Case Study on Ecommerce Application

Rakshitha Devi J

superset id: 5369940

1. Introduction

The rapid growth of online shopping has increased the demand for robust, scalable, and user-friendly ecommerce platforms. This project aims to simulate a simplified backend for an ecommerce system, focusing on core concepts of software development such as object-oriented programming, SQL database interaction, exception handling, and unit testing. By modeling real-world entities like customers, products, carts, and orders, this system demonstrates how data flows through different components in a layered architecture. The application is built using modular principles, making it easier to test, maintain, and extend. The project serves as a hands-on case study to bridge theoretical knowledge with practical implementation.

2. Purpose of the Project

The purpose of this E-commerce Application project is to simulate a real-world online shopping backend system using core concepts of object-oriented programming, SQL database integration, control structures, exception handling, and unit testing. This project aims to develop a menu-driven, modular application that manages customers, products, carts, and orders through a structured and layered architecture. By incorporating entity modeling, a DAO layer for database operations, custom exception handling, and utility-based DB connectivity, the project provides a hands-on understanding of full-stack backend development principles. It is intended to strengthen problem-solving and software engineering skills in a scalable, testable, and maintainable manner.

3. Scope of the Project

This project covers the complete development of an Ecommerce backend system with the following scope:

Entity Modeling: Implement real-world entities like Customer, Product, Cart, Order, and OrderItem with private attributes, constructors, and getter/setter methods.

Data Access Layer (DAO): Create interfaces and an implementation class (OrderProcessorRepositoryImpl) to handle all database operations using JDBC.

Custom Exceptions: Define and handle user-defined exceptions such as CustomerNotFoundException, ProductNotFoundException, and OrderNotFoundException.

Database Integration: Design and implement relational tables in MySQL with proper foreign key relationships. Use PropertyUtil and DBConnection utilities to manage secure and dynamic DB connections.

Functionalities: Provide customer registration, product creation/deletion, cart operations, and order processing with support for listing customer orders.

Menu-Driven Application: Develop a EcomApp class in the main package to drive all operations through user interaction.

Unit Testing: Ensure system reliability by writing test cases for product creation, cart addition, order placement, and exception handling using the unittest module.

4. Structure of the project

4.1 SQL Structure (Database Schema)

The Ecommerce database schema is designed to efficiently store and manage customer information, product details, cart operations, and order processing. It follows a relational structure using MySQL and ensures data integrity using primary and foreign key constraints.

Database creation:

```
create database ecomm_db;

use ecomm_db;
```

Table creation:

1. customers table:

• customer_id (Primary Key)

• name

• email

• password

Description:

This table stores details of all users who register on the platform.

- Primary Key: customer_id uniquely identifies each customer.
- Fields: name, email, and password.
- The email is unique to prevent duplicate accounts.

SQL Query:

```
create table customers (

    customer_id int primary key auto_increment,

    name varchar(100) not null,
    email varchar(100) unique not null,
    password varchar(255) not null
);
```

2. products table

• product_id (Primary Key)

• name

• price

• description

• stockQuantity

Description:

This table holds information about products available for purchase.

- Primary Key: product_id.
- Fields: name, price, description, and stockQuantity.
- price is stored as a decimal to handle currency format.
- stockQuantity tracks the available units in inventory.

SQL Query:

```
create table products (
    product_id int primary key auto_increment,
    name varchar(100) not null,
    price decimal(10,2) not null,
    description text,
    stockquantity int not null
);
```

3. cart table:

• cart_id (Primary Key)

• customer_id (Foreign Key)

• product_id (Foreign Key)

• quantity

Description:

This table maintains temporary product selections by customers.

- Primary Key: cart_id.
- Foreign Keys:
    - customer_id references customers
    - product_id references products
- Field: quantity indicates how many units the customer wants.
- ON DELETE CASCADE ensures cart items are removed if related customer/product is deleted.

SQL Query:

```
create table cart (
   cart_id int primary key auto_increment,
   customer_id int,
   product_id int,
   quantity int not null,
   foreign key (customer_id) references customers(customer_id) on delete cascade,
   foreign key (product_id) references products(product_id) on delete cascade
);
```

4. orders table:

• order_id (Primary Key)

• customer_id (Foreign Key)

• order_date

• total_price

• shipping_address

Description:

Stores the finalized purchases made by customers.

- Primary Key: order_id.
- Foreign Key: customer_id references customers.
- Fields:
  - order_date automatically captures the timestamp of the order.
  - total_price stores the total amount.
  - shipping_address contains delivery details.

SQL Query:

```
create table orders (
   order_id int primary key auto_increment,
   customer_id int,
   order_date timestamp default current_timestamp,
   total_price decimal(10,2) not null,
   shipping_address text not null,
   foreign key (customer_id) references customers(customer_id) on delete cascade
);
```

5. order_items table (to store order details):

• order_item_id (Primary Key)

• order_id (Foreign Key)

• product_id (Foreign Key)

• quantity

Description:

This is a junction table to handle many-to-many relationships between orders and products.

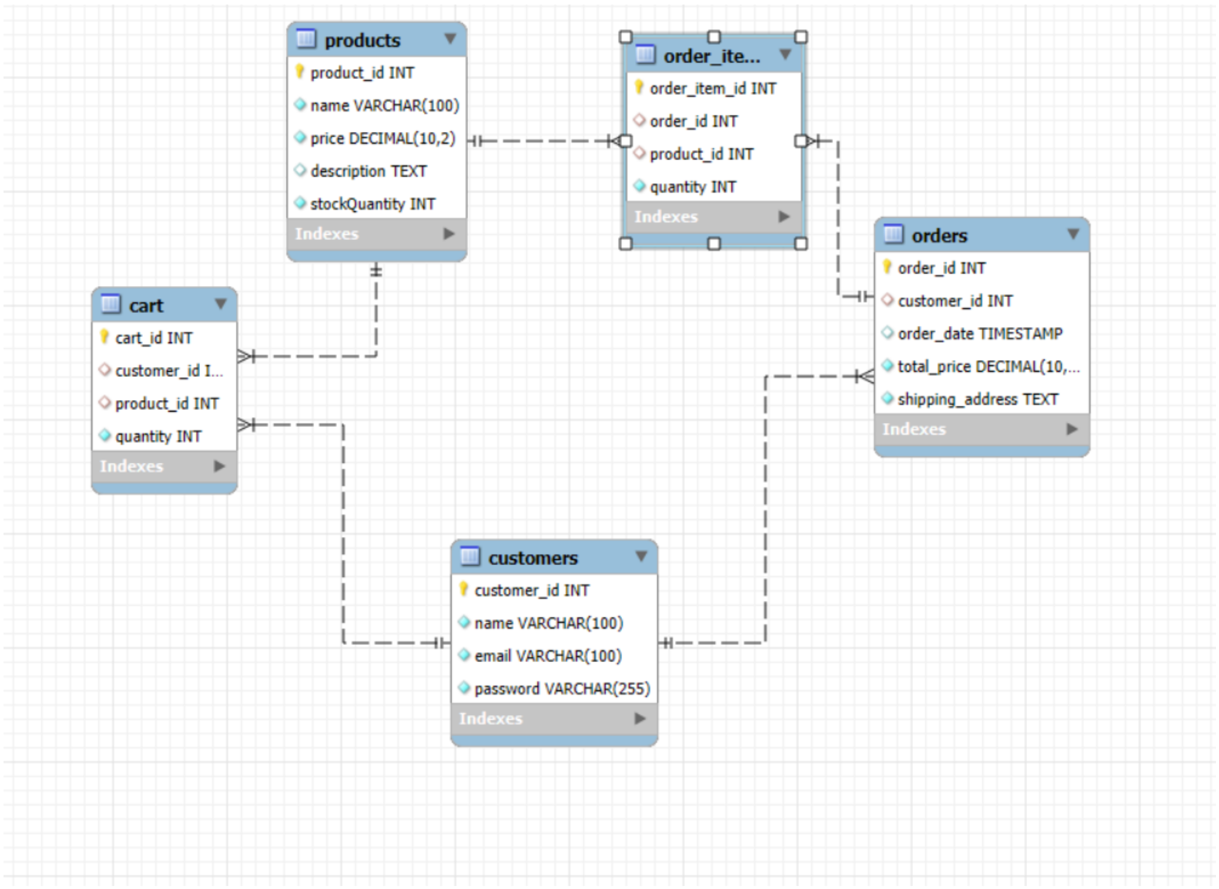
- Primary Key: order_item_id.
- Foreign Keys:
  - order_id references orders
  - product_id references products
- Field: quantity denotes how many units of the product are in that order.

SQL Query:

```
create table order_items (
   order_item_id int primary key auto_increment,
   order_id int,
   product_id int,
   quantity int not null,
   foreign key (order_id) references orders(order_id) on delete cascade,
   foreign key (product_id) references products(product_id) on delete cascade
);
```

ER Diagram

An ER (Entity-Relationship) Diagram is a visual representation of the entities (tables) in a database and the relationships between them. It helps in designing and understanding the logical structure of the database. Entities represent real-world objects like Customer, Product, Order, and the relationships represent how these entities are related (e.g., a customer can place many orders).



4.2 OOP Structure (Object-Oriented Programming)

1.Entity Module (entity/):

The entity package is a fundamental part of the project that represents the real-world objects (such as Customer, Product, Cart, Order) as Java/Python classes using Object-Oriented Programming (OOP) principles.

Each entity class directly maps to a table in the MySQL database and contains the attributes/fields representing the columns of the table.

These classes serve as Data Transfer Objects (DTOs) used for carrying data between processes.

Represents the data models (database tables) used in the application.

Classes files:

Admin.py

```
class Admin:
    def __init__(self, admin_id: Optional[int] = None, name: str = None,
password: str = None):
        self.admin_id = admin_id
        self.name = name
        self.password = password


    # Getters and Setters
    def get_admin_id(self):
        return self.admin_id

    def set_admin_id(self, admin_id: int):
        self.admin_id = admin_id

    def get_name(self):
        return self.name

    def set_name(self, name: str):
        self.name = name

    def get_password(self):
        return self.password

    def set_password(self, password: str):
        self.password = password

    # Override __repr__ for better string representation
    def __repr__(self):
        return f"Admin(admin_id={self.admin_id}, name={self.name})"
```

Customer.py

```
class Customer:
  def __init__(self, customer_id=None, name=None, email=None, password=None):
     self.__customer_id = customer_id
     self.__name = name
     self.__email = email
     self.__password = password

  # Getters
  def get_customer_id(self):
     return self.__customer_id

  def get_name(self):
     return self.__name

  def get_email(self):
     return self.__email

  def get_password(self):
     return self.__password
  # Setters
  def set_customer_id(self, customer_id):
    self.__customer_id = customer_id

  def set_name(self, name):
    self.__name = name

  def set_email(self, email):
    self.__email = email

  def set_password(self, password):
    self.__password = password
```

Product.py

```
class Product:
  def __init__(self, product_id=None, name=None, price=None, description=None, stock_quantity=None):
    self.__product_id = product_id
    self.__name = name
    self.__price = price
    self.__description = description
    self.__stock_quantity = stock_quantity

  # Getters
  def get_product_id(self):
    return self.__product_id

def get_name(self):
    return self.__name

def get_price(self):
    return self.__price

def get_description(self):
    return self.__description

def get_stock_quantity(self):
    return self.__stock_quantity

# Setters
def set_product_id(self, product_id):
    self.__product_id = product_id

def set_name(self, name):
    self.__name = name

def set_price(self, price):
    self.__price = price
def set_description(self, description):
    self.__description = description

def set_stock_quantity(self, stock_quantity):
    self.__stock_quantity = stock_quantity
```

Cart.py

```
class Cart:
    def __init__(self, cart_id=None, customer_id=None, product_id=None, quantity=None):
        self.__cart_id = cart_id
        self.__customer_id = customer_id
        self.__product_id = product_id
        self.__quantity = quantity

    # Getters
    def get_cart_id(self):
        return self.__cart_id

    def get_customer_id(self):
        return self.__customer_id

    def get_product_id(self):
        return self.__product_id
    def get_quantity(self):
        return self.__quantity

    # Setters
    def set_cart_id(self, cart_id):
        self.__cart_id = cart_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def set_product_id(self, product_id):
        self.__product_id = product_id

    def set_quantity(self, quantity):
        self.__quantity = quantity
```

Order.py

```
class Order:
    def __init__(self, order_id=None, customer_id=None, order_date=None, total_price=None, shipping_address=None):
        self.__order_id = order_id
        self.__customer_id = customer_id
        self.__order_date = order_date
        self.__total_price = total_price
        self.__shipping_address = shipping_address

    # Getters
    def get_order_id(self):
        return self.__order_id

    def get_customer_id(self):
        return self.__customer_id

    def get_order_date(self):
        return self.__order_date

    def get_total_price(self):
        return self.__total_price

    def get_shipping_address(self):
        return self.__shipping_address

    # Setters
    def set_order_id(self, order_id):
        self.__order_id = order_id
        def set_customer_id(self, customer_id):
            self.__customer_id = customer_id

        def set_order_date(self, order_date):
            self.__order_date = order_date

        def set_total_price(self, total_price):
            self.__total_price = total_price

        def set_shipping_address(self, shipping_address):
            self.__shipping_address = shipping_address
```

OrderItem.py

```
class OrderItem:
    def __init__(self, order_item_id=None, order_id=None, product_id=None, quantity=None):
        self.__order_item_id = order_item_id
        self.__order_id = order_id
        self.__product_id = product_id
        self.__quantity = quantity

    # Getters
    def get_order_item_id(self):
        return self.__order_item_id
```

```
def get_order_id(self):
    return self.__order_id

def get_product_id(self):
    return self.__product_id

def get_quantity(self):
    return self.__quantity

# Setters
def set_order_item_id(self, order_item_id):
    self.__order_item_id = order_item_id

def set_order_id(self, order_id):
    self.__order_id = order_id

def set_product_id(self, product_id):
    self.__product_id = product_id

def set_quantity(self, quantity):
    self.__quantity = quantity
```

Structure: Each class includes:

- Private variables
- Default and parameterized constructors
- Getters and setters

No business logic here, only data representation

2.DAO Module (dao/):

Handles business logic and database interaction.

Files:

- OrderProcessorRepository.py
  - Abstract interface with method signatures like createCustomer(), placeOrder(), etc.
- OrderProcessorRepositoryImpl.py
  - Implements all the methods defined in the interface
  - Uses SQL queries to interact with MySQL DB

Responsibilities:

- Add/delete customer/product
- Cart operations (add/remove/view)
- Place order and retrieve orders
- Handle data persistence

Files:

OrderProcessorRepository.py

```
from abc import ABC, abstractmethod
from typing import List, Dict
from entity.Admin import Admin
from entity.Customer import Customer
from entity.Product import Product
from entity.Order import Order


class OrderProcessorRepository(ABC):

    @abstractmethod
    def create_product(self, product: Product) -> bool:
        pass

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

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

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

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

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

    @abstractmethod
    def get_all_from_cart(self, customer_id: int) -> List[dict]:
        pass
    @abstractmethod
    def place_order(self, customer: Customer, cart_items: list,
shipping_address: str) -> bool:
        pass

    @abstractmethod
    def get_orders_by_customer(self, customer_id: int) -> list:
        pass

    @abstractmethod
    def create_admin(self, admin) -> bool:
        pass

    @abstractmethod
    def view_all_customers(self) -> List[Customer]:
        pass

    @abstractmethod
    def view_all_products(self) -> List[Product]:
        pass

    @abstractmethod
    def view_customer_orders(self, customer_id: int) -> List[Order]:
        pass

    @abstractmethod
    def get_customer_by_email(self, email: str) -> Customer:
        pass

    @abstractmethod
    def cancel_order(self, order_id: int) -> bool:
        pass

    @abstractmethod
    def login_admin(self, name: str, password: str) -> bool:
        pass

    @abstractmethod
    def get_product_by_id(self, product_id):
        pass
```

OrderProcessorRepositoryImpl.py

```
from dao.OrderProcessorRepository import OrderProcessorRepository
from entity.Customer import Customer
from entity.Product import Product
from entity.Admin import Admin
from entity.Order import Order
from util.db_connection import DBConnection
from exception.customer_not_found_exception import
CustomerNotFoundException
from exception.product_not_found_exception import
ProductNotFoundException
from exception.order_not_found_exception import
OrderNotFoundException


class OrderProcessorRepositoryImpl(OrderProcessorRepository):

    def create_product(self, product: Product) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            query = "INSERT INTO products (name, price, description,
stockQuantity) VALUES (%s, %s, %s, %s)"
            cursor.execute(query, (
                product.get_name(),
                product.get_price(),
                product.get_description(),
                product.get_stock_quantity()
            ))
            product.set_product_id(cursor.lastrowid)
            conn.commit()
            success = True
        except Exception as e:
            print(f"Error creating product: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
```

```
            conn.close()
        return success

    def create_customer(self, customer: Customer) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            query = "INSERT INTO customers (name, email, password)
VALUES (%s, %s, %s)"
            cursor.execute(query, (
                customer.get_name(),
                customer.get_email(),
                customer.get_password()
            ))
            customer.set_customer_id(cursor.lastrowid)
            conn.commit()
            success = True
        except Exception as e:
            print(f"Error creating customer: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def delete_product(self, product_id: int) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM products WHERE product_id
= %s", (product_id,))
            if cursor.fetchone() is None:
                raise ProductNotFoundException(f"Product with ID
{product_id} not found.")
```

```
            query = "DELETE FROM products WHERE product_id = %s"
            cursor.execute(query, (product_id,))
            conn.commit()
            success = cursor.rowcount > 0
        except ProductNotFoundException as e:
            print(e)
            raise
        except Exception as e:
            print(f"Error deleting product: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def delete_customer(self, customer_id: int) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM customers WHERE
customer_id = %s", (customer_id,))
            if cursor.fetchone() is None:
                raise CustomerNotFoundException(f"Customer with ID
{customer_id} not found.")

            query = "DELETE FROM customers WHERE customer_id = %s"
            cursor.execute(query, (customer_id,))
            conn.commit()
            success = cursor.rowcount > 0
        except CustomerNotFoundException as e:
            print(e)
            raise
        except Exception as e:
            print(f"Error deleting customer: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def add_to_cart(self, customer: Customer, product: Product, quantity:
int) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            # Check if customer exists
            cursor.execute("SELECT * FROM customers WHERE
customer_id = %s", (customer.get_customer_id(),))
            if cursor.fetchone() is None:
                raise CustomerNotFoundException(f"Customer with ID
{customer.get_customer_id()} not found.")

            # Check if product exists
            cursor.execute("SELECT * FROM products WHERE product_id
= %s", (product.get_product_id(),))
            if cursor.fetchone() is None:
                raise ProductNotFoundException(f"Product with ID
{product.get_product_id()} not found.")

            # Insert into cart
            query = "INSERT INTO cart (customer_id, product_id, quantity)
VALUES (%s, %s, %s)"
            cursor.execute(query, (
                customer.get_customer_id(),
                product.get_product_id(),
                quantity
            ))
            conn.commit()
            success = True
        except (CustomerNotFoundException, ProductNotFoundException)
as e:
            print(e)
            raise
        except Exception as e:
            print(f"Error adding to cart: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def remove_from_cart(self, customer: Customer, product: Product) ->
bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            cursor.execute("SELECT * FROM customers WHERE
customer_id = %s", (customer.get_customer_id(),))
            if cursor.fetchone() is None:
                raise CustomerNotFoundException(f"Customer with ID
{customer.get_customer_id()} not found.")

            cursor.execute("SELECT * FROM products WHERE product_id
= %s", (product.get_product_id(),))
            if cursor.fetchone() is None:
                raise ProductNotFoundException(f"Product with ID
{product.get_product_id()} not found.")

            query = "DELETE FROM cart WHERE customer_id = %s AND
product_id = %s"
            cursor.execute(query, (
                customer.get_customer_id(),
                product.get_product_id()
            ))
            conn.commit()
            success = cursor.rowcount > 0
        except (CustomerNotFoundException, ProductNotFoundException)
as e:
```

```
            print(e)
            raise
        except Exception as e:
            print(f"Error removing from cart: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def place_order(self, customer, cart_items: list, shipping_address: str) -
> bool:
        conn = None
        cursor = None
        success = False
        try:
            customer_id = customer.get_customer_id() if isinstance(customer,
Customer) else customer

            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            total_price = sum(item["product"].get_price() * item["quantity"]
for item in cart_items)

            order_query = "INSERT INTO orders (customer_id, order_date,
total_price, shipping_address) VALUES (%s, NOW(), %s, %s)"
            cursor.execute(order_query, (customer_id, total_price,
shipping_address))
            order_id = cursor.lastrowid

            for item in cart_items:
                product_id = item["product"].get_product_id()
                quantity = item["quantity"]
                order_item_query = "INSERT INTO order_items (order_id,
product_id, quantity) VALUES (%s, %s, %s)"
                cursor.execute(order_item_query, (order_id, product_id,
quantity))

            conn.commit()
            success = True
        except Exception as e:
            print(f"Error placing order: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def get_orders_by_customer(self, customer_id: int):
        conn = None
        cursor = None
        orders = []

        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            # Check if customer exists
            cursor.execute("SELECT * FROM customers WHERE
customer_id = %s", (customer_id,))
            if cursor.fetchone() is None:
                raise CustomerNotFoundException(f"Customer with ID
{customer_id} not found.")

            # Get all orders by the customer
            cursor.execute("""
                SELECT order_id, order_date, total_price, shipping_address
                FROM orders
                WHERE customer_id = %s
                ORDER BY order_date DESC
            """, (customer_id,))
            orders_data = cursor.fetchall()

            if not orders_data:
                raise OrderNotFoundException(f"No orders found for
Customer ID {customer_id}.")

            # For each order, get the associated products
            for order in orders_data:
```

```
            order_id, order_date, total_price, shipping_address = order

            # Get products associated with the order
            cursor.execute("""
                SELECT p.name, p.price, oi.quantity
                FROM order_items oi
                JOIN products p ON oi.product_id = p.product_id
                WHERE oi.order_id = %s
            """, (order_id,))
            items = cursor.fetchall()

            # Add the order and associated items to the orders list
            orders.append({
                "order_id": order_id,
                "order_date": order_date,
                "total_price": total_price,  # Ensure 'total_price' is used
consistently
                "shipping_address": shipping_address,
                "items": [{"name": item[0], "price": item[1], "quantity":
item[2]} for item in items]
                # Structure items as a list of dicts
            })

        except (CustomerNotFoundException, OrderNotFoundException) as
e:
            print(e)
            raise
        except Exception as e:
            print(f"Error retrieving orders: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()

        return orders

    def get_all_from_cart(self, customer_id: int) -> list:
        conn = None
        cursor = None
        cart_items = []
        try:
            if isinstance(customer_id, Customer):
                customer_id = customer_id.get_customer_id()

            conn = DBConnection.get_connection()
            cursor = conn.cursor(dictionary=True)
            query = """
                SELECT p.product_id, p.name, p.price, c.quantity
                FROM cart c
                JOIN products p ON c.product_id = p.product_id
                WHERE c.customer_id = %s
            """
            cursor.execute(query, (customer_id,))
            cart_items = cursor.fetchall()
        except Exception as e:
            print(f"Error retrieving cart items: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return cart_items

    def create_admin(self, admin) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            query = "INSERT INTO admin (name, password) VALUES (%s,
%s)"
            cursor.execute(query, (
                admin.get_name(),
                admin.get_password()
            ))
            admin.set_admin_id(cursor.lastrowid)
            conn.commit()
            success = True
        except Exception as e:
            print(f"Error creating admin: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success

    def view_all_customers(self) -> list[Customer]:
        conn = None
        cursor = None
        customers = []
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor(dictionary=True)
            query = "SELECT * FROM customers"
            cursor.execute(query)
            rows = cursor.fetchall()
            for row in rows:
                customer = Customer(row['customer_id'], row['name'],
row['email'], row['password'])
                customers.append(customer)
        except Exception as e:
            print(f"Error fetching customers: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return customers

    def view_all_products(self):
        conn = None
        cursor = None
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            query = """
                SELECT
                    p.product_id,
                    p.name,
                p.price,
                p.stockQuantity - IFNULL(SUM(c.quantity), 0) AS
available_stock
            FROM
                products p
            LEFT JOIN
                cart c ON p.product_id = c.product_id
            GROUP BY
                p.product_id, p.name, p.price, p.stockQuantity
            """
            cursor.execute(query)
            return cursor.fetchall()
        except Exception as e:
            print("Error retrieving products:", e)
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()

    def view_customer_orders(self, customer_id: int) -> list[Order]:
        conn = None
        cursor = None
        orders = []
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor(dictionary=True)

            cursor.execute("SELECT * FROM customers WHERE
customer_id = %s", (customer_id,))
            if cursor.fetchone() is None:
                raise CustomerNotFoundException(f"Customer with ID
{customer_id} not found.")

            query = "SELECT * FROM orders WHERE customer_id = %s"
            cursor.execute(query, (customer_id,))
            rows = cursor.fetchall()

            if not rows:
                raise OrderNotFoundException(f"No orders found for customer
ID {customer_id}.")
```

```
            for row in rows:
                order = Order(
                    order_id=row["order_id"],
                    customer_id=row["customer_id"],
                    order_date=row["order_date"],
                    total_price=row["total_price"],
                    shipping_address=row["shipping_address"]
                )
                orders.append(order)
        except (CustomerNotFoundException, OrderNotFoundException) as
e:
            print(e)
            raise
        except Exception as e:
            print(f"Error viewing customer orders: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return orders

    def get_customer_by_email(self, email: str) -> Customer:
        conn = None
        cursor = None
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor(dictionary=True)
            query = "SELECT * FROM customers WHERE email = %s"
            cursor.execute(query, (email,))
            row = cursor.fetchone()
            if row:
                return Customer(row['customer_id'], row['name'], row['email'],
row['password'])
            else:
                raise CustomerNotFoundException(f"No customer found with
email: {email}")
        except CustomerNotFoundException as e:
            # Just raise it; let the upper layer handle printing
            raise e
        except Exception as e:
            print(f"Error fetching customer by email: {e}")
            return None
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()

    def cancel_order(self, order_id: int) -> bool:
        conn = None
        cursor = None
        success = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            query = "DELETE FROM orders WHERE order_id = %s"
            cursor.execute(query, (order_id,))
            conn.commit()
            success = True
        except Exception as e:
            print(f"Error canceling order: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return success
    def login_admin(self, name: str, password: str) -> bool:
        conn = None
        cursor = None
        is_authenticated = False
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()
            query = "SELECT * FROM admin WHERE name = %s AND
password = %s"
            cursor.execute(query, (name, password))
            if cursor.fetchone():
                is_authenticated = True
        except Exception as e:
            print(f"Error during admin login: {e}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
        return is_authenticated

    def get_product_by_id(self, product_id: int):
        conn = None
        cursor = None
        try:
            conn = DBConnection.get_connection()
            cursor = conn.cursor()

            query = "SELECT product_id, name, price, description,
stockQuantity FROM products WHERE product_id = %s"
            cursor.execute(query, (product_id,))
            row = cursor.fetchone()

            if row:
                return Product(
                    product_id=row[0],
                    name=row[1],
                    price=row[2],
                    description=row[3],
                    stock_quantity=row[4]
                )
            else:
                return None
        except Exception as e:
            print(f"Error fetching product by ID: {e}")
            raise
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
```

3.Exception Module (exception/):

Defines custom exceptions to handle specific error scenarios.

Custom Exceptions files:

- CustomerNotFoundException
- ProductNotFoundException
- OrderNotFoundException

Purpose: To manage errors gracefully and avoid system crashes with meaningful messages.

Files:

Customer_Not_Found_Exception.py:

```
class CustomerNotFoundException(Exception):
   def __init__(self, message="Customer not found in the database"):
       self.message = message
       super().__init__(self.message)
```

Order_Not_Found_Exception.py:

```
class OrderNotFoundException(Exception):
  def __init__(self, message="Order not found in the database"):
      self.message = message
      super().__init__(self.message)
```

Product_Not_Found_Exception.py:

```
class ProductNotFoundException(Exception):
  def __init__(self, message="Product not found in the database"):
      self.message = message
      super().__init__(self.message)
```

4.Utility Module (util/)

Provides reusable utility functions, especially for DB connection.

Files:

- PropertyUtil.py
    - Reads database connection details from db.properties
- DBConnection.py
    - Establishes DB connection using property values

Files:

DB_connection.py:

```
import mysql.connector
from util.property_util import PropertyUtil


class DBConnection:
  _connection = None


  @staticmethod
  def get_connection():
        try:
      if DBConnection._connection is None or not
DBConnection._connection.is_connected():
        db_config = PropertyUtil.get_database_config()
        DBConnection._connection = mysql.connector.connect(
          host=db_config["host"],
          user=db_config["user"],
          password=db_config["password"],
          database=db_config["database"]
        )
      return DBConnection._connection
    except mysql.connector.Error as e:
      print(f"Error connecting to MySQL: {e}")
      return None


  @staticmethod
  def close_connection():
        if DBConnection._connection and DBConnection._connection.is_connected():
            DBConnection._connection.close()
            DBConnection._connection = None
```

Property_Util.py:

```
import configparser
import os
class PropertyUtil:
  @staticmethod
  def get_database_config():
     config = configparser.ConfigParser()

     # Absolute Path
     config_path = os.path.join(os.path.dirname(__file__), "../config.ini")

     if not os.path.exists(config_path):
        raise FileNotFoundError(f"Config file not found: {config_path}")

     config.read(config_path)

     return {
        "host": config.get("database", "host"),
        "user": config.get("database", "user"),
        "password": config.get("database", "password"),
        "database": config.get("database", "database")
    }
```

<u>5.Main Application</u> (app/):

Menu-driven interface to interact with the system.

File:

- EcomApp.py

Responsibilities:

- Prompt user with options like:
    - Register customer
    - Add/delete product
    - Add/remove/view cart
    - Place order
    - View orders
- Call corresponding methods from the DAO layer
- Handle exceptions

Files:

Main.py:

```
import sys
from dao.OrderProcessorRepositoryImpl import
OrderProcessorRepositoryImpl
from entity.Customer import Customer
from entity.Product import Product
from entity.Admin import Admin
from exception.customer_not_found_exception import
CustomerNotFoundException
from exception.product_not_found_exception import
ProductNotFoundException
from exception.order_not_found_exception import
```

OrderNotFoundException

```
class EcomApp:
    def __init__(self):
        self.order_repo = OrderProcessorRepositoryImpl()
        self.logged_in_user = None
        self.is_admin = False

    def login(self):
        print("===== Welcome to E-commerce App =====")
        print("1. Sign In")
        print("2. Sign Up")
        choice = input("Enter your choice (1 or 2): ")

        if choice == "1":
            self.sign_in()
        elif choice == "2":
            self.sign_up()
        else:
            print("Invalid input. Try again.")
            self.login()

    def sign_in(self):
        print("\n===== Sign In =====")
        user_type = input("Are you a (1) Admin or (2) Customer? Enter 1 or
2: ")

        if user_type == "1":
            self.admin_login()
        elif user_type == "2":
            self.customer_login()
        else:
            print("Invalid choice! Please select 1 for Admin or 2 for
Customer.")
            self.sign_in()

    def sign_up(self):
        print("\n===== Sign Up =====")
        user_type = input("Sign up as (1) Admin or (2) Customer? Enter 1 or
2: ")
```

```
        if user_type == "1":
            name = input("Enter admin name: ")
            password = input("Enter admin password: ")
            self.create_admin(name, password)
        elif user_type == "2":
            name = input("Enter your name: ")
            email = input("Enter your email: ")
            password = input("Enter your password: ")
            customer = Customer(name=name, email=email,
password=password)
            if self.order_repo.create_customer(customer):
                print("Customer account created successfully! You can now log
in.")
            else:
                print("Failed to create customer account.")
            self.login()
        else:
            print("Invalid choice! Please select 1 for Admin or 2 for
Customer.")
            self.sign_up()

    def create_admin(self, name, password):
        admin = Admin(name=name, password=password)
        if self.order_repo.create_admin(admin):
            print("Admin account created successfully!")
        else:
            print("Failed to create admin account.")
        self.login()

    def create_customer(self, name, email, password):
        customer = Customer(name=name, email=email,
password=password)
        return self.order_repo.create_customer(customer)

    def admin_login(self):
        admin_name = input("Enter admin name: ").strip()
        admin_password = input("Enter admin password: ").strip()

        if self.order_repo.login_admin(admin_name, admin_password):
            self.is_admin = True
            print("Admin login successful!")
            self.admin_menu()
        else:
            print("Invalid admin credentials! Please try again.")
            self.login()

    def customer_login(self):
        print("\n===== Customer Sign In =====")
        email = input("Enter customer email: ")
        password = input("Enter customer password: ")
        try:
            customer = self.order_repo.get_customer_by_email(email)
            if customer and customer.get_password() == password:
                self.logged_in_user = customer  # Make sure this line is being
executed
                print("Customer login successful!")
                self.customer_menu(customer)
            else:
                print("Incorrect password. Please try again.")
        except CustomerNotFoundException as e:
            print(str(e))

    def admin_menu(self):
        while True:
            print("\n===== Admin Dashboard =====")
            print("1. View Customers")
            print("2. View Products")
            print("3. View Customer Orders")
            print("4. Add Product")
            print("5. Logout")

            choice = input("Enter your choice: ")

            if choice == "1":
                self.view_customers()
            elif choice == "2":
                self.view_products()
            elif choice == "3":
                self.view_customer_orders()
            elif choice == "4":
                self.create_product()
            elif choice == "5":
                print("Logging out...")
                self.logged_in_user = None
                self.is_admin = False
                self.login()
            else:
                print("Invalid choice! Please enter a number from 1 to 5.")

    def customer_menu(self,customer):
        while True:
            print("\n===== Customer Dashboard =====")
            print("1. View Products")
            print("2. Add to Cart")
            print("3. Remove from Cart")
            print("4. View Cart")
            print("5. Place Order")
            print("6. View Customer Orders")
            print("7. Cancel Order")
            print("8. Logout")

            choice = input("Enter your choice: ")

            if choice == "1":
                self.view_products()
            elif choice == "2":
                self.add_to_cart()
            elif choice == "3":
                self.remove_from_cart()
            elif choice == "4":
                self.view_cart()
            elif choice == "5":
                self.place_order()
            elif choice == "6":
                self.view_orders()
            elif choice == "7":
                self.cancel_order()
            elif choice == "8":
                print("Logging out...")
                self.logged_in_user = None
                self.is_admin = False
                self.login()
            else:
                print("Invalid choice! Please enter a number from 1 to 8.")

    def view_customers(self):
        # Logic to view customers, e.g., fetching from DB
        customers = self.order_repo.view_all_customers()
        if customers:
            print("Customers:")
            for customer in customers:
                print(f"ID: {customer.get_customer_id()}, Name:
{customer.get_name()}, Email: {customer.get_email()}")
        else:
            print("No customers found.")

    def view_products(self):
        try:
            products = self.order_repo.view_all_products()
            if products:
                print("Products:")
                for product in products:
                    print(
                        f"ID: {product[0]}, Name: {product[1]}, Price:
₹{product[2]:.2f}, Stock Available: {product[3]}")
            else:
                print("No products found.")
        except Exception as e:
            print("Error retrieving products:", e)

    def view_customer_orders(self):
        try:
            customer_id = int(input("Enter customer ID to view orders: "))
            orders = self.order_repo.get_orders_by_customer(customer_id)

            if orders:
                print("Orders:")
                for order in orders:
                    print(
                        f"Order ID: {order['order_id']}, Date:
{order['order_date']}, Total Price: {order['total_price']}, Shipping
Address: {order['shipping_address']}")
                    print("Items:")
                for item in order["items"]:
                    print(f" - {item['name']} (x{item['quantity']}):
{item['price']} each")
            else:
                print("No orders found.")
        except Exception as e:
            print("An error occurred while viewing orders:", e)


    def create_product(self):
        name = input("Enter product name: ")
        price = float(input("Enter product price: "))
        description = input("Enter product description: ")
        stock_quantity = int(input("Enter stock quantity: "))
        product = Product(name=name, price=price,
description=description, stock_quantity=stock_quantity)
        if self.order_repo.create_product(product):
            print("Product created successfully!")
        else:
            print("Failed to create product.")

    def add_to_cart(self):
        try:
            if self.logged_in_user is None:
                print("Please login first.")
                return

            product_id = int(input("Enter product ID: "))
            quantity = int(input("Enter quantity: "))
            customer_id = self.logged_in_user.get_customer_id()  # Fix here:
use logged_in_user

            product = self.order_repo.get_product_by_id(product_id)
            if product is None:
                raise ProductNotFoundException("Product not found.")

            success = self.order_repo.add_to_cart(self.logged_in_user,
product, quantity)  # Fix here: use logged_in_user
            if success:
                print("Product added to cart successfully!")
            else:
                print("Failed to add product to cart.")
        except ProductNotFoundException as e:
            print(f"Error: {e}")
        except CustomerNotFoundException as e:
            print(f"Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    def remove_from_cart(self):
        try:
            if self.logged_in_user is None:  # Fix here: use logged_in_user
                print("Please login first.")
                return

            product_id = int(input("Enter product ID to remove: "))
            customer_id = self.logged_in_user.get_customer_id()  # Fix here:
use logged_in_user

            product = self.order_repo.get_product_by_id(product_id)
            if product is None:
                raise ProductNotFoundException("Product not found.")

            if self.order_repo.remove_from_cart(self.logged_in_user,
product):  # Fix here: use logged_in_user
                print("Product removed from cart successfully!")
            else:
                print("Product not found in cart.")
        except ProductNotFoundException as e:
            print(f"Error: {e}")
        except CustomerNotFoundException as e:
            print(f"Error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    def view_cart(self):
        try:
            if self.logged_in_user is None:  # Fix here: use logged_in_user
                print("Please login first.")
                return

            customer_id = self.logged_in_user.get_customer_id()  # Fix here:
```

```
use logged_in_user
            cart_items = self.order_repo.get_all_from_cart(customer_id)

            if cart_items:
                print("Cart Items:")
                for item in cart_items:
                    print(
                        f"Product ID: {item['product_id']}, Name: {item['name']},
Price: ₹{item['price']}, Quantity: {item['quantity']}")
            else:
                print("Cart is empty.")
        except CustomerNotFoundException:
            print("Error: Customer not found.")
        except Exception as e:
            print("Unexpected error:", e)

    def place_order(self):
        try:
            if self.logged_in_user is None:  # Fix here: use logged_in_user
                print("Please login first.")
                return

            customer_id = self.logged_in_user.get_customer_id()  # Fix here:
use logged_in_user
            shipping_address = input("Enter shipping address: ")
            cart_items = self.order_repo.get_all_from_cart(customer_id)

            if cart_items:
                order_items = [
                    {"product": Product(
                        product_id=item["product_id"],
                        name=item["name"],
                        price=item["price"],
                        description="",
                        stock_quantity=0
                    ), "quantity": item["quantity"]}
                    for item in cart_items
                ]

                if self.order_repo.place_order(customer_id, order_items,
shipping_address):
```

```
                print("Order placed successfully!")
            else:
                print("Failed to place order.")
        else:
            print("Cart is empty.")
    except OrderNotFoundException:
        print("Error: Unable to place order.")
    except Exception as e:
        print("Unexpected error:", e)

def view_orders(self):
    try:
        if self.logged_in_user is None:  # Fix here: use logged_in_user
            print("Please login first.")
            return

        customer_id = self.logged_in_user.get_customer_id()  # Fix here: 
use logged_in_user
        orders = self.order_repo.get_orders_by_customer(customer_id)

        if orders:
            for order in orders:
                print(f"Order ID: {order['order_id']}, Shipping Address: 
{order['shipping_address']}, Total Amount: ₹{order['total_price']}")
        else:
            print("No orders found.")
    except CustomerNotFoundException:
        print("Error: Customer not found.")
    except Exception as e:
        print(f"Unexpected error: {e}")

def cancel_order(self):
    try:
        if self.logged_in_user is None:  # Fix here: use logged_in_user
            print("Please login first.")
            return

        order_id = int(input("Enter order ID to cancel: "))
        if self.order_repo.cancel_order(order_id):
            print("Order canceled successfully.")
        else:
            print("Failed to cancel order.")
    except OrderNotFoundException as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"Unexpected error: {e}")


if __name__ == "__main__":
app = EcomApp()
app.login()
```

6. Testing Module (tests/)

Unit tests to validate correctness and reliability.

File:

- test_order_processor.py

Test Cases Cover:

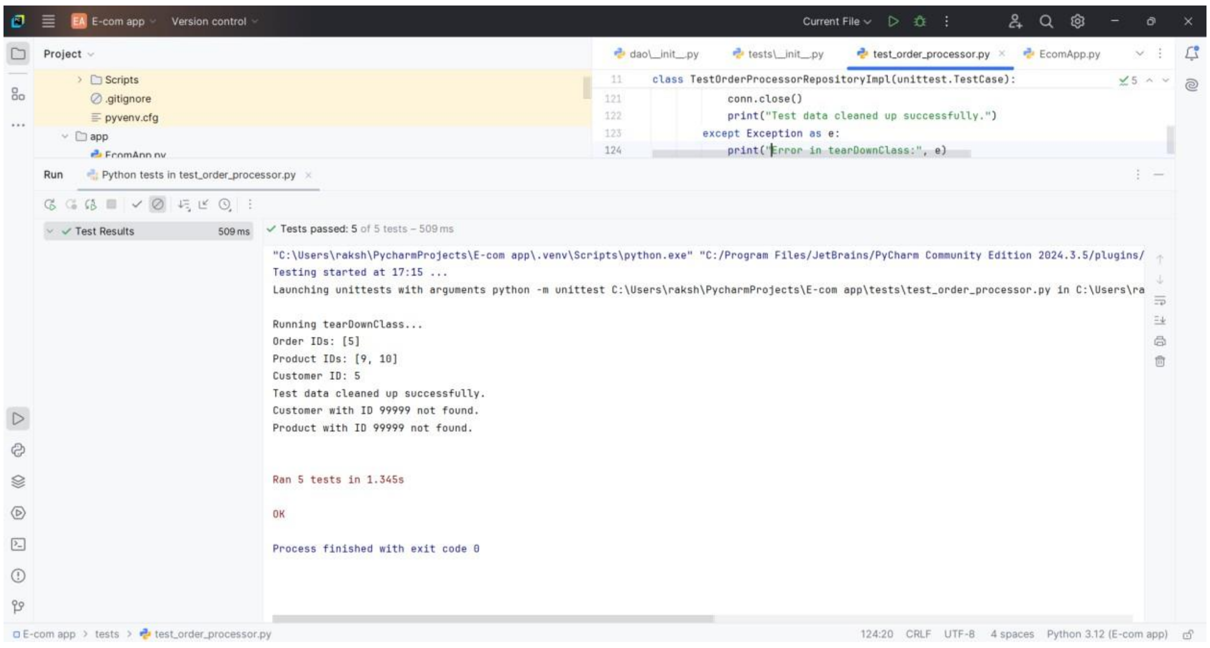
- Product creation
- Cart addition
- Order placement
- Exception handling when customer/product not found

Test_order_processor.py:

```
import unittest

from dao.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl

from entity.Product import Product
from entity.Customer import Customer
from exception.customer_not_found_exception import CustomerNotFoundException
from exception.product_not_found_exception import ProductNotFoundException
import random
import mysql.connector
class TestOrderProcessorRepositoryImpl(unittest.TestCase):
  @classmethod
  def setUpClass(cls):
      cls.repo = OrderProcessorRepositoryImpl()
      cls.customer = Customer(
          name="UnitTest User",
          email=f"testuser_{random.randint(1000, 9999)}@test.com",
          password="test123"
      )
      cls.repo.create_customer(cls.customer)
      cls.test_customer_id = cls.customer.get_customer_id()



      cls.product = Product(
          name="Test Product",
          price=99.99,
          description="Unit test product",
          stock_quantity=50
      )
      cls.repo.create_product(cls.product)
```

```
        cls.test_product_ids = [cls.product.get_product_id()]
        cls.test_order_ids = []

    def test_1_create_product_success(self):
        new_product = Product(
            name="New Test Product",
            price=49.99,
            description="Another test product",
            stock_quantity=30
        )
        result = self.repo.create_product(new_product)
        self.assertTrue(result)
        self.assertIsNotNone(new_product.get_product_id())



        type(self).test_product_ids.append(new_product.get_product_id())

    def test_2_add_to_cart_success(self):
        result = self.repo.add_to_cart(self.customer, self.product, quantity=2)
        self.assertTrue(result)

    def test_3_place_order_success(self):
        cart_items = [{"product": self.product, "quantity": 2}]
        result = self.repo.place_order(self.customer, cart_items, "123 Test Lane")
        self.assertTrue(result)
        conn = mysql.connector.connect(
            host='localhost',
            user='root',
            password='rakshi430',
            database='ecomm_db'
        )
        cursor = conn.cursor()
        cursor.execute("SELECT MAX(order_id) FROM orders WHERE customer_id = %s", (self.test_customer_id,))
        order_id = cursor.fetchone()[0]
        if order_id:
            type(self).test_order_ids.append(order_id)
        cursor.close()
        conn.close()

    def test_4_customer_not_found_exception(self):
        fake_customer = Customer(name="Fake", email="fake@test.com", password="fake123")
        fake_customer.set_customer_id(99999)
        with self.assertRaises(CustomerNotFoundException):
            self.repo.add_to_cart(fake_customer, self.product, quantity=1)

    def test_5_product_not_found_exception(self):
        fake_product = Product(name="Ghost", price=0.0, description="Ghost", stock_quantity=0)
        fake_product.set_product_id(99999)
        with self.assertRaises(ProductNotFoundException):
            self.repo.add_to_cart(self.customer, fake_product, quantity=1)

    @classmethod
    def tearDownClass(cls):
        print("Running tearDownClass...")
        print("Order IDs:", cls.test_order_ids)
        print("Product IDs:", cls.test_product_ids)
        print(“Customer ID:", cls.test_customer_id)

        try:
            conn = mysql.connector.connect(
                host='localhost',
                user='root',
                password='rakshi430',
                database='ecomm_db'
            )
            cursor = conn.cursor()



            for order_id in cls.test_order_ids:
                cursor.execute("DELETE FROM order_items WHERE order_id = %s",
(order_id,))
                cursor.execute("DELETE FROM orders WHERE order_id = %s",
(order_id,))
```

```
            for product_id in cls.test_product_ids:
                cursor.execute("DELETE FROM cart WHERE customer_id = %s AND product_id = %s",
                               (cls.test_customer_id, product_id))


            for product_id in cls.test_product_ids:
                cursor.execute("DELETE FROM products WHERE product_id = %s", (product_id,))


            cursor.execute("DELETE FROM customers WHERE customer_id = %s", (cls.test_customer_id,))

            conn.commit()
            cursor.close()
            conn.close()
            print("Test data cleaned up successfully.")
        except Exception as e:
            print(“ Error in tearDownClass:", e)


if __name___== "__main__":
   unittest.main()
```



```python
class TestOrderProcessorRepositoryImpl(unittest.TestCase):
            conn.close()
            print("Test data cleaned up successfully.")
        except Exception as e:
            print("Error in tearDownClass:", e)
```

Tests passed: 5 of 5 tests – 509 ms

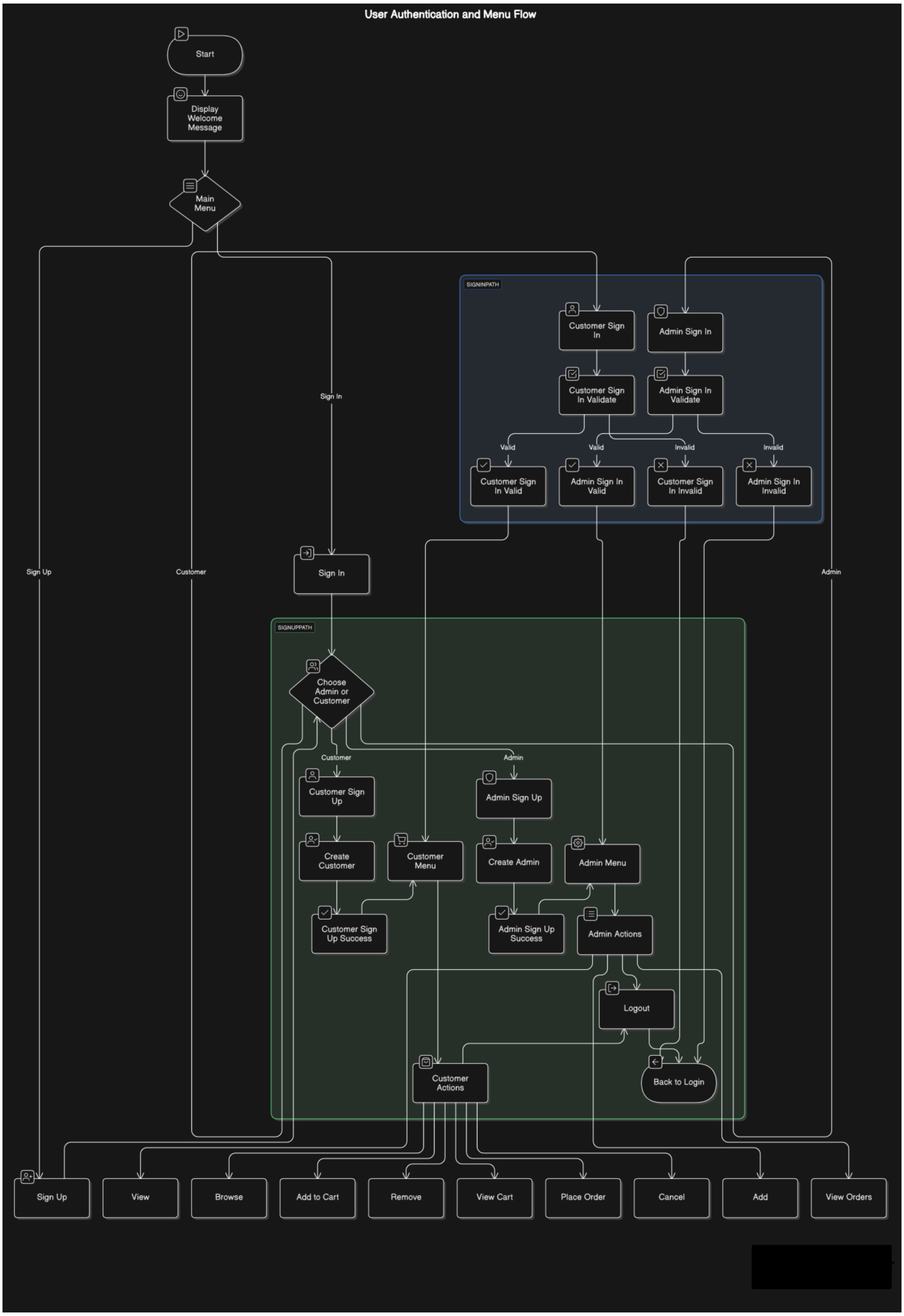
```
"C:\Users\raksh\PycharmProjects\E-com app\.venv\Scripts\python.exe" "C:/Program Files/JetBrains/PyCharm Community Edition 2024.3.5/plugins/
Testing started at 17:15 ...
Launching unittests with arguments python -m unittest C:\Users\raksh\PycharmProjects\E-com app\tests\test_order_processor.py in C:\Users\ra

Running tearDownClass...
Order IDs: [5]
Product IDs: [9, 10]
Customer ID: 5
Test data cleaned up successfully.
Customer with ID 99999 not found.
Product with ID 99999 not found.


Ran 5 tests in 1.345s

OK

Process finished with exit code 0
```

5. Technologies Used

- Programming Language- Python
- Database- **MySQL Workbench**.
- Database Connectivity- MySQL Connector/Python
- Software Development Concepts- Object-Oriented Programming (OOP) and Exception Handling
- Testing Framework- unittest (Python Standard Library)
- IDE / Tools- PyCharm

6. Flow Chart:

A flowchart is a graphical representation of the logical flow of processes or steps in a system. It uses symbols like ovals (start/end), rectangles (processes), diamonds (decisions), and arrows (flow direction) to show how data or control moves through the system.


User Authentication and Menu Flow
Start
Display Welcome Message
Main Menu
Sign Up
Customer
Sign In
Admin
SIGNINPATH
Customer Sign In
Admin Sign In
Customer Sign In Validate
Admin Sign In Validate
Valid
Valid
Invalid
Invalid
Customer Sign In Valid
Admin Sign In Valid
Customer Sign In Invalid
Admin Sign In Invalid
Sign In
SIGNUPPATH
Choose Admin or Customer
Customer
Admin
Customer Sign Up
Admin Sign Up
Create Customer
Customer Menu
Create Admin
Admin Menu
Customer Sign Up Success
Admin Sign Up Success
Admin Actions
Logout
Customer Actions
Back to Login
Sign Up
View
Browse
Add to Cart
Remove
View Cart
Place Order
Cancel
Add
View Orders

[redacted]

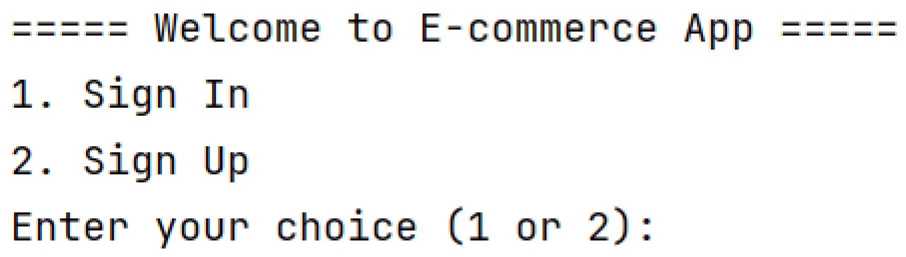
7. Output:

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.

Work flow:

- **Main menu**
  1. **Sign in**
  2. **Sign up**

Terminal output:

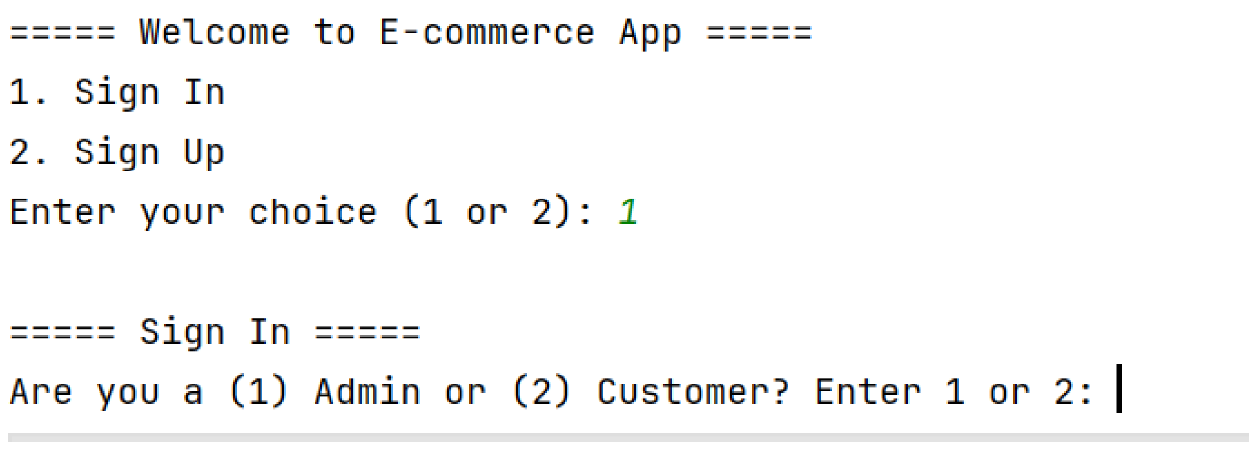


```
===== Welcome to E-commerce App =====
1. Sign In
2. Sign Up
Enter your choice (1 or 2):
```

➢ **Sign in as admin(1)**

1. **As admin**
2. **As customer**

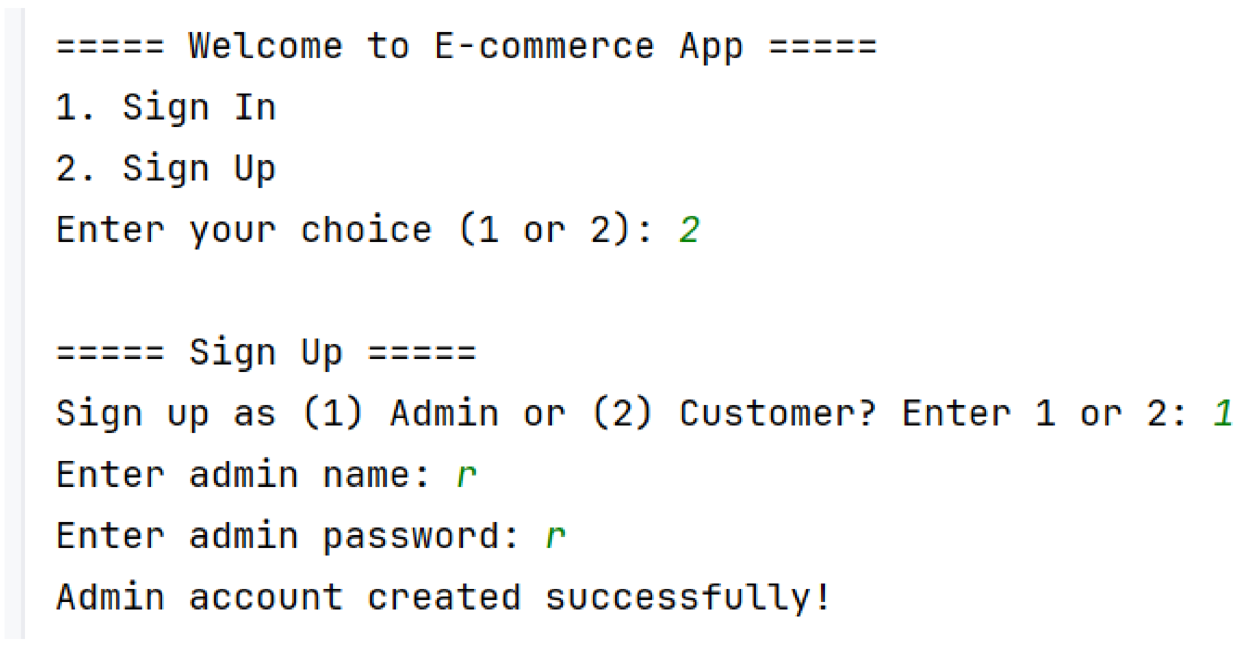
Terminal output:



```
===== Welcome to E-commerce App =====
1. Sign In
2. Sign Up
Enter your choice (1 or 2): 1

===== Sign In =====
Are you a (1) Admin or (2) Customer? Enter 1 or 2: 
```

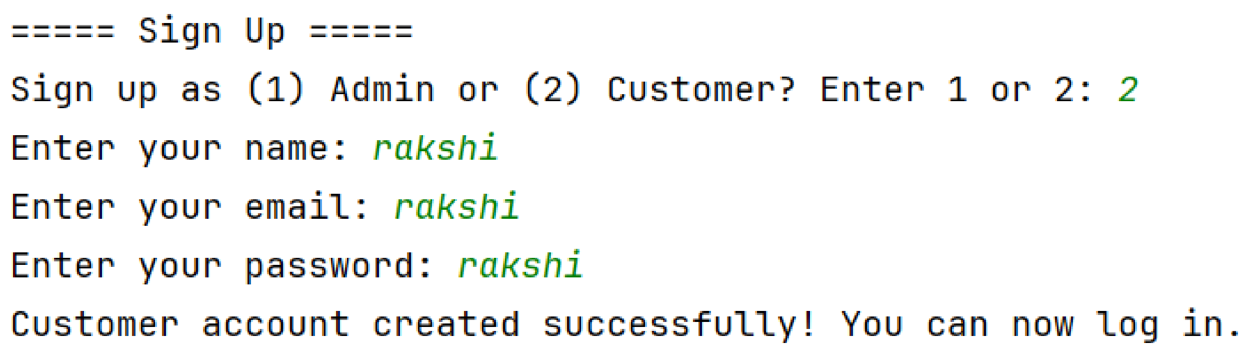
➢ **Sign up**

1. **Sign up As admin**



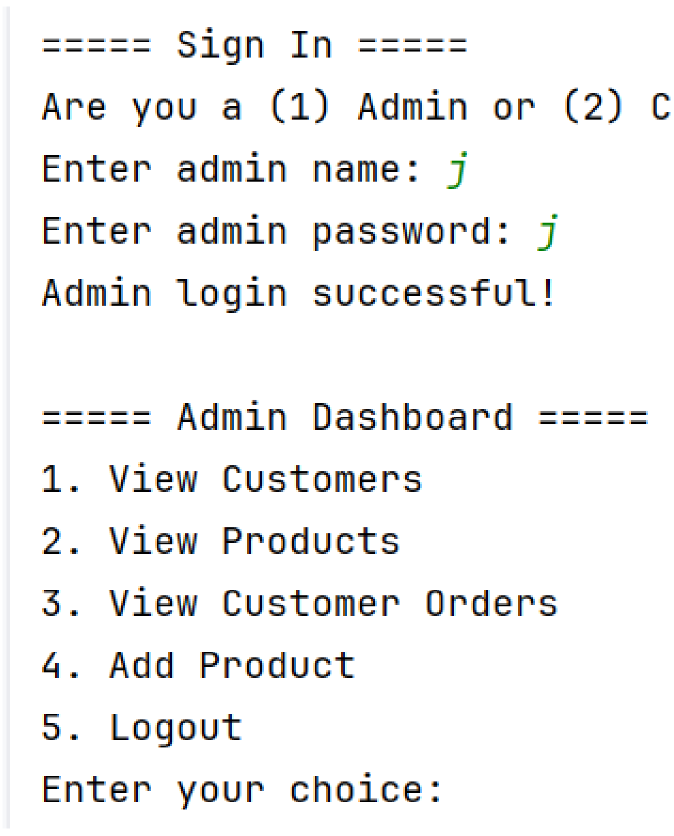
```
===== Welcome to E-commerce App =====
1. Sign In
2. Sign Up
Enter your choice (1 or 2): 2

===== Sign Up =====
Sign up as (1) Admin or (2) Customer? Enter 1 or 2: 1
Enter admin name: r
Enter admin password: r
Admin account created successfully!
```

2. **Sign up As customer**



```
===== Sign Up =====
Sign up as (1) Admin or (2) Customer? Enter 1 or 2: 2
Enter your name: rakshi
Enter your email: rakshi
Enter your password: rakshi
Customer account created successfully! You can now log in.
```

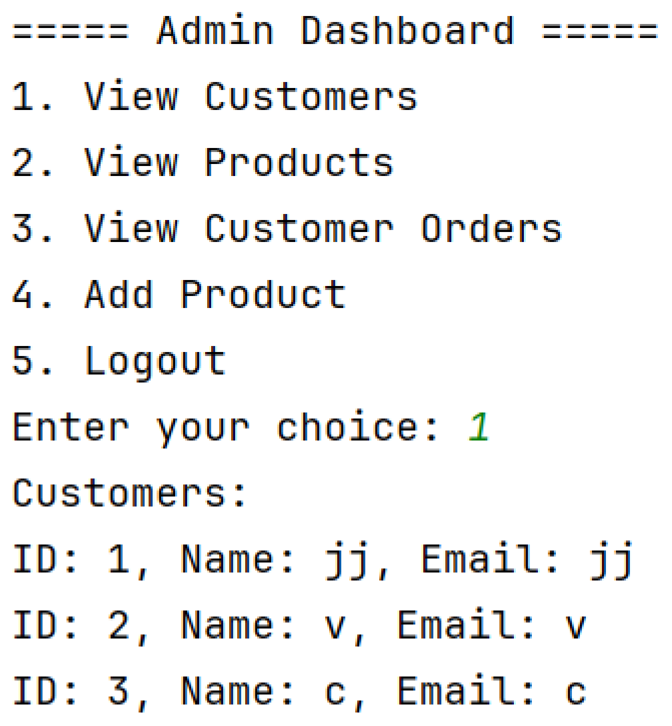
➢ **Sign in as admin:**



```
===== Sign In =====
Are you a (1) Admin or (2) Customer? Enter 1 or 2: 1
Enter admin name: j
Enter admin password: j
Admin login successful!

===== Admin Dashboard =====
1. View Customers
2. View Products
3. View Customer Orders
4. Add Product
5. Logout
Enter your choice:
```

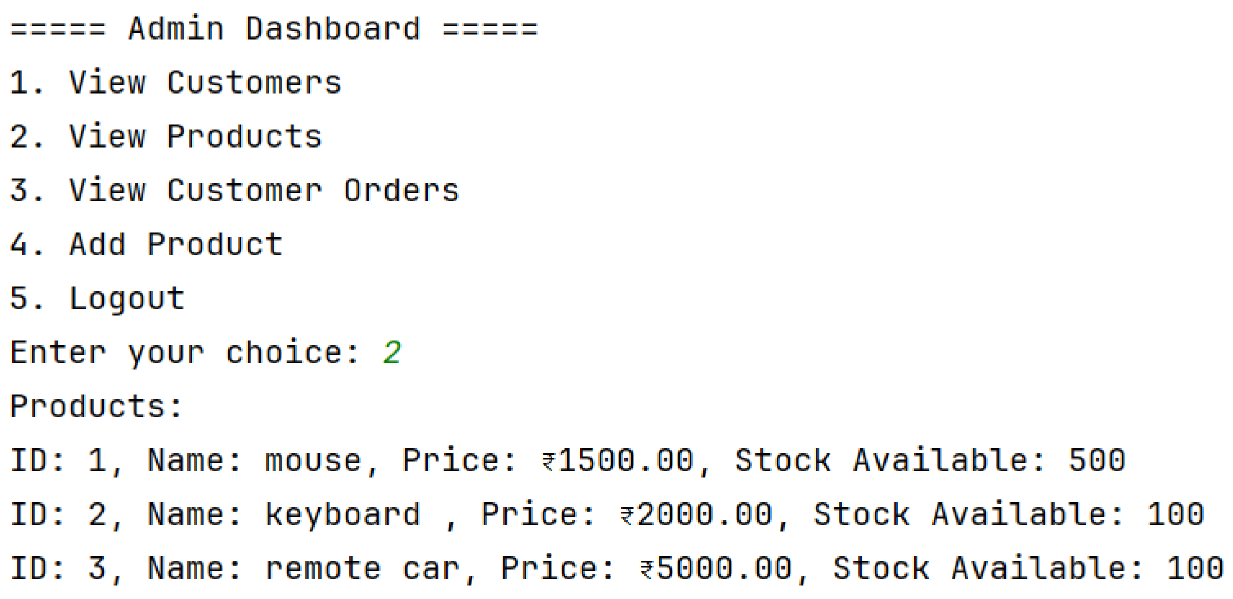
Admin operations:

1. **View customer**



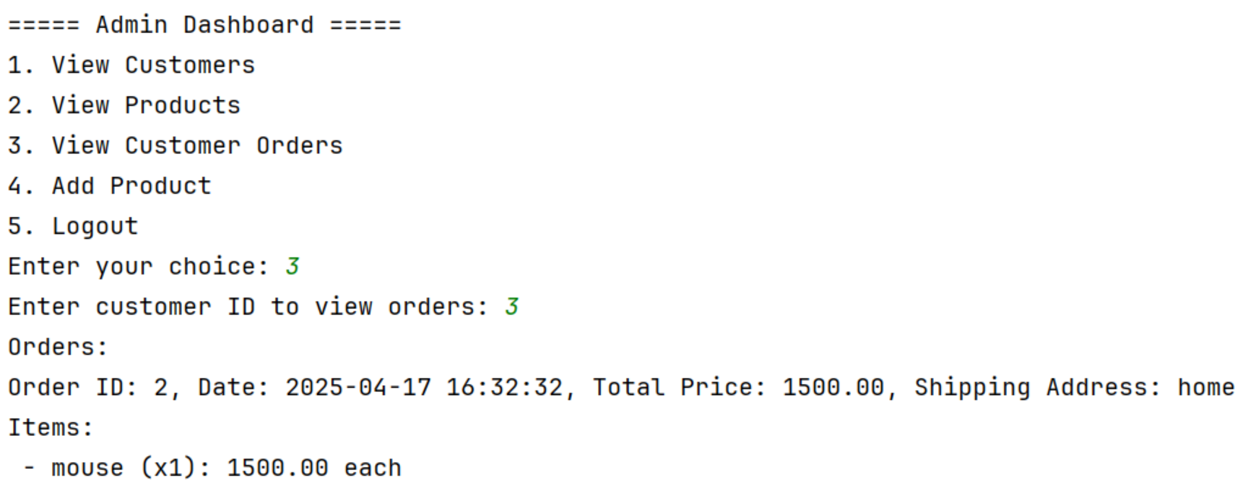
```
===== Admin Dashboard =====
1. View Customers
2. View Products
3. View Customer Orders
4. Add Product
5. Logout
Enter your choice: 1
Customers:
ID: 1, Name: jj, Email: jj
ID: 2, Name: v, Email: v
ID: 3, Name: c, Email: c
```

2. **View products**



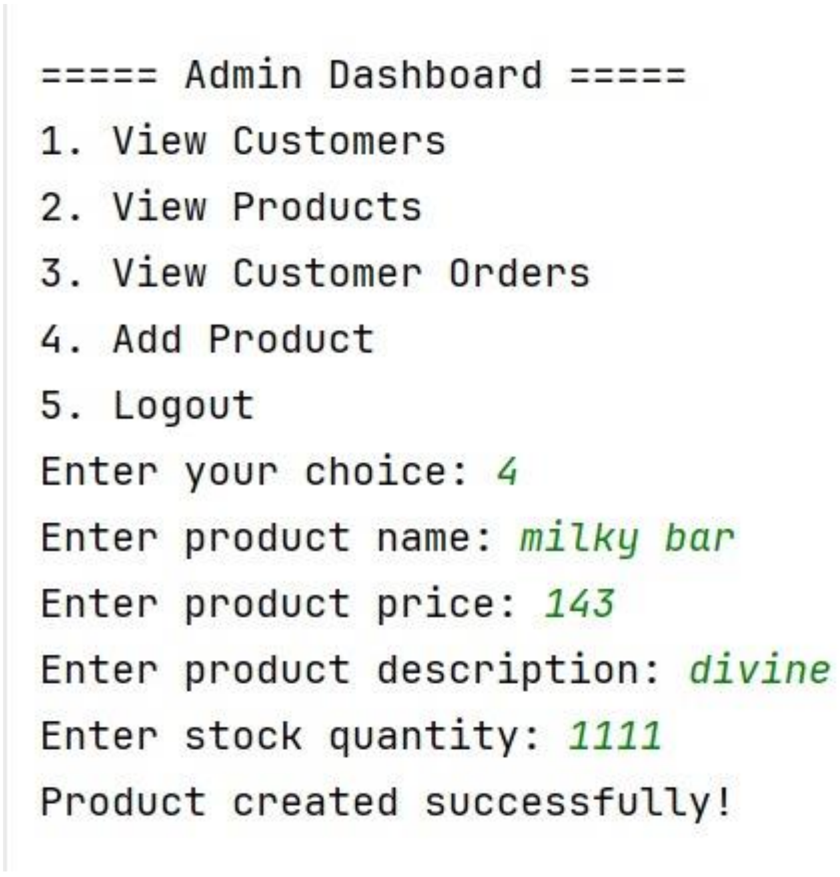
```
===== Admin Dashboard =====
1. View Customers
2. View Products
3. View Customer Orders
4. Add Product
5. Logout
Enter your choice: 2
Products:
ID: 1, Name: mouse, Price: ₹1500.00, Stock Available: 500
ID: 2, Name: keyboard , Price: ₹2000.00, Stock Available: 100
ID: 3, Name: remote car, Price: ₹5000.00, Stock Available: 100
```

3. View customer order



```
===== Admin Dashboard =====
1. View Customers
2. View Products
3. View Customer Orders
4. Add Product
5. Logout
Enter your choice: 3
Enter customer ID to view orders: 3
Orders:
Order ID: 2, Date: 2025-04-17 16:32:32, Total Price: 1500.00, Shipping Address: home
Items:
 - mouse (x1): 1500.00 each
```

4. Add product

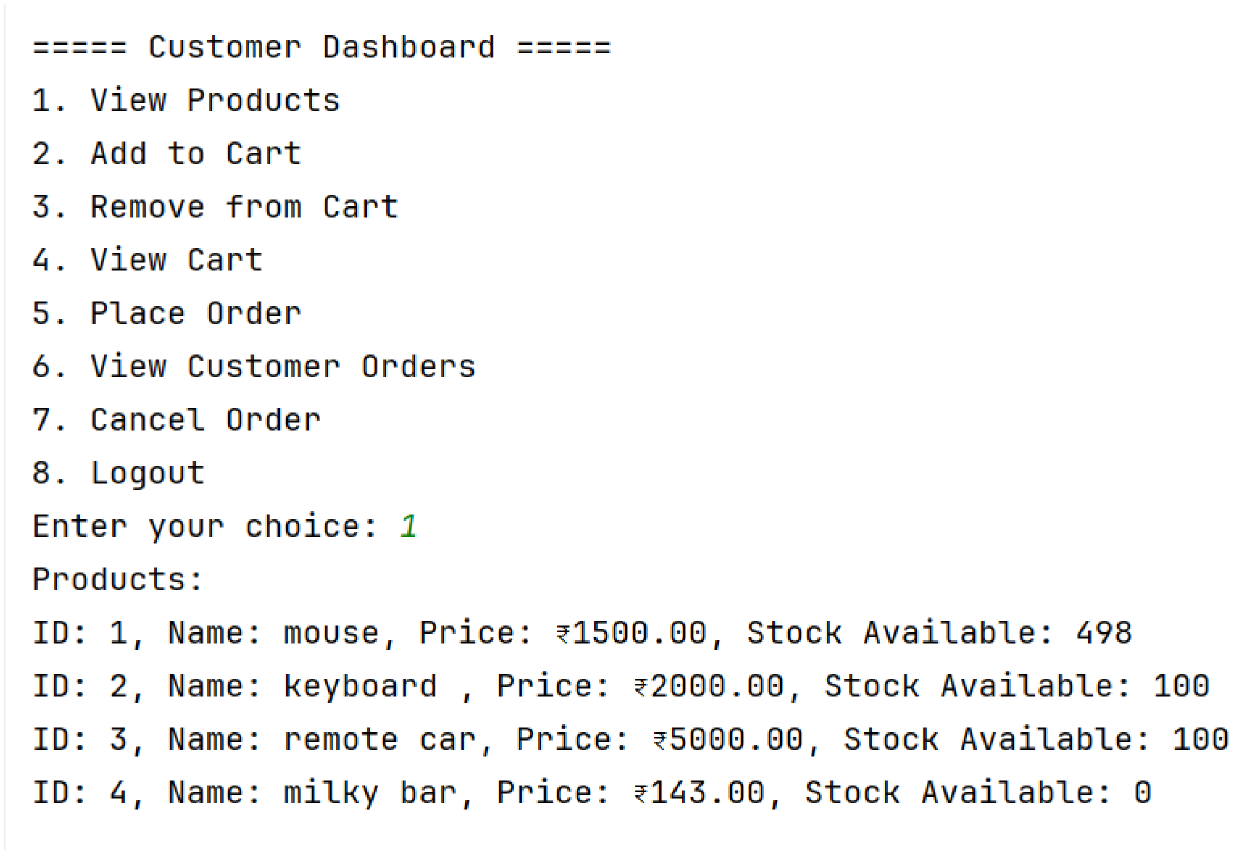


```
===== Admin Dashboard =====
1. View Customers
2. View Products
3. View Customer Orders
4. Add Product
5. Logout
Enter your choice: 4
Enter product name: milky bar
Enter product price: 143
Enter product description: divine
Enter stock quantity: 1111
Product created successfully!
```

➢ **Sign in as customer (2)**

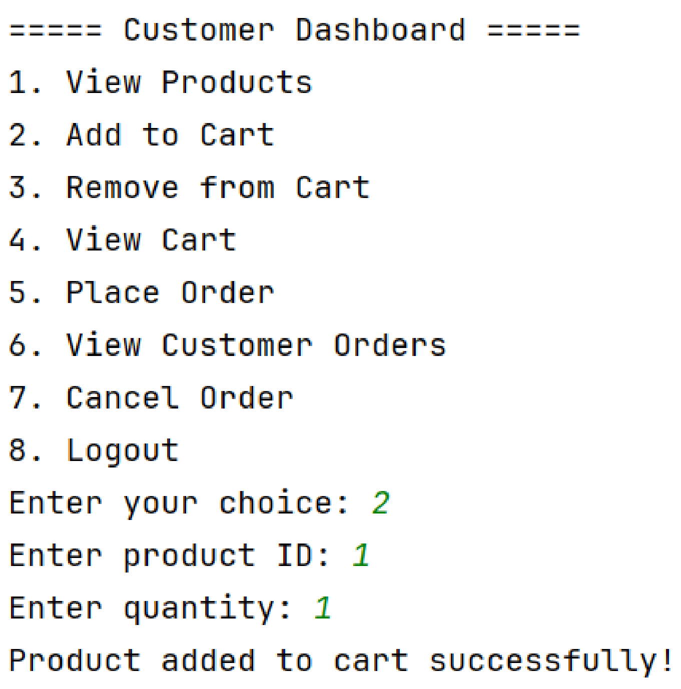
➢ **Customer operations:**

1. View products



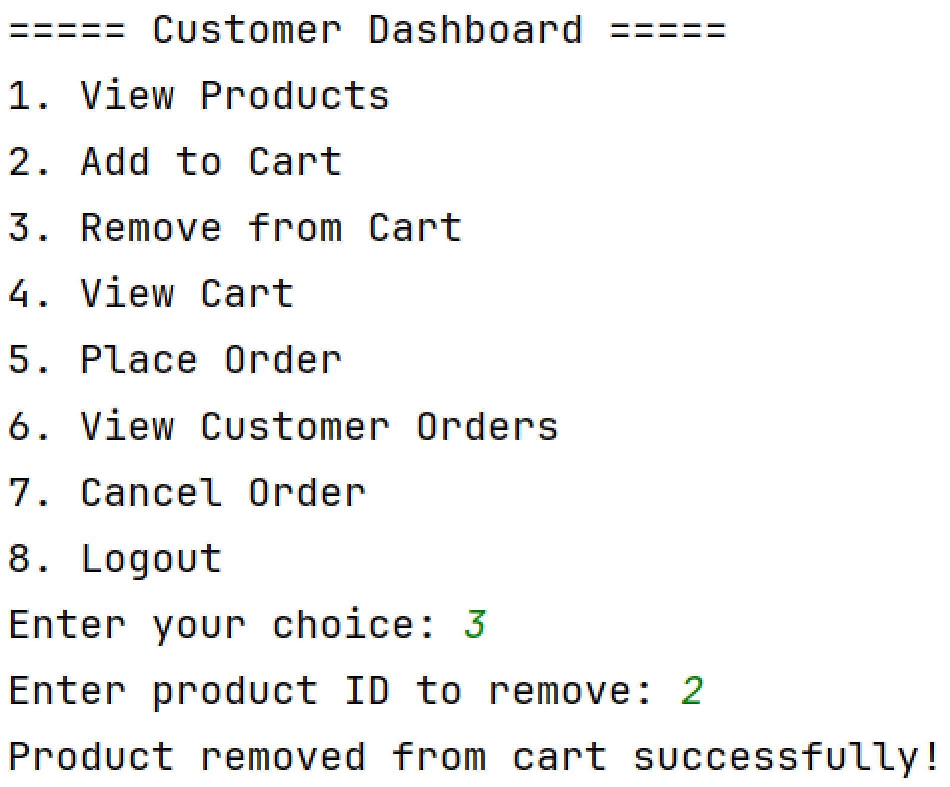
```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 1
Products:
ID: 1, Name: mouse, Price: ₹1500.00, Stock Available: 498
ID: 2, Name: keyboard , Price: ₹2000.00, Stock Available: 100
ID: 3, Name: remote car, Price: ₹5000.00, Stock Available: 100
ID: 4, Name: milky bar, Price: ₹143.00, Stock Available: 0
```

2. Add to cart



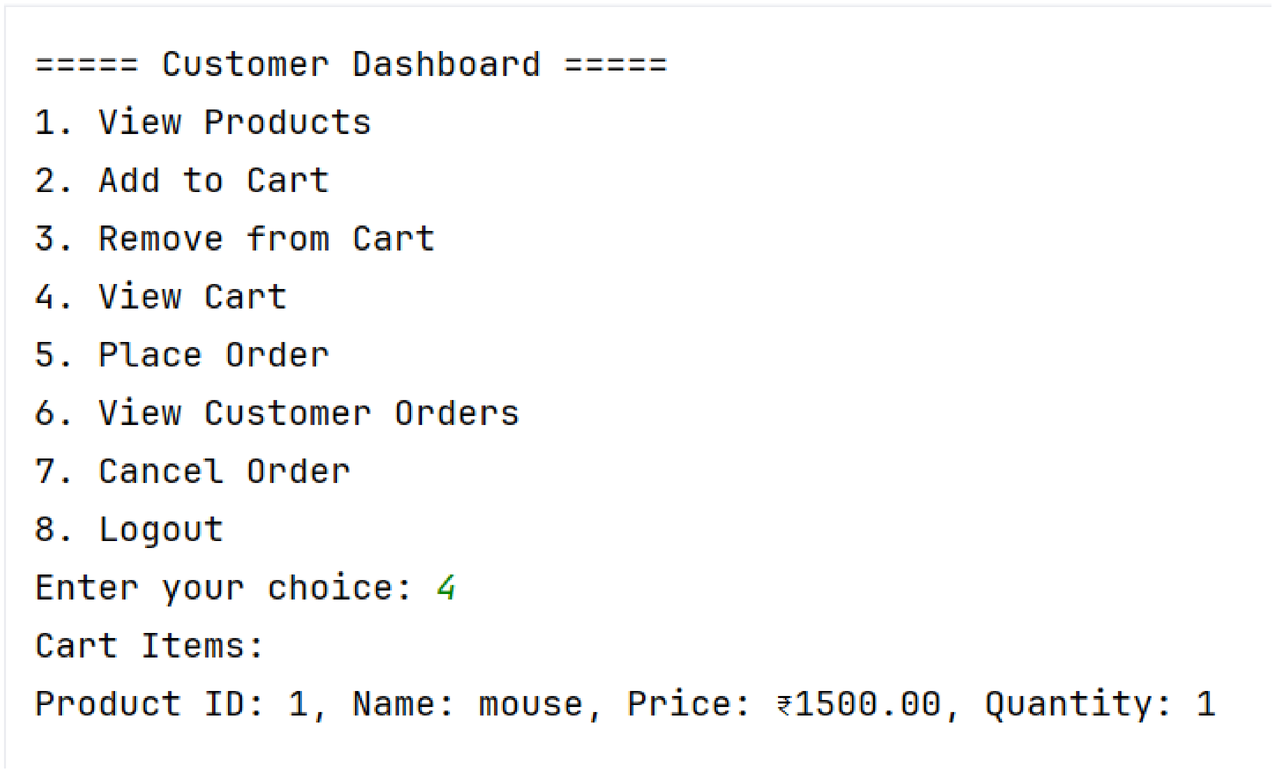
```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 2
Enter product ID: 1
Enter quantity: 1
Product added to cart successfully!
```

3. **Remove from cart**



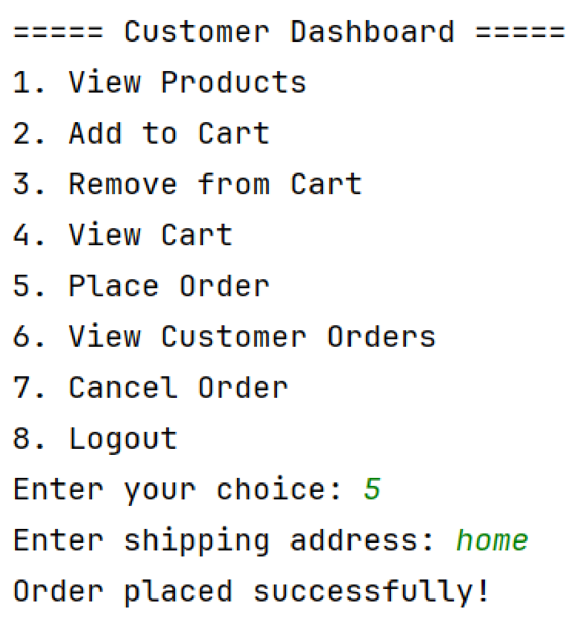
```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 3
Enter product ID to remove: 2
Product removed from cart successfully!
```

4. **View cart**



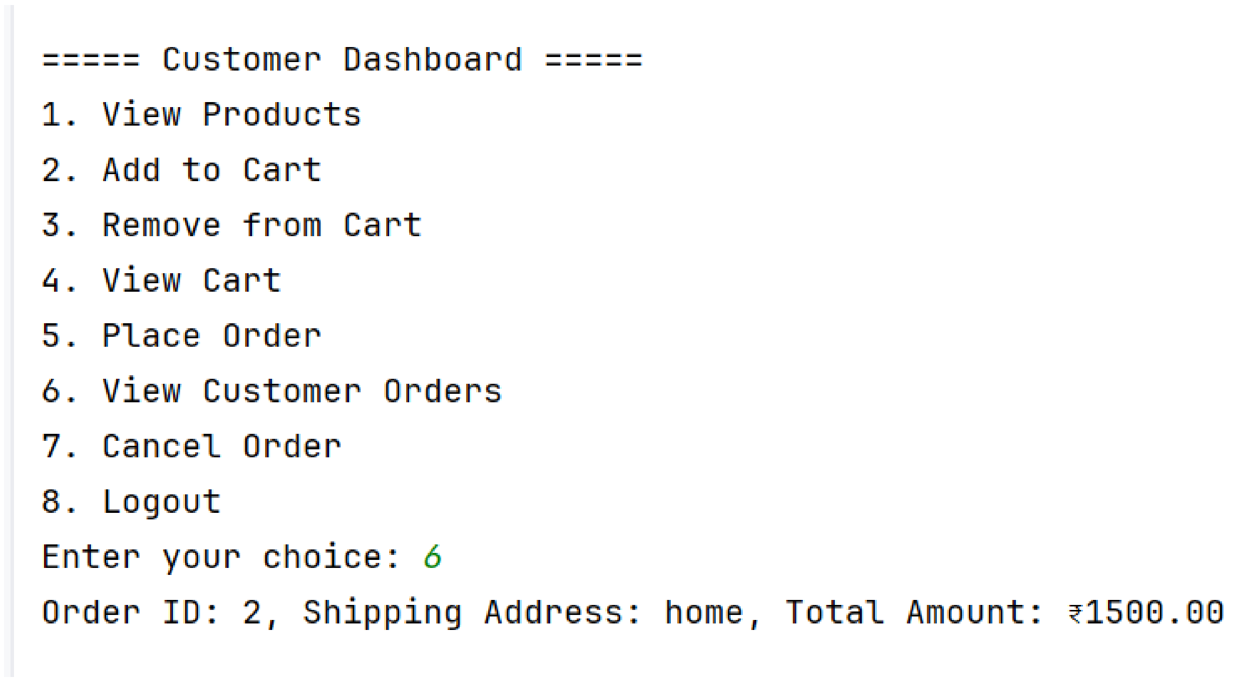
```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 4
Cart Items:
Product ID: 1, Name: mouse, Price: ₹1500.00, Quantity: 1
```

5. Place order



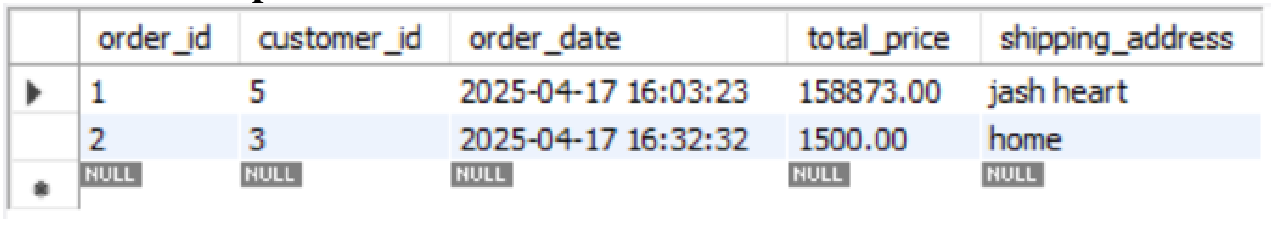
```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 5
Enter shipping address: home
Order placed successfully!
```

6. View customer order



```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 6
Order ID: 2, Shipping Address: home, Total Amount: ₹1500.00
```

Data base output:

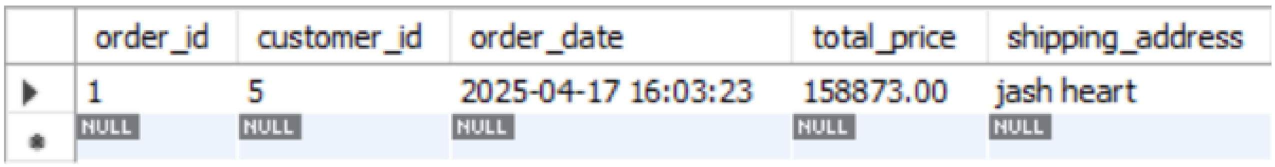


	order_id	customer_id	order_date	total_price	shipping_address
▶	1	5	2025-04-17 16:03:23	158873.00	jash heart
	2	3	2025-04-17 16:32:32	1500.00	home
*	NULL	NULL	NULL	NULL	NULL

7. Cancel order

```
===== Customer Dashboard =====
1. View Products
2. Add to Cart
3. Remove from Cart
4. View Cart
5. Place Order
6. View Customer Orders
7. Cancel Order
8. Logout
Enter your choice: 7
Enter order ID to cancel: 2
Order canceled successfully.
```

Data base output:



	order_id	customer_id	order_date	total_price	shipping_address
▶	1	5	2025-04-17 16:03:23	158873.00	jash heart
*	NULL	NULL	NULL	NULL	NULL

8. Future Enhancements

The Ecommerce Application has vast potential for future enhancements that can significantly improve user experience, business efficiency, and scalability. One of the key areas for improvement is the shopping cart system, which can be enhanced to support session persistence, discount coupons, and cart expiration mechanisms. In terms of security, implementing advanced user authentication with encrypted passwords, OTP verification, and role-based access control would add a strong layer of protection. Integrating a payment gateway such as Razorpay or Stripe would allow secure and seamless online transactions. Additionally, a shipping and logistics module could be introduced to track delivery status and sync with third-party couriers.

Another enhancement would be developing a product recommendation engine using machine learning to offer personalized suggestions to users based on their browsing and purchase history. For admins, an interactive dashboard could be created to provide insights into sales, revenue, inventory, and customer behavior through visual reports. Customers could also be given the ability to rate and review products, enhancing transparency and trust. Supporting multiple languages and currencies would allow the application to serve a global user base, and creating a mobile app would provide more accessibility to customers on smartphones.

9. Conclusion

The Ecommerce Application Project successfully demonstrates the development of a fully functional backend system using core principles of object-oriented programming, MySQL database integration, exception handling, and unit testing. Through the implementation of modules such as customer management, product management, cart operations, and order processing, this project replicates the foundational operations of a real-world ecommerce platform. It effectively showcases the interaction between frontend input, backend logic, and persistent data storage.By following a modular and layered architecture — including the use of entity classes, DAO interfaces, utility handlers, and custom exceptions — the project ensures scalability, maintainability, and code reusability. The use of SQL for structured data management, along with robust error handling and unit testing, enhances the reliability and stability of the application.In conclusion, this project not only meets academic and practical requirements but also lays a strong foundation for real-time ecommerce solutions. It can be further extended with advanced features like payment integration, analytics, and enhanced security to make it production-ready.