         [Product( name: "TestProduct",  price: "test_category",  category: "testing",  stockQuantity: 50), 2],
         "RDS Testing", True),
        (Customer( name: "TestCustomer",  email: "test@email.test",  password: "Test@1"), [Product( name: "TestProduct",  price: 700.5,  category: "testing",  stockQuantity: 50), 5],
         "RDS Testing", True),
    ]
    for customer, product, address, expected_value in test_cases:
        mock_place_order.return_value = expected_value
        result = placing_order.placeOrder(customer, product, address)
        assert result == expected_value
```

- Write test case to test exception is thrown correctly or not when customer id or product id not found in database.

```python
def test_for_exception_handling(mocker):
    creating_customer = OrderProcessorRepositoryImpl()
    mock_create_customer = mocker.patch("dao.OrderProcessorRepositoryImpl.OrderProcessorRepositoryImpl.get_customer_by_id")
    test_cases = [
        (9999, True),
        (9999, False)
    ]
    for c_id, expected_output in test_cases:
        mock_create_customer.return_value = expected_output
        result = creating_customer.get_customer_by_id(c_id)
        assert result == mock_create_customer.return_value
```

Output:

```
============================= test session starts ==============================
platform win32 -- Python 3.11.3, pytest-8.0.0, pluggy-1.4.0 -- E:\HexawareAssignments\Python\Ecommerce\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: E:\HexawareAssignments\Python\Ecommerce
plugins: mock-3.12.0
collected 4 items

test_Ecom_App.py::test_for_create_product PASSED                         [ 25%]
test_Ecom_App.py::test_for_create_customer PASSED                        [ 50%]
test_Ecom_App.py::test_for_place_order PASSED                            [ 75%]
test_Ecom_App.py::test_for_exception_handling PASSED                     [100%]

============================== 4 passed in 0.12s ==============================
```

Package Management: