**Case Study on Ecommerce Application**

**Student Name:Sugandan Elangovan**

**Instructions**

• Project submissions should be done through the partcipants’ Github repository, and the link

should be shared with trainers and Hexavarsity.

• Each section builds upon the previous one, and by the end, you will have a comprehensive

**Ecommerce** implemented with a strong focus on **SQL**, **control flow statements, loops, arrays,**

**collections, exception handling**, **database interaction** and **Unit Testing**.

• Follow **object-oriented principles** throughout the project. Use classes and objects to model real

world entities, **encapsulate data and behavior**, and **ensure code reusability**.

• Throw **user defined exceptions** from corresponding methods and handled**.**

• The following **Directory structure** is to be followed in the application.

• **entity/model**

• Create entity classes in this package. All entity class should not have any

business logic.

• **dao**

• Create Service Provider interface to showcase functionalities.

• Create the implementation class for the above interface with db interaction.

• **exception**

• Create user defined exceptions in this package and handle exceptions whenever

needed.

• **util**

• Create a **DBPropertyUtil** class with a static function which takes property file

name as parameter and returns connection string.

• Create a **DBConnUtil** class which holds **static method** which takes connection

string as parameter file and returns **connection object(Use method defined in**

**DBPropertyUtil class to get the connection String )**.

• **main**

• Create a class MainModule and demonstrate the functionalities in a menu

driven application.

**Key Functionalities:**

1. **Customer Management**

• Add new customers, Update, and retrieve customer information and order details,

2. **Product Management:**

• Users can view a list of available products, add, and delete products.

3. **Cart Management:**

• Users can add and remove products to their shopping cart.

4. **Order Management:**

• Users can place orders, which include product details, quantities, and shipping

information.

• The order total is calculated based on the cart contents.

**Create following tables in SQL Schema with appropriate class and write the unit test case for the**

**Ecommerce application.**

**Schema Design:**

1. **customers** table:

• customer\_id (Primary Key)

• name

• email

• password

2. **products** table:

• product\_id (Primary Key)

• name

• price

• description

• stockQuantity

3. **cart** table:

• cart\_id (Primary Key)

• customer\_id (Foreign Key)

• product\_id (Foreign Key)

• quantity

4. **orders** table:

• order\_id (Primary Key)

• customer\_id (Foreign Key)

• order\_date

• total\_price

• shipping\_address

5. **order\_items** table (to store order details):

• order\_item\_id (Primary Key)

• order\_id (Foreign Key)

• product\_id (Foreign Key)

• quantity

**SQL CODE:**

CREATE DATABASE ecom

USE ecom

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

name VARCHAR(255),

email VARCHAR(255),

password VARCHAR(255)

);

INSERT INTO customers VALUES

(1, 'John Doe', 'johndoe@example.com', 'password1'),

(2, 'Jane Smith', 'janesmith@example.com', 'password2'),

(3, 'Robert Johnson', 'robert@example.com', 'password3'),

(4, 'Sarah Brown', 'sarah@example.com', 'password4'),

(5, 'David Lee', 'david@example.com', 'password5'),

(6, 'Laura Hall', 'laura@example.com', 'password6'),

(7, 'Michael Davis', 'michael@example.com', 'password7'),

(8, 'Emma Wilson', 'emma@example.com', 'password8'),

(9, 'William Taylor', 'william@example.com', 'password9'),

(10, 'Olivia Adams', 'olivia@example.com', 'password10');

CREATE TABLE products (

product\_id INT PRIMARY KEY,

name VARCHAR(255),

description VARCHAR(255),

price DECIMAL(10, 2),

stock\_quantity INT

);

INSERT INTO products VALUES

(1, 'Laptop', 'High-performance laptop', 800.00, 10),

(2, 'Smartphone', 'Latest smartphone', 600.00, 15),

(3, 'Tablet', 'Portable tablet', 300.00, 20),

(4, 'Headphones', 'Noise-canceling', 150.00, 30),

(5, 'TV', '4K Smart TV', 900.00, 5),

(6, 'Coffee Maker', 'Automatic coffee maker', 50.00, 25),

(7, 'Refrigerator', 'Energy-efficient', 700.00, 10),

(8, 'Microwave Oven', 'Countertop microwave', 80.00, 15),

(9, 'Blender', 'High-speed blender', 70.00, 20),

(10, 'Vacuum Cleaner', 'Bagless vacuum cleaner', 120.00, 10);

CREATE TABLE cart (

cart\_id INT PRIMARY KEY,

customer\_id INT,

product\_id INT,

quantity INT,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

INSERT INTO cart VALUES

(1, 1, 1, 2),

(2, 1, 3, 1),

(3, 2, 2, 3),

(4, 3, 4, 4),

(5, 3, 5, 2),

(6, 4, 6, 1),

(7, 5, 1, 1),

(8, 6, 10, 2),

(9, 6, 9, 3),

(10, 7, 7, 2);

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

total\_price DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO orders VALUES

(1, 1, '2023-01-05', 1200.00),

(2, 2, '2023-02-10', 900.00),

(3, 3, '2023-03-15', 300.00),

(4, 4, '2023-04-20', 150.00),

(5, 5, '2023-05-25', 1800.00),

(6, 6, '2023-06-30', 400.00),

(7, 7, '2023-07-05', 700.00),

(8, 8, '2023-08-10', 160.00),

(9, 9, '2023-09-15', 140.00),

(10, 10, '2023-10-20', 1400.00);

CREATE TABLE order\_items (

order\_item\_id int IDENTITY(1,1),

order\_id INT,

product\_id INT,

quantity INT,

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

PRIMARY KEY (order\_item\_id)

);

drop table order\_items

INSERT INTO order\_items VALUES

( 1, 1, 2),

( 1, 3, 1),

( 2, 2, 3),

( 3, 5, 2),

( 4, 4, 4),

( 4, 6, 1),

( 5, 1, 1),

( 5, 2, 2),

( 6, 10, 2),

( 6, 9, 3);

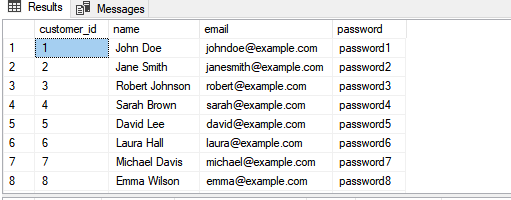
SELECT \* FROM CUSTOMERS;

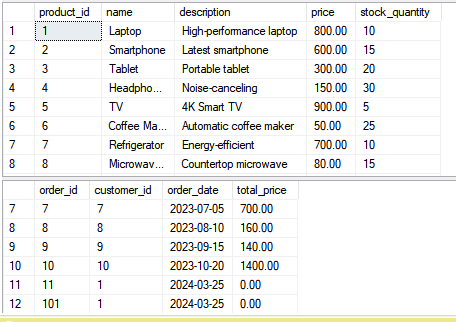
SELECT \* FROM PRODUCTS;

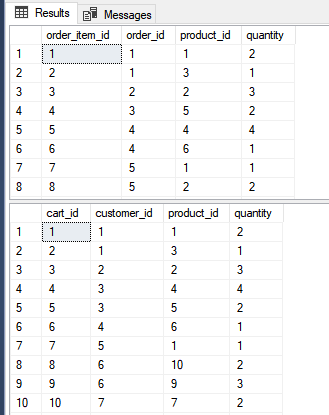
SELECT \* FROM ORDERS;

SELECT \* FROM order\_items;

SELECT \* FROM cart;







**Create the model/entity classes corresponding to the schema within package entity with variables**

**declared private, constructors(default and parametrized) and getters,setters )**

6. **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<Product,quantity>>, string**

**shippingAddress):** should update order table and orderItems table.

1. parameter: Customer customer, list of product and quantity

2. return type: boolean

9. **getOrdersByCustomer()**

1. parameter: customerid

2. return type: list of product and quantity

**Entity.py:**

import ecom.util

import ecom.exception

import ecom.dao

import pyodbc

class OrderProcessorRepositoryImpl(ecom.exception.OrderProcessorRepository):

def \_\_init\_\_(self):

self.conn = ecom.dao.DBConnection.getConnection()

def retrieveProductsFromCart(self, customer\_id, cart\_id):

try:

cursor = self.conn.cursor()

cursor.execute(

"SELECT p.product\_id, p.price, c.quantity FROM cart c JOIN products p ON c.product\_id = p.product\_id WHERE c.customer\_id = ? AND c.cart\_id = ?",

(customer\_id, cart\_id))

rows = cursor.fetchall()

products\_quantity\_map = {}

for row in rows:

product\_id = row.product\_id

price = row.price

quantity = row.quantity

print(f"Raw quantity for product {product\_id}: {quantity}")

try:

quantity = int(quantity)

except ValueError:

print(f"Error converting quantity to integer for product {product\_id}. Quantity value: {quantity}")

quantity = 0

print(f"Processed quantity for product {product\_id}: {quantity}")

products\_quantity\_map[product\_id] = {'price': price, 'quantity': quantity}

return products\_quantity\_map

except pyodbc.Error as ex:

print(f"Error retrieving products from cart: {ex}")

return None

def createProduct(self, product):

try:

cursor = self.conn.cursor()

cursor.execute("INSERT INTO products (product\_id,name, description, price, stock\_quantity) VALUES (?,?, ?, ?, ?)",

(product['product\_id'],product['name'], product['description'], product['price'], product['stock\_quantity']))

self.conn.commit()

print("Product created successfully")

return True

except pyodbc.Error as ex:

print(f"Error creating product: {ex}")

return False

def createCustomer(self, customer):

try:

cursor = self.conn.cursor()

cursor.execute("INSERT INTO customers (customer\_id,name, email, password) VALUES (?,?, ?, ?)",

(customer['customer\_id'],customer['name'], customer['email'], customer['password']))

self.conn.commit()

print("Customer created successfully")

return True

except pyodbc.Error as ex:

print(f"Error creating customer: {ex}")

return False

def deleteProduct(self, productId):

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM products WHERE product\_id = ?", (productId,))

self.conn.commit()

print("Product deleted successfully")

return True

except pyodbc.Error as ex:

print(f"Error deleting product: {ex}")

return False

def deleteCustomer(self, customerId):

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM customers WHERE customer\_id = ?", (customerId,))

self.conn.commit()

print("Customer deleted successfully")

return True

except pyodbc.Error as ex:

print(f"Error deleting customer: {ex}")

return False

def addToCart(self,cart\_id, customer, product, quantity):

try:

cursor = self.conn.cursor()

cursor.execute("INSERT INTO cart (cart\_id,customer\_id, product\_id, quantity) VALUES (?,?, ?, ?)",

(cart\_id,customer['customer\_id'], product['product\_id'], quantity))

self.conn.commit()

print("Product added to cart successfully")

return True

except pyodbc.Error as ex:

print(f"Error adding product to cart: {ex}")

return False

def removeFromCart(self, customer, product):

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM cart WHERE customer\_id = ? AND product\_id = ?",

(customer['customer\_id'], product['product\_id']))

self.conn.commit()

print("Product removed from cart successfully")

return True

except pyodbc.Error as ex:

print(f"Error removing product from cart: {ex}")

return False

def getAllFromCart(self, customer):

try:

cursor = self.conn.cursor()

rows = cursor.execute("SELECT products.\* FROM products JOIN cart ON products.product\_id = cart.product\_id WHERE cart.customer\_id = ?",

(customer['customer\_id'],))

products = []

for row in rows:

products.append({'product\_id': row.product\_id, 'name': row.name, 'description': row.description, 'price': row.price, 'stock\_quantity': row.stock\_quantity})

return products

except pyodbc.Error as ex:

print(f"Error getting products from cart: {ex}")

return []

def placeOrder(self, order\_id, customer, products\_quantity\_map, shippingAddress):

try:

cursor = self.conn.cursor()

cart\_products = self.retrieveProductsFromCart(customer['customer\_id'], customer['cart\_id'])

total\_price = sum(

item['quantity'] \* item['price'] for item in products\_quantity\_map.values()

)

cursor.execute(

"INSERT INTO orders (order\_id, customer\_id, order\_date, total\_price) VALUES (?, ?, GETDATE(), ?)",

(order\_id, customer['customer\_id'], total\_price))

for product\_id, item in products\_quantity\_map.items():

cart\_quantity = cart\_products.get(product\_id, {'quantity': 0})['quantity']

if cart\_quantity >= item['quantity']:

cursor.execute("INSERT INTO order\_items (order\_id, product\_id, quantity) VALUES (?, ?, ?)",

(order\_id, product\_id, item['quantity']))

self.conn.commit()

print(f"Product with ID {product\_id} added to order successfully")

else:

print(f"Insufficient quantity for product with ID {product\_id} in the cart")

return False

print("Order placed successfully")

return True

except pyodbc.Error as ex:

print(f"Error placing order: {ex}")

return False

def getOrdersByCustomer(self, customerId, customer\_id=None):

if not self.customerExists(customer\_id):

raise ecom.exception.CustomerNotFoundException(f"Customer with ID {customer\_id} not found")

try:

cursor = self.conn.cursor()

cursor.execute("SELECT products.\*, order\_items.quantity FROM products JOIN order\_items ON products.product\_id = order\_items.product\_id JOIN orders ON orders.order\_id = order\_items.order\_id WHERE orders.customer\_id = ?",

(customerId,))

rows = cursor.fetchall()

orders = []

for row in rows:

orders.append({'product\_id': row.product\_id, 'name': row.name, 'description': row.description, 'price': row.price, 'stock\_quantity': row.stock\_quantity, 'quantity': row.quantity})

return orders

except ecom.exception.CustomerNotFoundException as ex:

print(f"Error getting orders by customer: {ex}")

return []

def customerExists(self, customer\_id):

try:

cursor = self.conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM customers WHERE customer\_id = ?", (customer\_id,))

count = cursor.fetchone()[0] # Fetch the count result

return count > 0

except pyodbc.Error as ex:

print(f"Error checking if customer exists: {ex}")

return False

def deleteProductByID(self,product\_id):

if not self.ProductExists(product\_id):

raise ecom.exception.ProductNotFoundException(f"Product with ID {product\_id} not found")

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM products WHERE product\_id = ?", (product\_id,))

self.conn.commit() # Commit the transaction

print("Product deleted successfully")

return True

except pyodbc.Error as ex:

print(f"Error deleting product: {ex}")

return False

def ProductExists(self, product\_id):

try:

cursor = self.conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM products WHERE product\_id = ?", (product\_id,))

count = cursor.fetchone()[0] # Fetch the count result

return count > 0

except pyodbc.Error as ex:

print(f"Error checking if customer exists: {ex}")

return False

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

Connect your application to the SQL database:

**Dao.py:**

import pyodbc

import ecom.util

class DBConnection:

connection = None

@staticmethod

def getConnection():

if DBConnection.connection is None:

try:

connection\_string = ecom.util.PropertyUtil.getPropertyString()

DBConnection.connection = pyodbc.connect(connection\_string)

print("Connected Successfully")

except pyodbc.Error as ex:

print(f"Error: {ex}")

return DBConnection.connection

def close\_connection(self):

if DBConnection.connection:

DBConnection.connection.close()

print("Connection closed.")

**8. 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

.

**Util.py:**

class PropertyUtil:

@staticmethod

def getPropertyString():

return 'Driver={SQL Server};Server=DESKTOP-A08GADU\SQLEXPRESS01;Database=ecom;Trusted\_Connection=yes;'

1. **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

• **ProductNotFoundException**: throw this exception when user enters an invalid product

id which doesn’t exist in db

• **OrderNotFoundException**: throw this exception when user enters an invalid order id

which doesn’t exist in db

**Exception.py:**

class CustomerNotFoundException(Exception):

pass

class ProductNotFoundException(Exception):

pass

class OrderNotFoundException(Exception):

pass

class OrderProcessorRepository:

def createProduct(self, product):

pass

def createCustomer(self, customer):

pass

def deleteProduct(self, productId):

pass

def deleteCustomer(self, customerId):

pass

def addToCart(self, customer, product, quantity):

pass

def removeFromCart(self, customer, product):

pass

def getAllFromCart(self, customer):

pass

def placeOrder(self, customer, products\_quantity\_map, shippingAddress):

pass

def getOrdersByCustomer(self, customerId):

pass

**10. 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. Register Customer.

2. Create Product.

3. Delete Product.

4. Add to cart.

5. View cart.

6. Place order.

7. View Customer Order

**Main.py:**

import pyodbc

import ecom

class EcomApp:

@staticmethod

def main():

order = ecom.entity.OrderProcessorRepositoryImpl()

order\_repo = ecom.dao.DBConnection()

order\_repo.getConnection()

while True:

print("\nMenu:")

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 = input("Enter your choice: ")

if choice == "1":

# Register Customer

customer\_id = input("Enter customer ID: ")

name = input("Enter customer name: ")

email = input("Enter customer email: ")

password = input("Enter customer password: ")

customer = {'customer\_id':customer\_id,'name': name, 'email': email, 'password': password}

order.createCustomer(customer)

elif choice == "2":

# Create Product

product\_id=int(input("Enter product ID:"))

name = input("Enter product name: ")

description = input("Enter product description: ")

price = float(input("Enter product price: "))

stock\_quantity = int(input("Enter product stock quantity: "))

product = {'product\_id':product\_id,'name': name, 'description': description, 'price': price, 'stock\_quantity': stock\_quantity}

order.createProduct(product)

elif choice == "3":

# Delete Product

product\_id = int(input("Enter product ID to delete: "))

order.deleteProduct(product\_id)

elif choice == "4":

# Add to Cart

cart\_id=int(input("Enter Cart ID:"))

customer\_id = int(input("Enter customer ID: "))

product\_id = int(input("Enter product ID to add to cart: "))

quantity = int(input("Enter quantity: "))

customer = {'customer\_id': customer\_id}

product = {'product\_id': product\_id}

order.addToCart(cart\_id,customer, product, quantity)

elif choice == "5":

# View Cart

customer\_id = int(input("Enter customer ID: "))

customer = {'customer\_id': customer\_id}

cart\_items = order.getAllFromCart(customer)

print("Cart Items:")

for item in cart\_items:

print(item)

elif choice == "6":

order\_id=int(input("Enter the order ID:"))

customer\_id = int(input("Enter customer ID: "))

cart\_id=int(input("Enter cart ID:"))

customer = {'customer\_id': customer\_id, 'cart\_id': cart\_id}

products\_quantity\_map=order.retrieveProductsFromCart(customer\_id,cart\_id)

shipping\_address = input("Enter shipping address: ")

order.placeOrder(order\_id, customer, products\_quantity\_map, shipping\_address)

elif choice == "7":

customer\_id = int(input("Enter customer ID: "))

orders = order.getOrdersByCustomer(customer\_id)

print("Customer Orders:")

for order in orders:

print(order)

elif choice == "8":

print("Exiting...")

order\_repo.close\_connection()

break

else:

print("Invalid choice. Please enter a number between 1 and 8.")

if \_\_name\_\_ == "\_\_main\_\_":

EcomApp.main()

**Unit Testing**

11. 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.

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

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

• write test case to test exception is thrown correctly or not when customer id or product

id not found in database.

**Testing.py:**

import unittest

import ecom

class TestEcommerceSystem(unittest.TestCase):

def setUp(self):

self.order\_processor = ecom.entity.OrderProcessorRepositoryImpl()

def tearDown(self):

pass

def test\_create\_product\_success(self):

product = {'product\_id':81,'name': 'Test Product', 'description': 'Test Description', 'price': 50.0, 'stock\_quantity': 10}

result = self.order\_processor.createProduct(product)

self.assertTrue(result)

def test\_add\_to\_cart\_success(self):

cart\_id = 902

customer\_id = 1

customer = {'customer\_id': customer\_id}

product\_id = 1

product = {'product\_id': product\_id}

quantity = 2

result = self.order\_processor.addToCart(cart\_id, customer, product, quantity)

self.assertTrue(result)

def test\_place\_order\_success(self):

order\_id=901

customer = {'customer\_id': 1, 'cart\_id': 1}

products\_quantity\_map = self.order\_processor.retrieveProductsFromCart(customer['customer\_id'],customer['cart\_id']) # Example: {product\_id: quantity}

shipping\_address = 'Test Address'

result = self.order\_processor.placeOrder(order\_id,customer, products\_quantity\_map, shipping\_address)

self.assertTrue(result)

def test\_customer\_not\_found\_exception(self):

with self.assertRaises(ecom.exception.CustomerNotFoundException):

self.order\_processor.getOrdersByCustomer(999)

def test\_product\_not\_found\_exception(self):

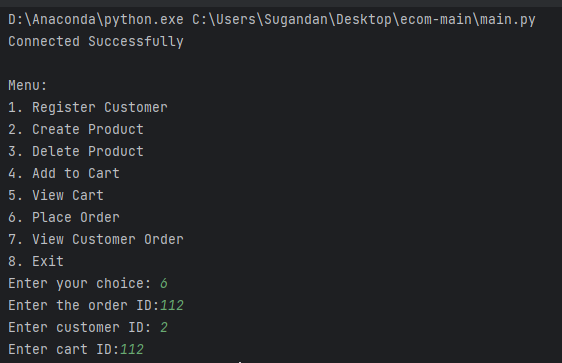
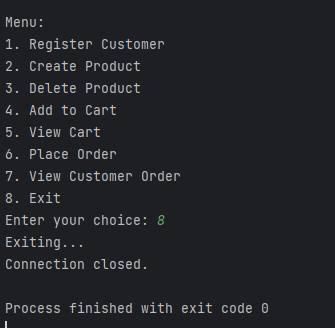
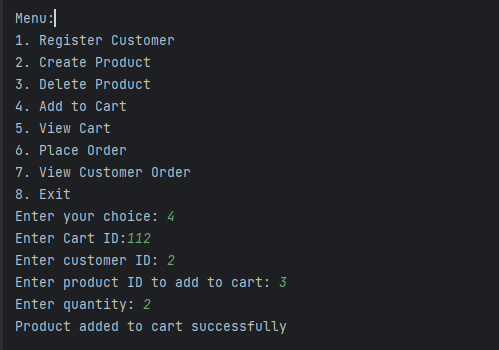
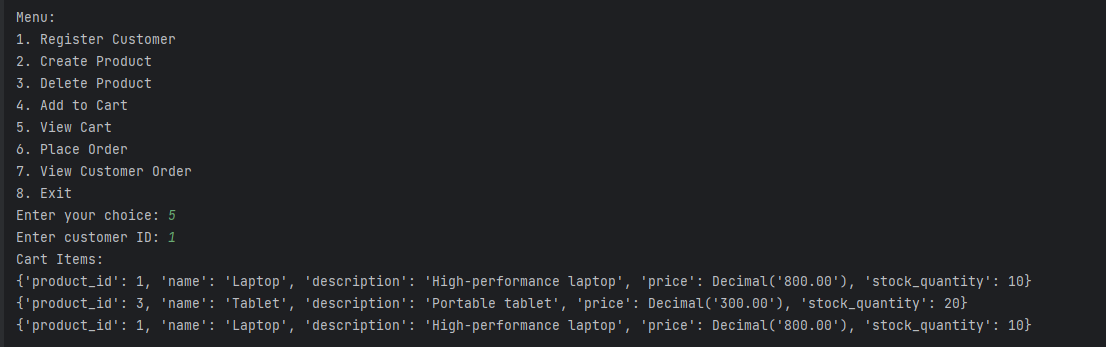
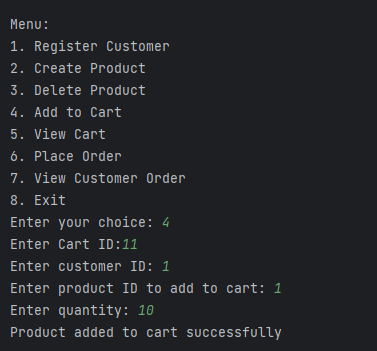
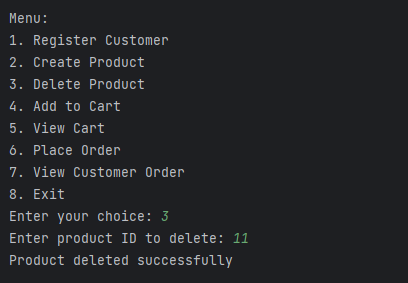
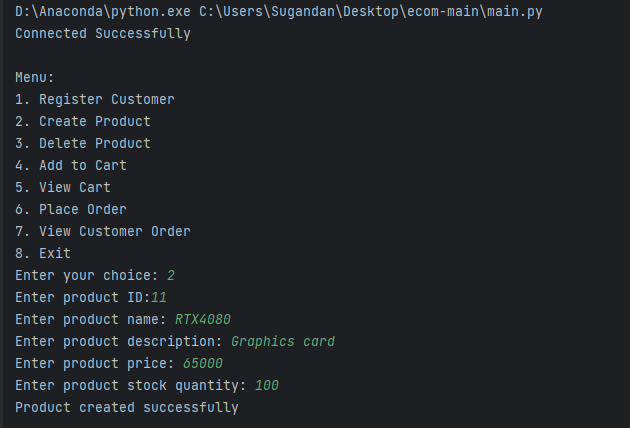
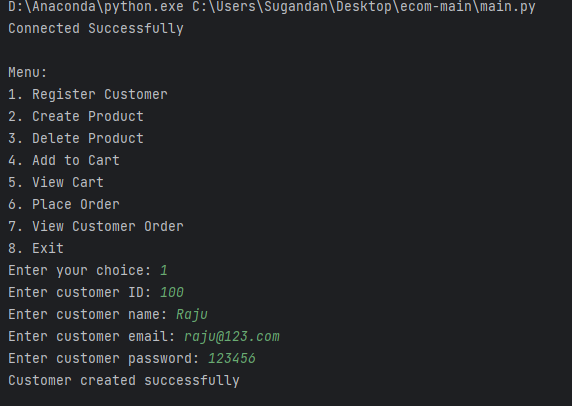
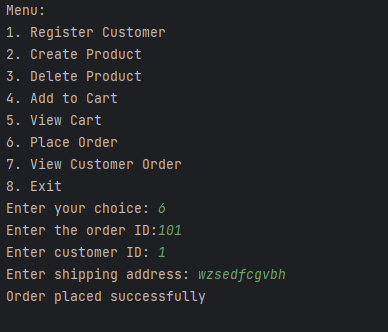
with self.assertRaises(ecom.exception.ProductNotFoundException):

self.order\_processor.deleteProductByID(999)

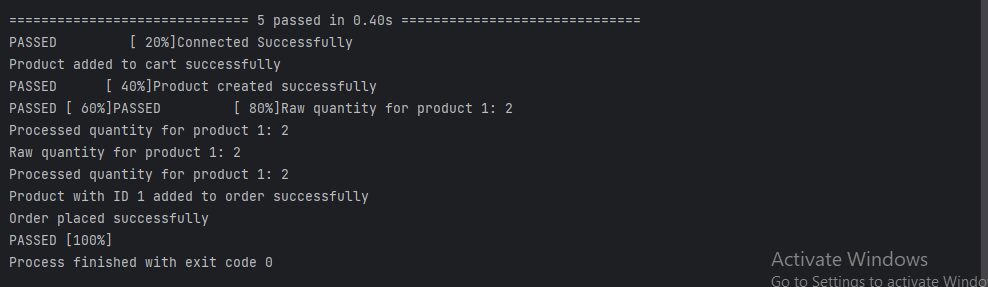
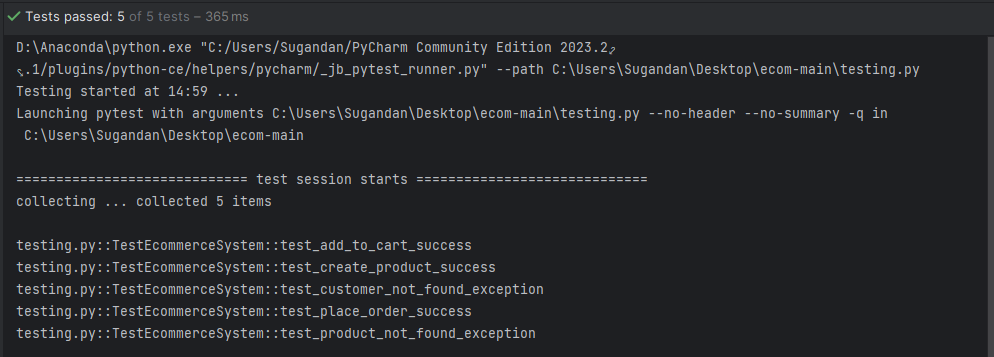
if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Main Output:**

**ec62ec63**

**Testing Output:**

****