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;
```

Results Messages

	customer_id	name	email	password
1	1	John Doe	johndoe@example.com	password1
2	2	Jane Smith	janesmith@example.com	password2
3	3	Robert Johnson	robert@example.com	password3
4	4	Sarah Brown	sarah@example.com	password4
5	5	David Lee	david@example.com	password5
6	6	Laura Hall	laura@example.com	password6
7	7	Michael Davis	michael@example.com	password7
8	8	Emma Wilson	emma@example.com	password8

	product_id	name	description	price	stock_quantity
1	1	Laptop	High-performance laptop	800.00	10
2	2	Smartphone	Latest smartphone	600.00	15
3	3	Tablet	Portable tablet	300.00	20
4	4	Headpho...	Noise-canceling	150.00	30
5	5	TV	4K Smart TV	900.00	5
6	6	Coffee Ma...	Automatic coffee maker	50.00	25
7	7	Refrigerator	Energy-efficient	700.00	10
8	8	Microwav...	Countertop microwave	80.00	15

	order_id	customer_id	order_date	total_price
7	7	7	2023-07-05	700.00
8	8	8	2023-08-10	160.00
9	9	9	2023-09-15	140.00
10	10	10	2023-10-20	1400.00
11	11	1	2024-03-25	0.00
12	101	1	2024-03-25	0.00

Results Messages

	order_item_id	order_id	product_id	quantity
1	1	1	1	2
2	2	1	3	1
3	3	2	2	3
4	4	3	5	2
5	5	4	4	4
6	6	4	6	1
7	7	5	1	1
8	8	5	2	2

	cart_id	customer_id	product_id	quantity
1	1	1	1	2
2	2	1	3	1
3	3	2	2	3
4	4	3	4	4
5	5	3	5	2
6	6	4	6	1
7	7	5	1	1
8	8	6	10	2
9	9	6	9	3
10	10	7	7	2

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;'
```

9. 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['car
t_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:

```
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: 6
Enter the order ID:101
Enter customer ID: 1
Enter shipping address: wzsedfcgvbh
Order placed successfully
```

```
D:\Anaconda\python.exe C:\Users\Sugandan\Desktop\ecom-main\main.py
Connected Successfully

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: 1
Enter customer ID: 100
Enter customer name: Raju
Enter customer email: raju@123.com
Enter customer password: 123456
Customer created successfully
```

```
D:\Anaconda\python.exe C:\Users\Sugandan\Desktop\ecom-main\main.py
Connected Successfully

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: 2
Enter product ID:11
Enter product name: RTX4080
Enter product description: Graphics card
Enter product price: 65000
Enter product stock quantity: 100
Product created successfully

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: 3
Enter product ID to delete: 11
Product deleted successfully
```

```
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: 4
Enter Cart ID:11
Enter customer ID: 1
Enter product ID to add to cart: 1
Enter quantity: 10
Product added to cart successfully
```

```
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: 5
Enter customer ID: 1
Cart Items:
{'product_id': 1, 'name': 'Laptop', 'description': 'High-performance laptop', 'price': Decimal('800.00'), 'stock_quantity': 10}
{'product_id': 3, 'name': 'Tablet', 'description': 'Portable tablet', 'price': Decimal('300.00'), 'stock_quantity': 20}
{'product_id': 1, 'name': 'Laptop', 'description': 'High-performance laptop', 'price': Decimal('800.00'), 'stock_quantity': 10}
```

```
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: 4
Enter Cart ID:112
Enter customer ID: 2
Enter product ID to add to cart: 3
Enter quantity: 2
Product added to cart successfully
```

```
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: 5
Enter customer ID: 2
Cart Items:
{'product_id': 2, 'name': 'Smartphone', 'description': 'Latest smartphone', 'price': Decimal('600.00'), 'stock_quantity': 15}
{'product_id': 1, 'name': 'Laptop', 'description': 'High-performance laptop', 'price': Decimal('800.00'), 'stock_quantity': 10}
{'product_id': 3, 'name': 'Tablet', 'description': 'Portable tablet', 'price': Decimal('300.00'), 'stock_quantity': 20}
```

```
D:\Anaconda\python.exe C:\Users\Sugandan\Desktop\ecom-main\main.py
Connected Successfully

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: 7
Enter customer ID: 2
Customer Orders:
{'product_id': 2, 'name': 'Smartphone', 'description': 'Latest smartphone', 'price': Decimal('600.00'), 'stock_quantity': 15, 'quantity': 3}
{'product_id': 3, 'name': 'Tablet', 'description': 'Portable tablet', 'price': Decimal('300.00'), 'stock_quantity': 20, 'quantity': 2}
```

```
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: 8
Exiting...
Connection closed.

Process finished with exit code 0
```

```
D:\Anaconda\python.exe C:\Users\Sugandan\Desktop\ecom-main\main.py
Connected Successfully

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: 6
Enter the order ID:112
Enter customer ID: 2
Enter cart ID:112
Enter shipping address: dafsfsa
Product with ID 3 added to order successfully
Order placed successfully
```

Testing Output:

```
✓ Tests passed: 5 of 5 tests – 365 ms
D:\Anaconda\python.exe "C:/Users/Sugandan/PyCharm Community Edition 2023.2.1/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py" --path C:\Users\Sugandan\Desktop\ecom-main\testing.py
Testing started at 14:59 ...
Launching pytest with arguments C:\Users\Sugandan\Desktop\ecom-main\testing.py --no-header --no-summary -q in C:\Users\Sugandan\Desktop\ecom-main

============================ test session starts ============================
collecting ... collected 5 items

testing.py::TestEcommerceSystem::test_add_to_cart_success
testing.py::TestEcommerceSystem::test_create_product_success
testing.py::TestEcommerceSystem::test_customer_not_found_exception
testing.py::TestEcommerceSystem::test_place_order_success
testing.py::TestEcommerceSystem::test_product_not_found_exception

============================ 5 passed in 0.40s ============================
PASSED         [ 20%]Connected Successfully
Product added to cart successfully
PASSED      [ 40%]Product created successfully
PASSED [ 60%]PASSED         [ 80%]Raw quantity for product 1: 2
Processed quantity for product 1: 2
Raw quantity for product 1: 2
Processed quantity for product 1: 2
Product with ID 1 added to order successfully
Order placed successfully
PASSED [100%]
Process finished with exit code 0
```