Case Study on Ecommerce Application

BY SHREYASI REJA

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.

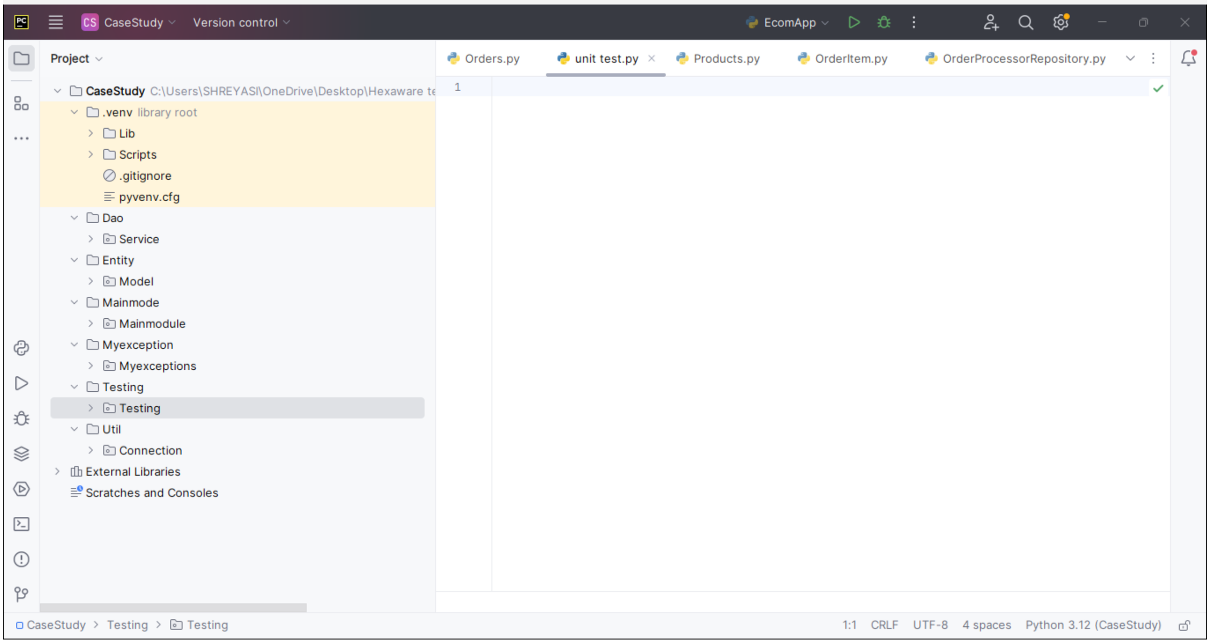
util

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.



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

Schema Design:

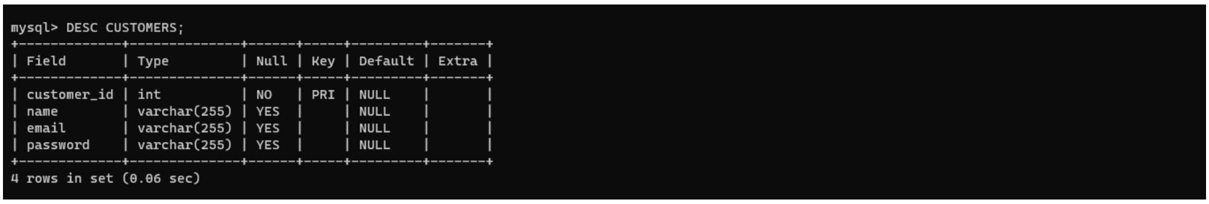
1. customers table:

customer_id (Primary Key)

name

email

password



```
mysql> DESC CUSTOMERS;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| customer_id | int          | NO   | PRI | NULL    |       |
| name        | varchar(255) | YES  |     | NULL    |       |
| email       | varchar(255) | YES  |     | NULL    |       |
| password    | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
4 rows in set (0.06 sec)
```

2. products table:

⍰

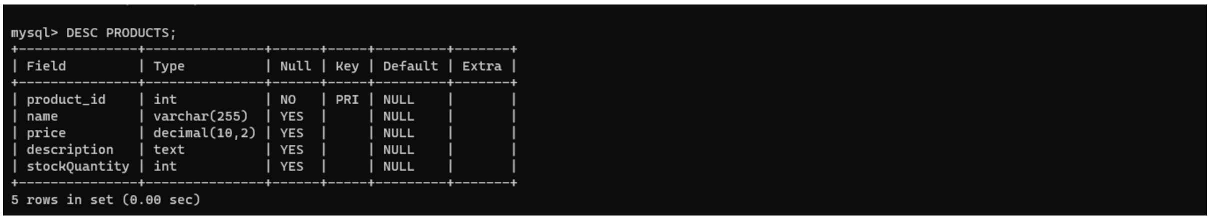
product_id (Primary Key)

name

price

description

stockQuantity



```
mysql> DESC PRODUCTS;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| product_id    | int           | NO   | PRI | NULL    |       |
| name          | varchar(255)  | YES  |     | NULL    |       |
| price         | decimal(10,2) | YES  |     | NULL    |       |
| description   | text          | YES  |     | NULL    |       |
| stockQuantity | int           | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

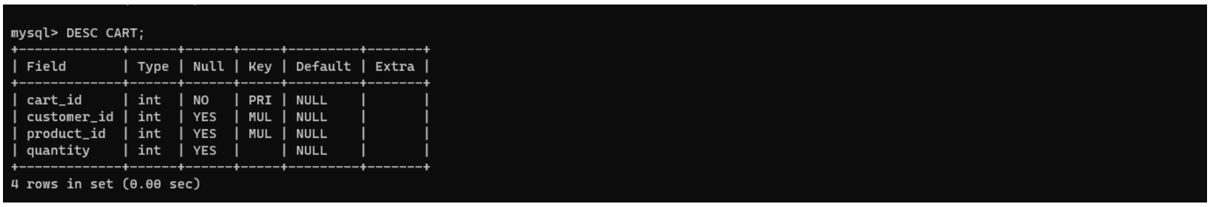
3. cart table:

cart_id (Primary Key)

customer_id (Foreign Key)

product_id (Foreign Key)

quantity



```
mysql> DESC CART;
+-------------+------+------+-----+---------+-------+
| Field       | Type | Null | Key | Default | Extra |
+-------------+------+------+-----+---------+-------+
| cart_id     | int  | NO   | PRI | NULL    |       |
| customer_id | int  | YES  | MUL | NULL    |       |
| product_id  | int  | YES  | MUL | NULL    |       |
| quantity    | int  | YES  |     | NULL    |       |
+-------------+------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4. orders table:

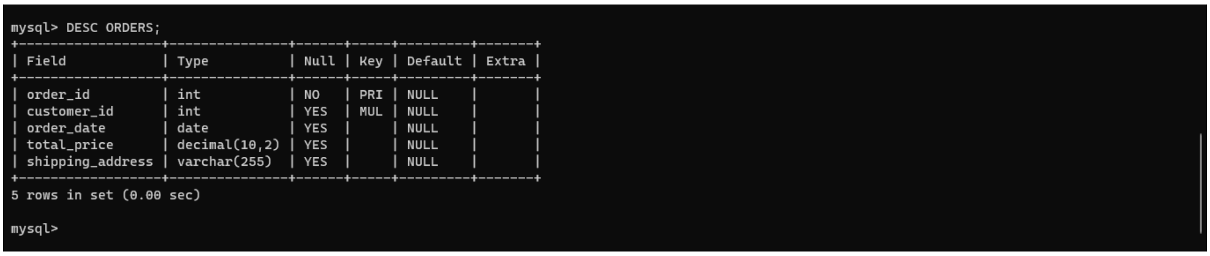
order_id (Primary Key)

customer_id (Foreign Key)

order_date

total_price

shipping_address



```
mysql> DESC ORDERS;
+------------------+---------------+------+-----+---------+-------+
| Field            | Type          | Null | Key | Default | Extra |
+------------------+---------------+------+-----+---------+-------+
| order_id         | int           | NO   | PRI | NULL    |       |
| customer_id      | int           | YES  | MUL | NULL    |       |
| order_date       | date          | YES  |     | NULL    |       |
| total_price      | decimal(10,2) | YES  |     | NULL    |       |
| shipping_address | varchar(255)  | YES  |     | NULL    |       |
+------------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

5. order_items table (to store order details):

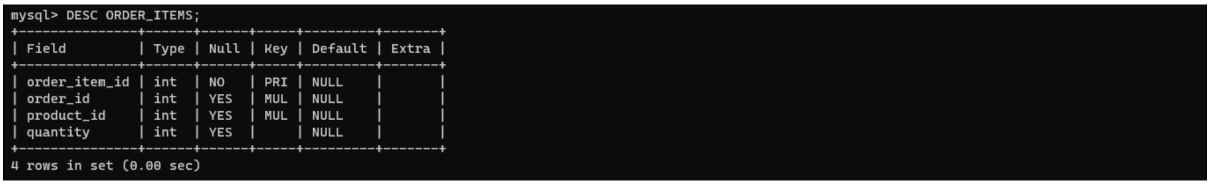
⍰

order_item_id (Primary Key)

order_id (Foreign Key)

product_id (Foreign Key)

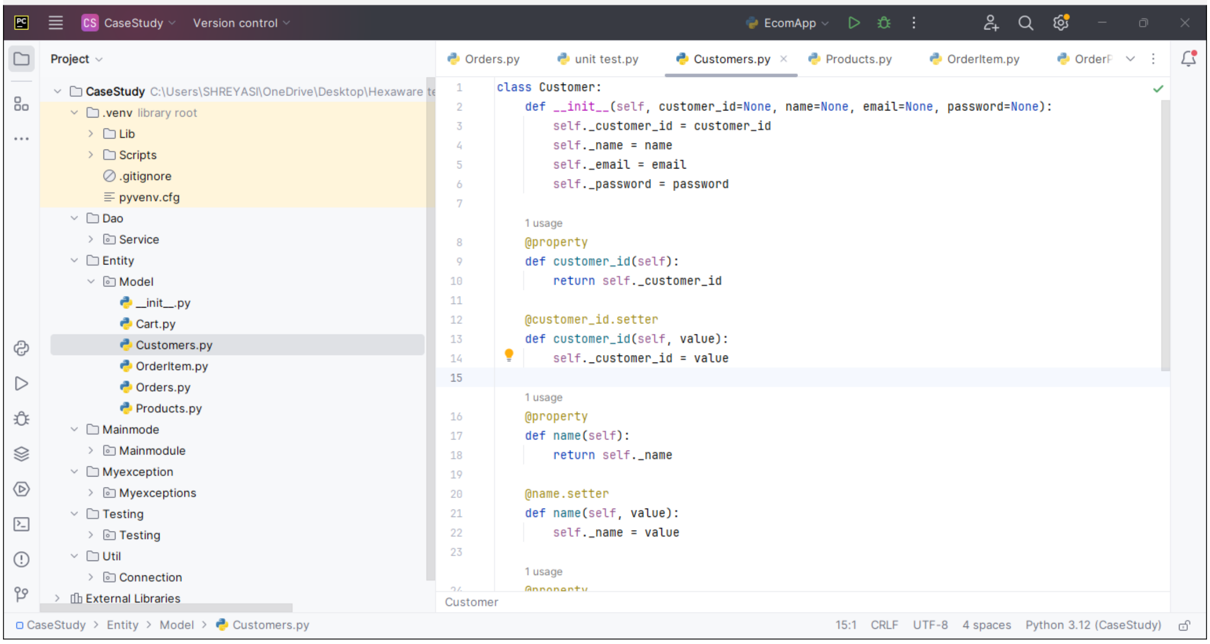
quantity



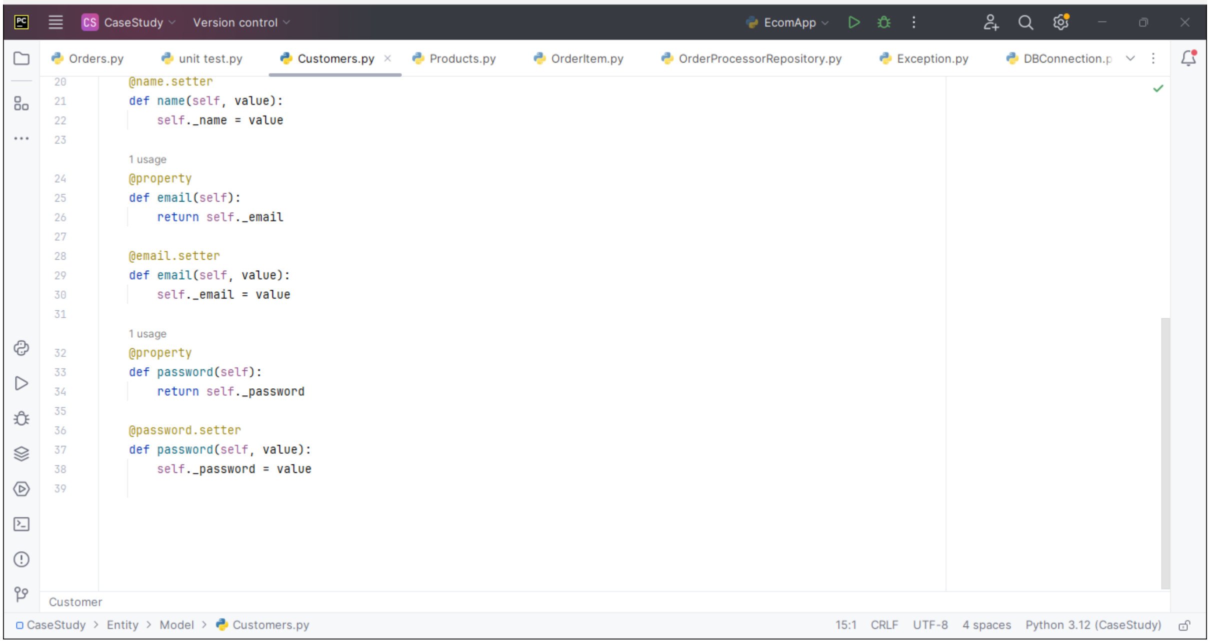
```
mysql> DESC ORDER_ITEMS;
+---------------+------+------+-----+---------+-------+
| Field         | Type | Null | Key | Default | Extra |
+---------------+------+------+-----+---------+-------+
| order_item_id | int  | NO   | PRI | NULL    |       |
| order_id      | int  | YES  | MUL | NULL    |       |
| product_id    | int  | YES  | MUL | NULL    |       |
| quantity      | int  | YES  |     | NULL    |       |
+---------------+------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

Customer:-

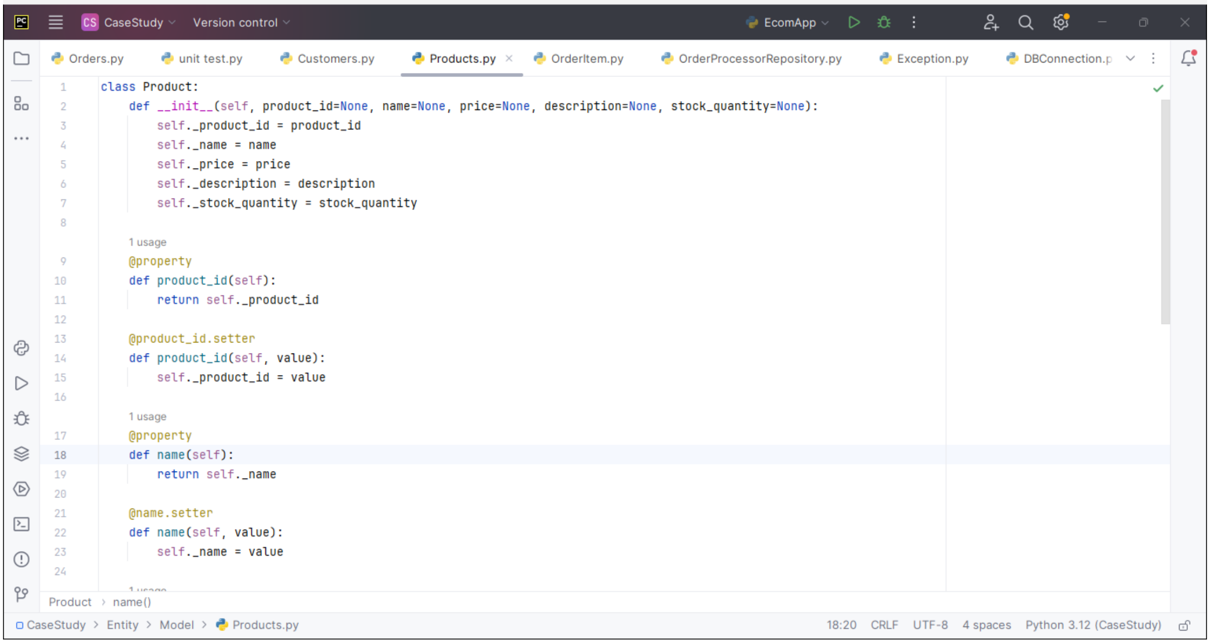


```
class Customer:
    def __init__(self, customer_id=None, name=None, email=None, password=None):
        self._customer_id = customer_id
        self._name = name
        self._email = email
        self._password = password

    @property
    def customer_id(self):
        return self._customer_id

    @customer_id.setter
    def customer_id(self, value):
        self._customer_id = value

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
```

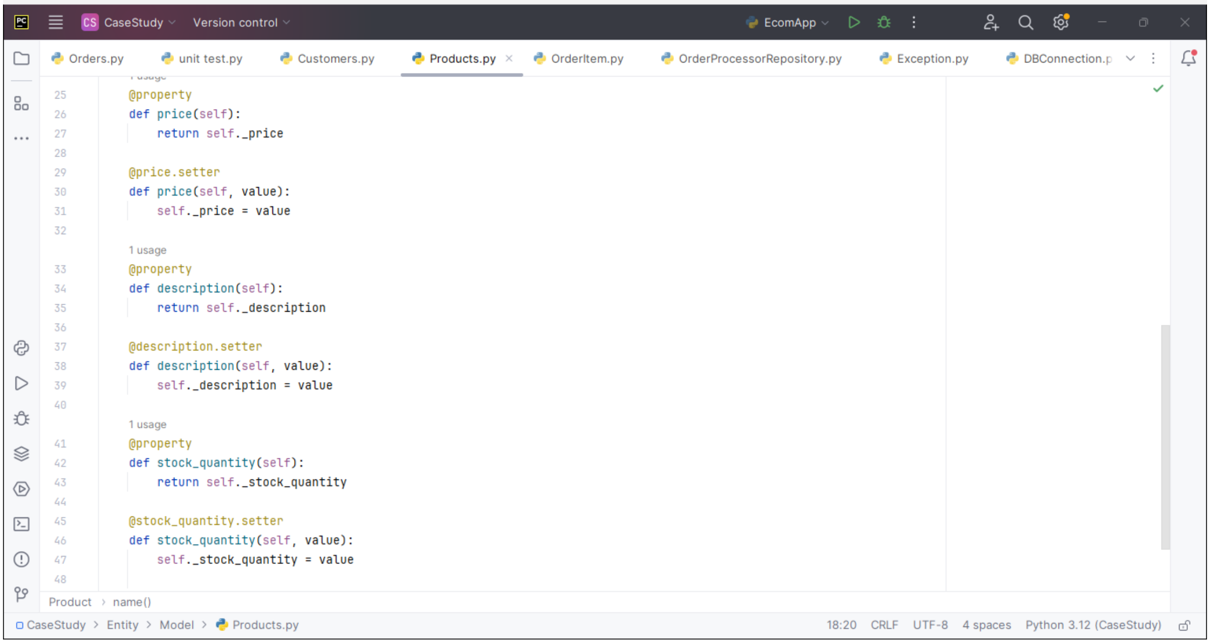


```python
    @name.setter
    def name(self, value):
        self._name = value

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        self._email = value

    @property
    def password(self):
        return self._password

    @password.setter
    def password(self, value):
        self._password = value

```

Product:-

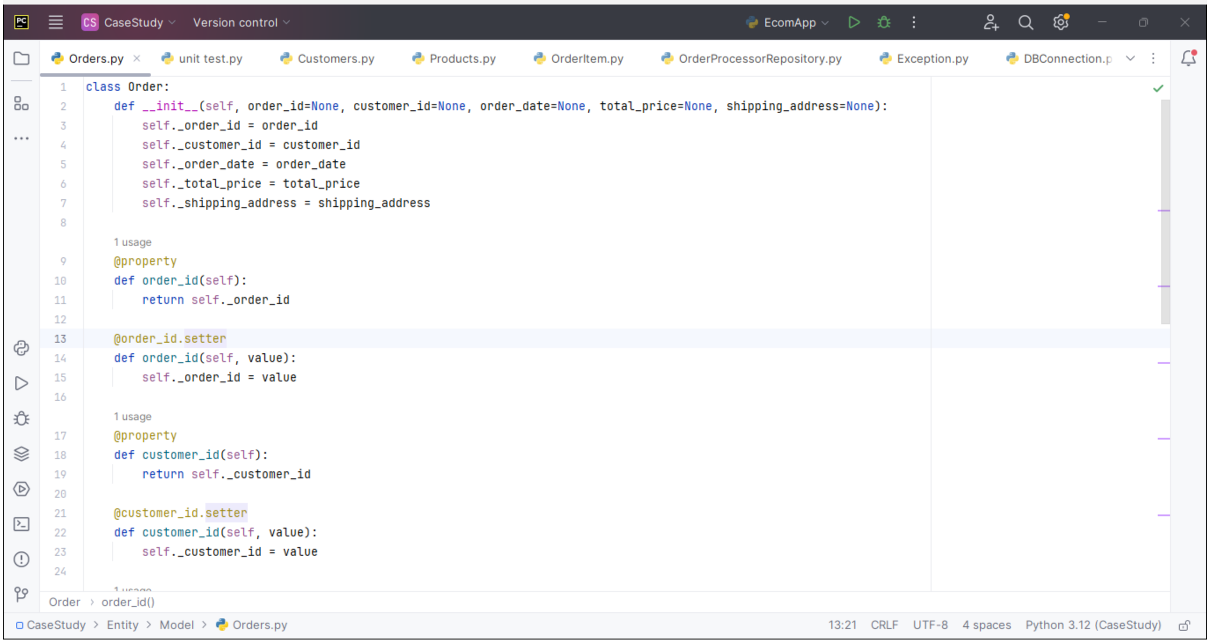


```python
class Product:
    def __init__(self, product_id=None, name=None, price=None, description=None, stock_quantity=None):
        self._product_id = product_id
        self._name = name
        self._price = price
        self._description = description
        self._stock_quantity = stock_quantity

    @property
    def product_id(self):
        return self._product_id

    @product_id.setter
    def product_id(self, value):
        self._product_id = value

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

```

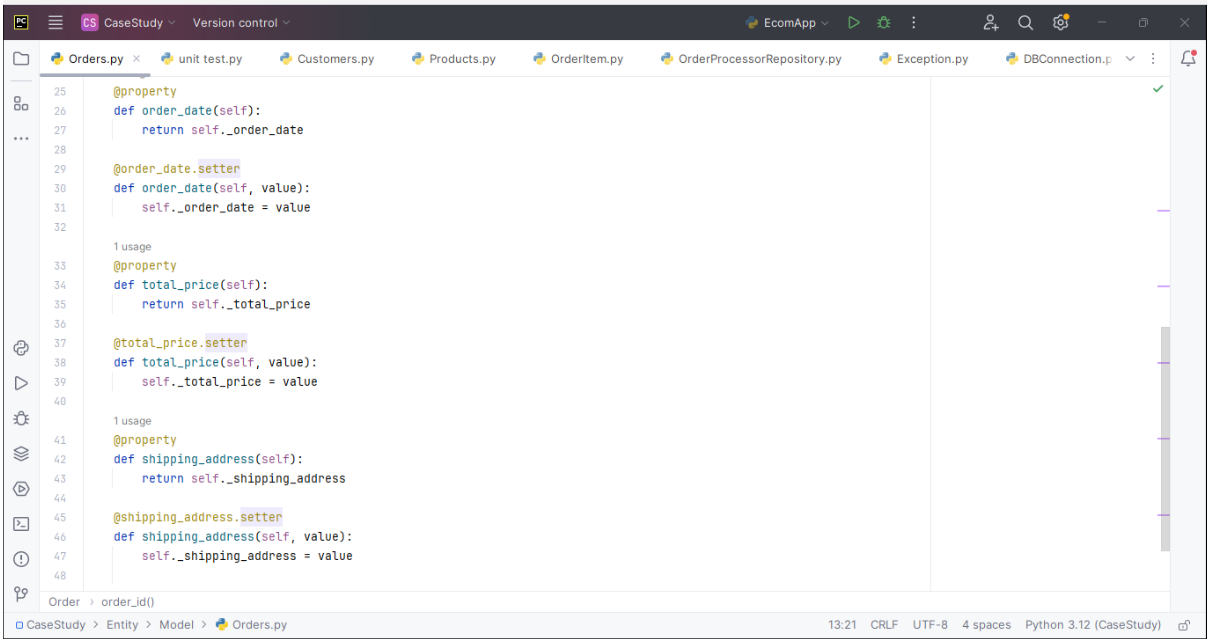


```python
    @property
    def price(self):
        return self._price

    @price.setter
    def price(self, value):
        self._price = value

    @property
    def description(self):
        return self._description

    @description.setter
    def description(self, value):
        self._description = value

    @property
    def stock_quantity(self):
        return self._stock_quantity

    @stock_quantity.setter
    def stock_quantity(self, value):
        self._stock_quantity = value

```

Orders:-

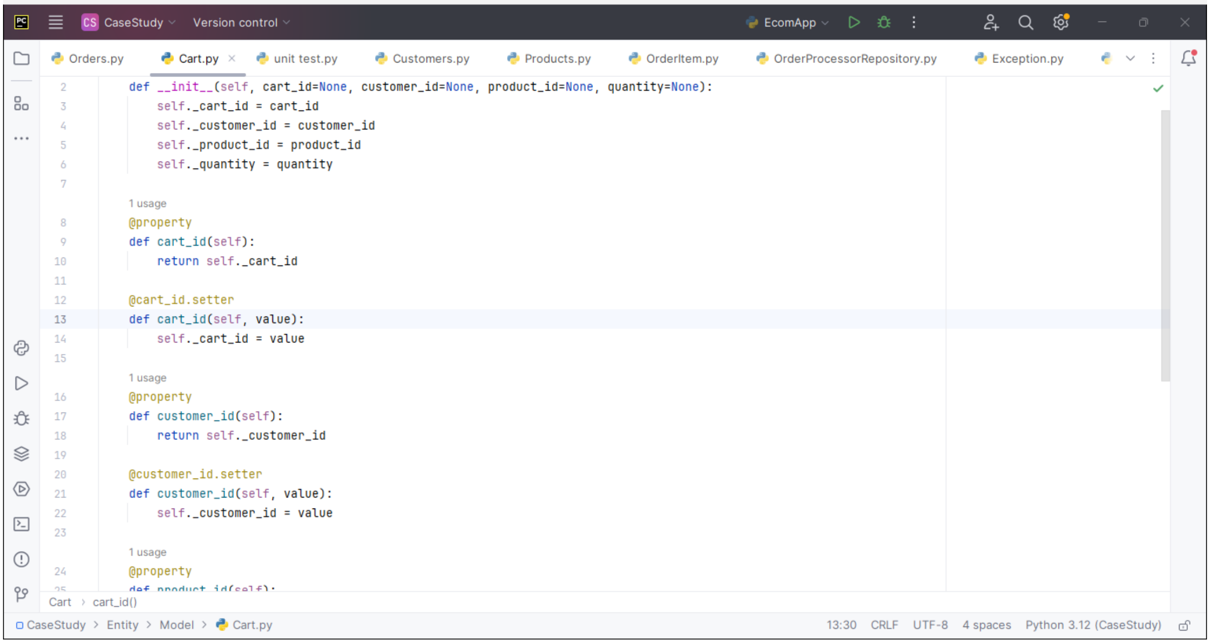


```python
class Order:
    def __init__(self, order_id=None, customer_id=None, order_date=None, total_price=None, shipping_address=None):
        self._order_id = order_id
        self._customer_id = customer_id
        self._order_date = order_date
        self._total_price = total_price
        self._shipping_address = shipping_address

    @property
    def order_id(self):
        return self._order_id

    @order_id.setter
    def order_id(self, value):
        self._order_id = value

    @property
    def customer_id(self):
        return self._customer_id

    @customer_id.setter
    def customer_id(self, value):
        self._customer_id = value

```

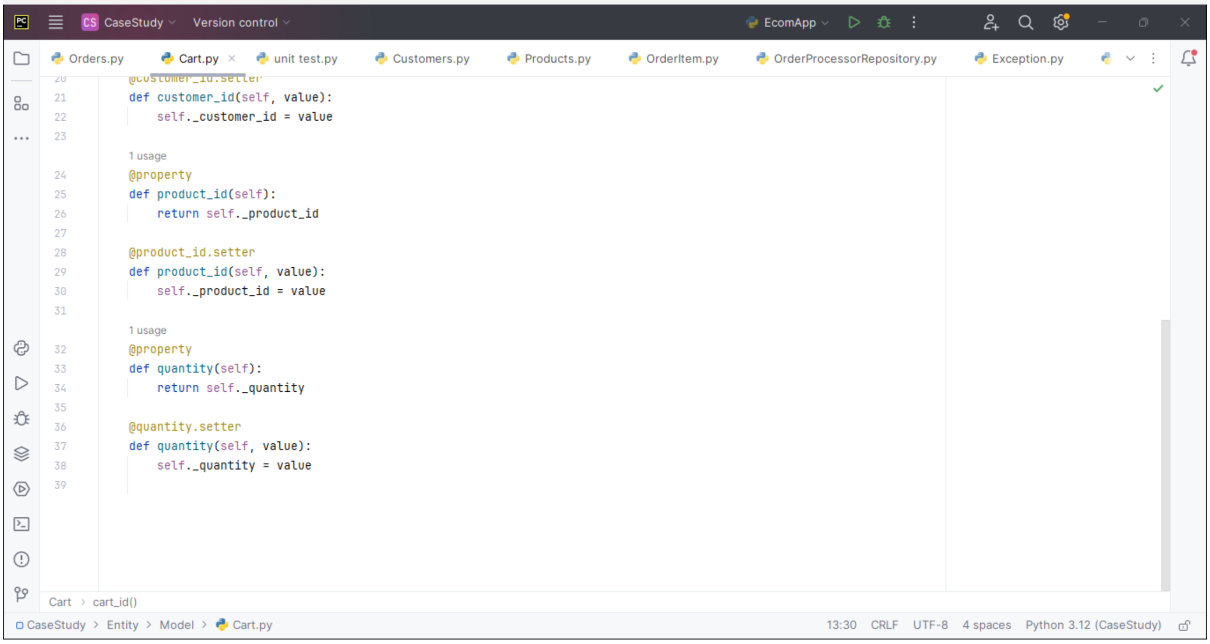


```
    @property
    def order_date(self):
        return self._order_date

    @order_date.setter
    def order_date(self, value):
        self._order_date = value

    @property
    def total_price(self):
        return self._total_price

    @total_price.setter
    def total_price(self, value):
        self._total_price = value

    @property
    def shipping_address(self):
        return self._shipping_address

    @shipping_address.setter
    def shipping_address(self, value):
        self._shipping_address = value

```

Cart:-

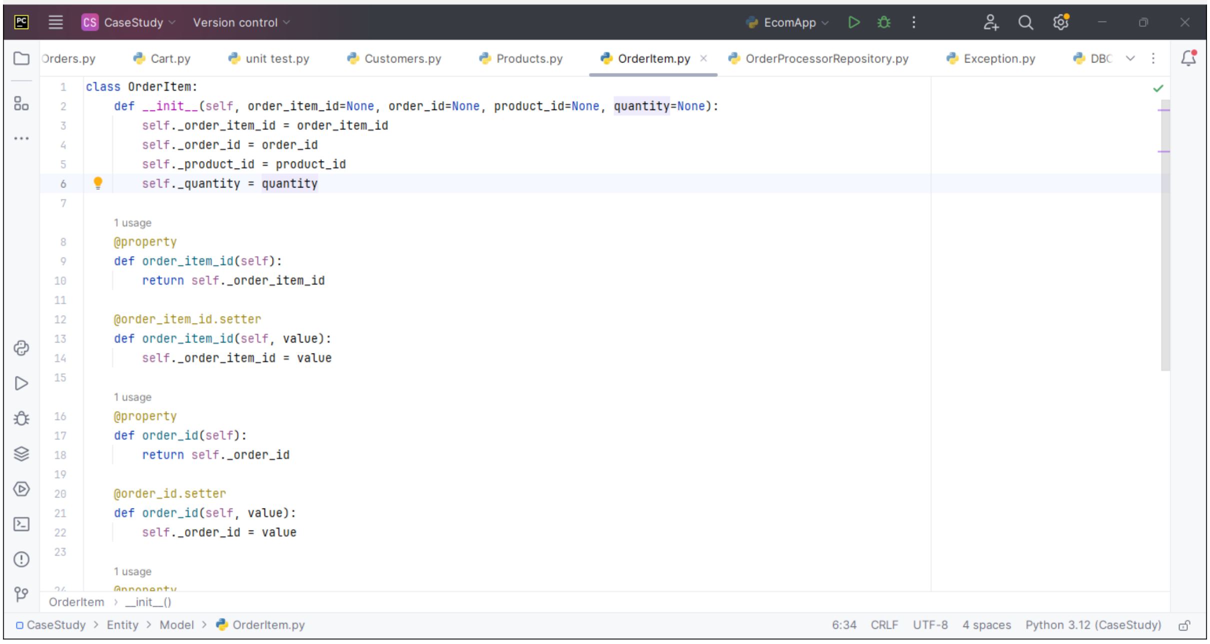


```
    def __init__(self, cart_id=None, customer_id=None, product_id=None, quantity=None):
        self._cart_id = cart_id
        self._customer_id = customer_id
        self._product_id = product_id
        self._quantity = quantity

    @property
    def cart_id(self):
        return self._cart_id

    @cart_id.setter
    def cart_id(self, value):
        self._cart_id = value

    @property
    def customer_id(self):
        return self._customer_id

    @customer_id.setter
    def customer_id(self, value):
        self._customer_id = value

    @property
    def product_id(self):
```

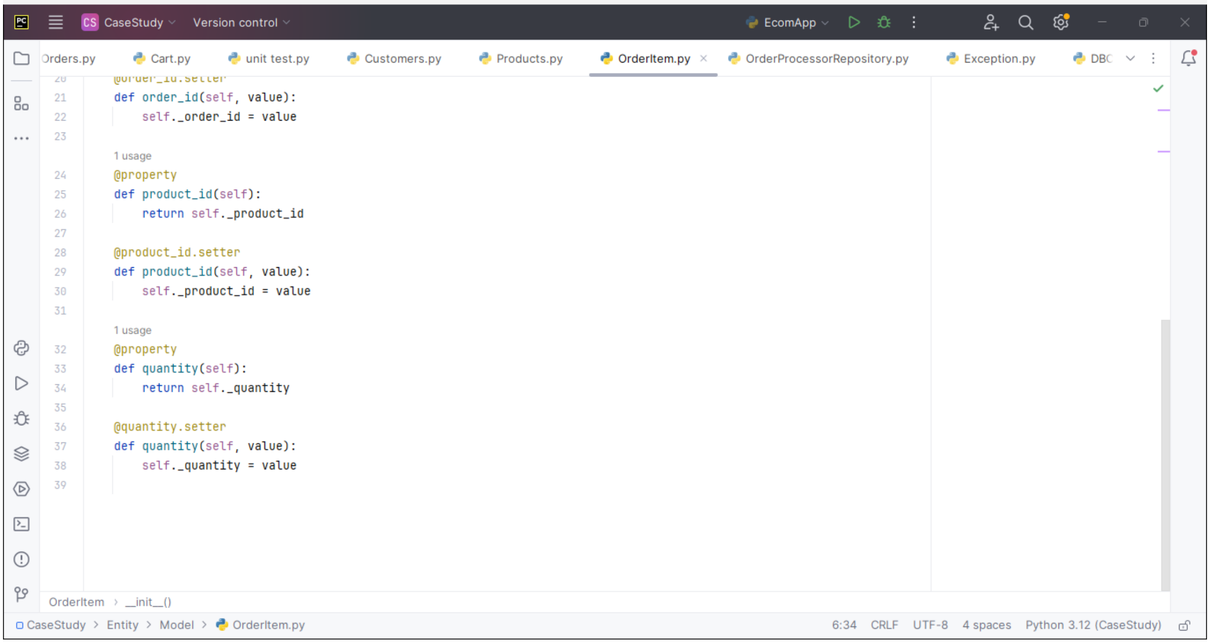


```python
    @customer_id.setter
    def customer_id(self, value):
        self._customer_id = value

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

    @product_id.setter
    def product_id(self, value):
        self._product_id = value

    1 usage
    @property
    def quantity(self):
        return self._quantity

    @quantity.setter
    def quantity(self, value):
        self._quantity = value
```

Orderitem:-



```python
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

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

    @order_item_id.setter
    def order_item_id(self, value):
        self._order_item_id = value

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

    @order_id.setter
    def order_id(self, value):
        self._order_id = value

    1 usage
    @property
```



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

    @property
    def product_id(self):
        return self._product_id

    @product_id.setter
    def product_id(self, value):
        self._product_id = value

    @property
    def quantity(self):
        return self._quantity

    @quantity.setter
    def quantity(self, value):
        self._quantity = value
```

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.

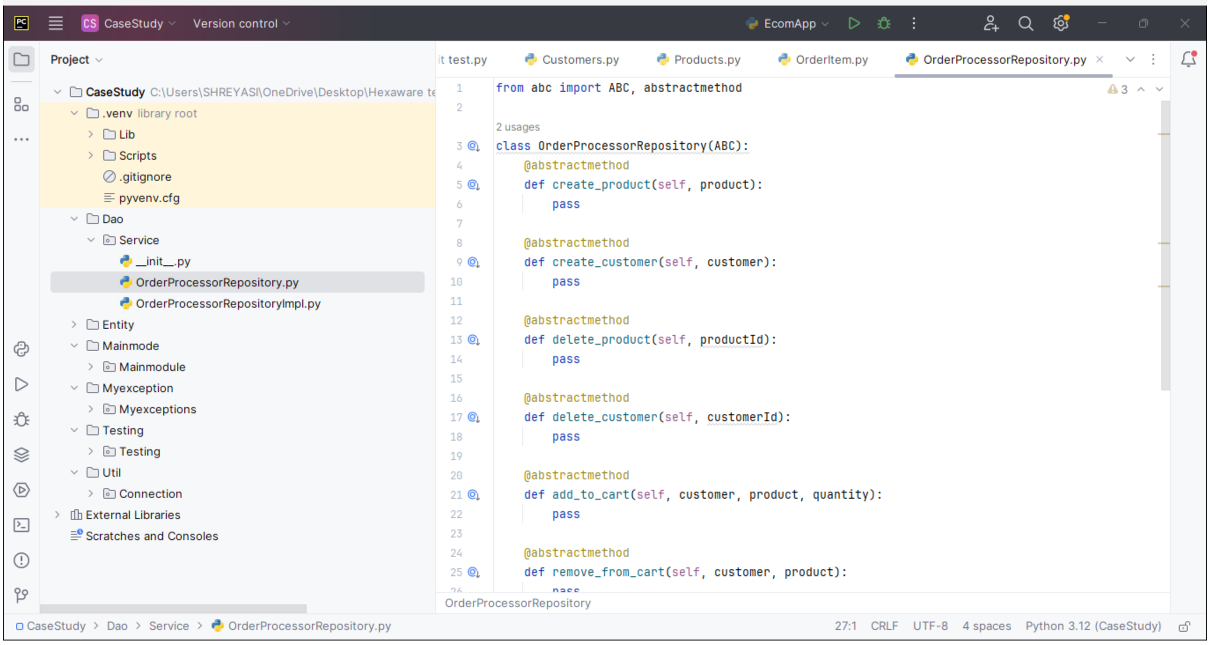
1. parameter: Customer customer, list of product and quantity

2. return type: boolean

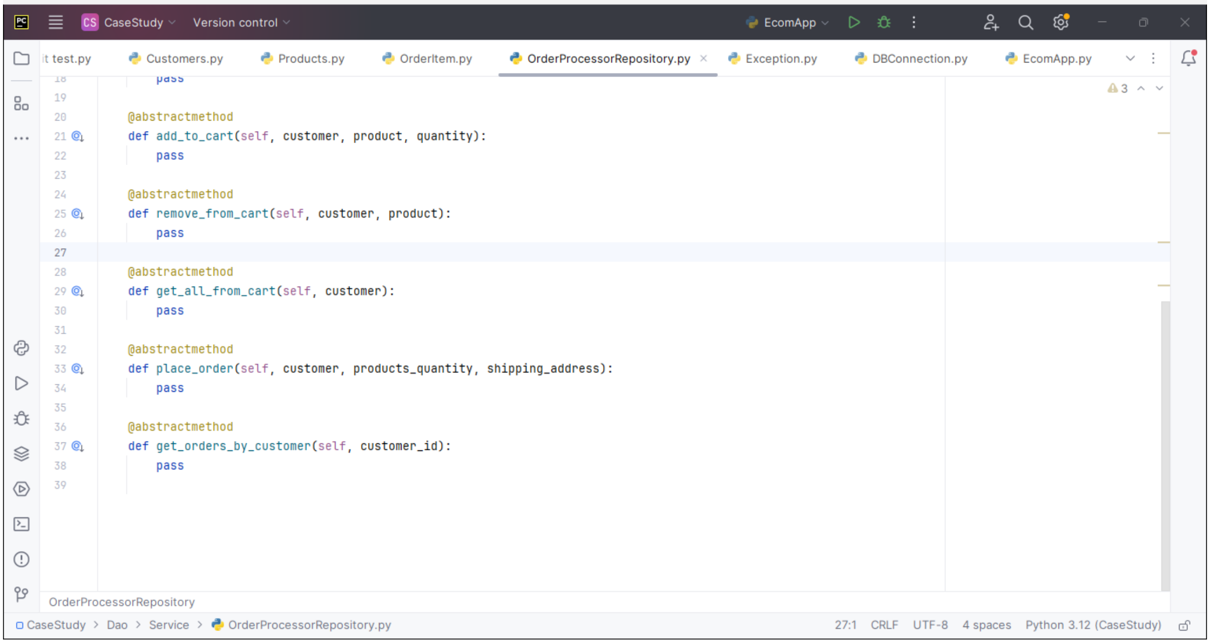
9. getOrdersByCustomer()

1. parameter: customerid

2. return type: list of product and quantity

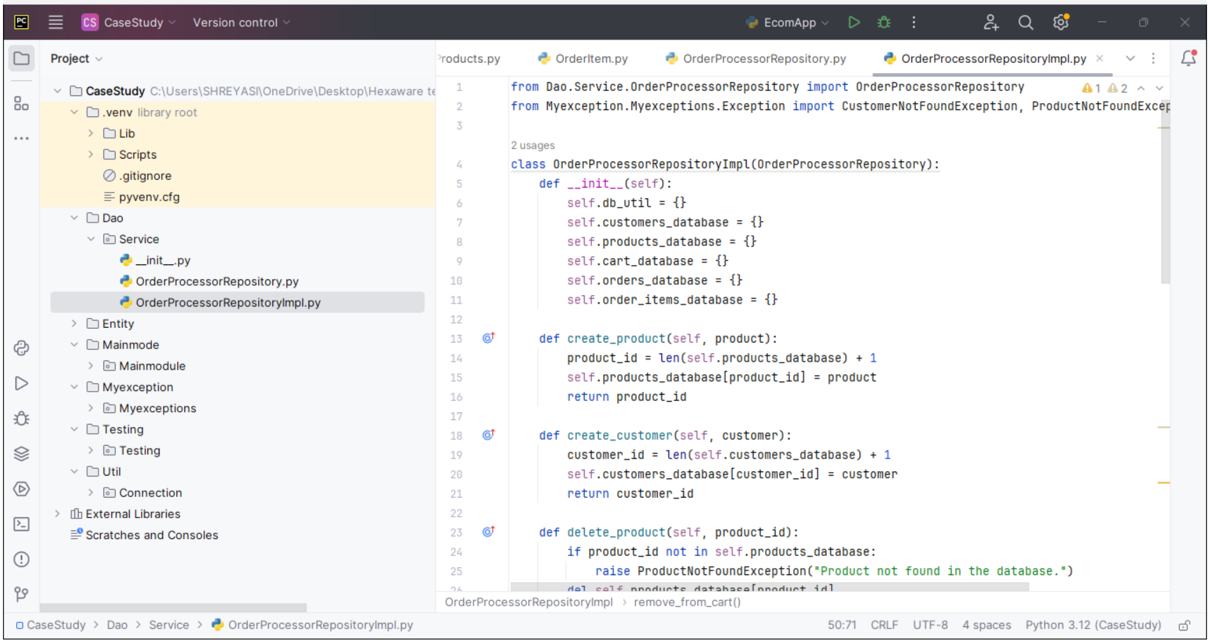


```python
from abc import ABC, abstractmethod

class OrderProcessorRepository(ABC):
    @abstractmethod
    def create_product(self, product):
        pass

    @abstractmethod
    def create_customer(self, customer):
        pass

    @abstractmethod
    def delete_product(self, productId):
        pass

    @abstractmethod
    def delete_customer(self, customerId):
        pass

    @abstractmethod
    def add_to_cart(self, customer, product, quantity):
        pass

    @abstractmethod
    def remove_from_cart(self, customer, product):
        pass
```



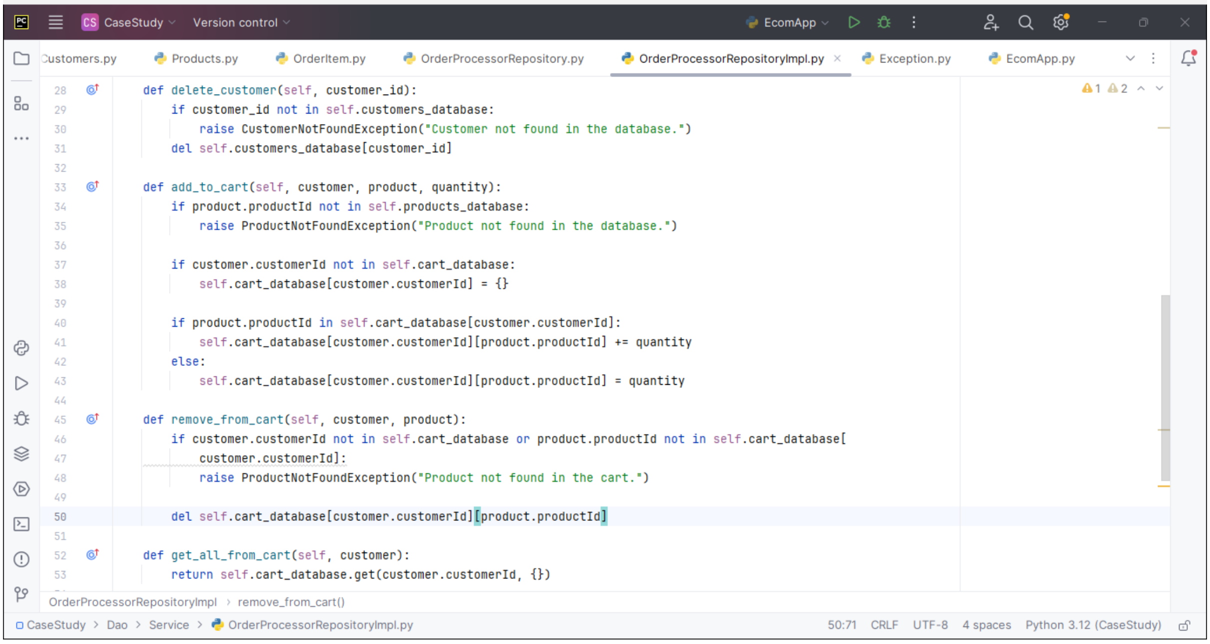
```python
        pass

    @abstractmethod
    def add_to_cart(self, customer, product, quantity):
        pass

    @abstractmethod
    def remove_from_cart(self, customer, product):
        pass

    @abstractmethod
    def get_all_from_cart(self, customer):
        pass

    @abstractmethod
    def place_order(self, customer, products_quantity, shipping_address):
        pass

    @abstractmethod
    def get_orders_by_customer(self, customer_id):
        pass

```

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

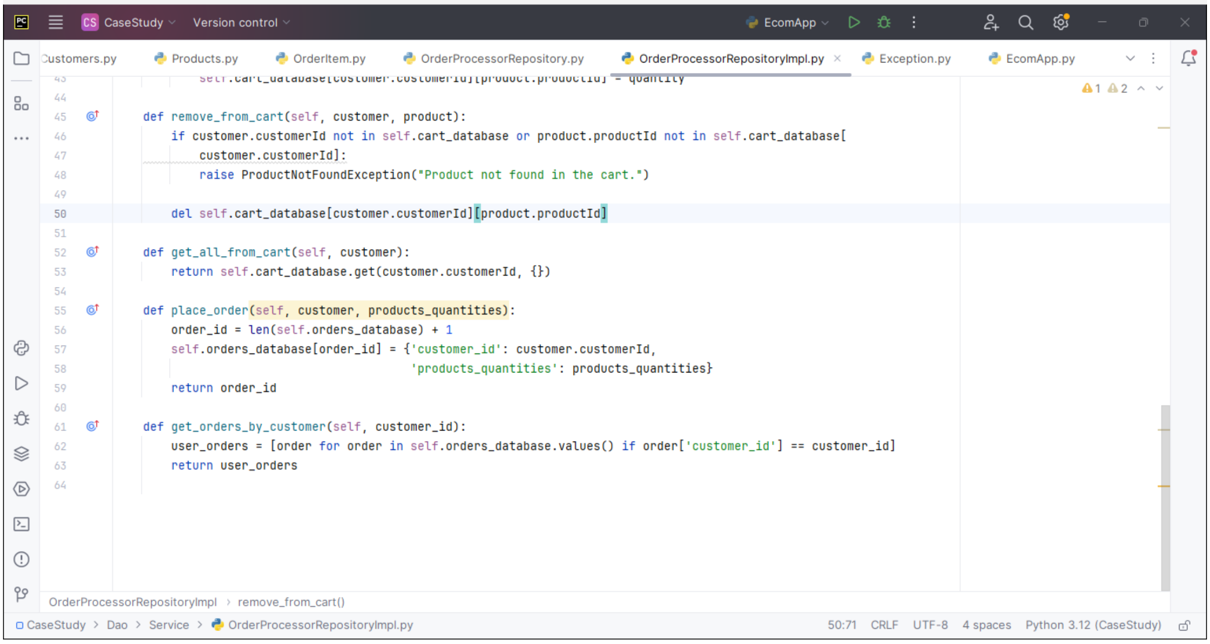


CaseStudy
Version control
EcomApp
Project
CaseStudy C:\Users\SHREYASI\OneDrive\Desktop\Hexaware te
.venv library root
Lib
Scripts
.gitignore
pyvenv.cfg
Dao
Service
__init__.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
Entity
Mainmode
Mainmodule
Myexception
Myexceptions
Testing
Testing
Util
Connection
External Libraries
Scratches and Consoles
roducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
2 usages
OrderProcessorRepositoryImpl › remove_from_cart()
CaseStudy › Dao › Service › OrderProcessorRepositoryImpl.py
50:71 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```python
from Dao.Service.OrderProcessorRepository import OrderProcessorRepository
from Myexception.Myexceptions.Exception import CustomerNotFoundException, ProductNotFoundExcep

class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def __init__(self):
        self.db_util = {}
        self.customers_database = {}
        self.products_database = {}
        self.cart_database = {}
        self.orders_database = {}
        self.order_items_database = {}

    def create_product(self, product):
        product_id = len(self.products_database) + 1
        self.products_database[product_id] = product
        return product_id

    def create_customer(self, customer):
        customer_id = len(self.customers_database) + 1
        self.customers_database[customer_id] = customer
        return customer_id

    def delete_product(self, product_id):
        if product_id not in self.products_database:
            raise ProductNotFoundException("Product not found in the database.")
        del self.products_database[product_id]
```


CaseStudy
Version control
EcomApp
Customers.py
Products.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
Exception.py
EcomApp.py
OrderProcessorRepositoryImpl › remove_from_cart()
CaseStudy › Dao › Service › OrderProcessorRepositoryImpl.py
50:71 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```
    def delete_customer(self, customer_id):
        if customer_id not in self.customers_database:
            raise CustomerNotFoundException("Customer not found in the database.")
        del self.customers_database[customer_id]

    def add_to_cart(self, customer, product, quantity):
        if product.productId not in self.products_database:
            raise ProductNotFoundException("Product not found in the database.")

        if customer.customerId not in self.cart_database:
            self.cart_database[customer.customerId] = {}

        if product.productId in self.cart_database[customer.customerId]:
            self.cart_database[customer.customerId][product.productId] += quantity
        else:
            self.cart_database[customer.customerId][product.productId] = quantity

    def remove_from_cart(self, customer, product):
        if customer.customerId not in self.cart_database or product.productId not in self.cart_database[
            customer.customerId]:
            raise ProductNotFoundException("Product not found in the cart.")

        del self.cart_database[customer.customerId][product.productId]

    def get_all_from_cart(self, customer):
        return self.cart_database.get(customer.customerId, {})
```


CaseStudy
Version control
EcomApp
Customers.py
Products.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
Exception.py
EcomApp.py
OrderProcessorRepositoryImpl › remove_from_cart()
CaseStudy › Dao › Service › OrderProcessorRepositoryImpl.py
50:71 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```
            self.cart_database[customer.customerId][product.productId] = quantity

    def remove_from_cart(self, customer, product):
        if customer.customerId not in self.cart_database or product.productId not in self.cart_database[
            customer.customerId]:
            raise ProductNotFoundException("Product not found in the cart.")

        del self.cart_database[customer.customerId][product.productId]

    def get_all_from_cart(self, customer):
        return self.cart_database.get(customer.customerId, {})

    def place_order(self, customer, products_quantities):
        order_id = len(self.orders_database) + 1
        self.orders_database[order_id] = {'customer_id': customer.customerId,
                                          'products_quantities': products_quantities}
        return order_id

    def get_orders_by_customer(self, customer_id):
        user_orders = [order for order in self.orders_database.values() if order['customer_id'] == customer_id]
        return user_orders
```

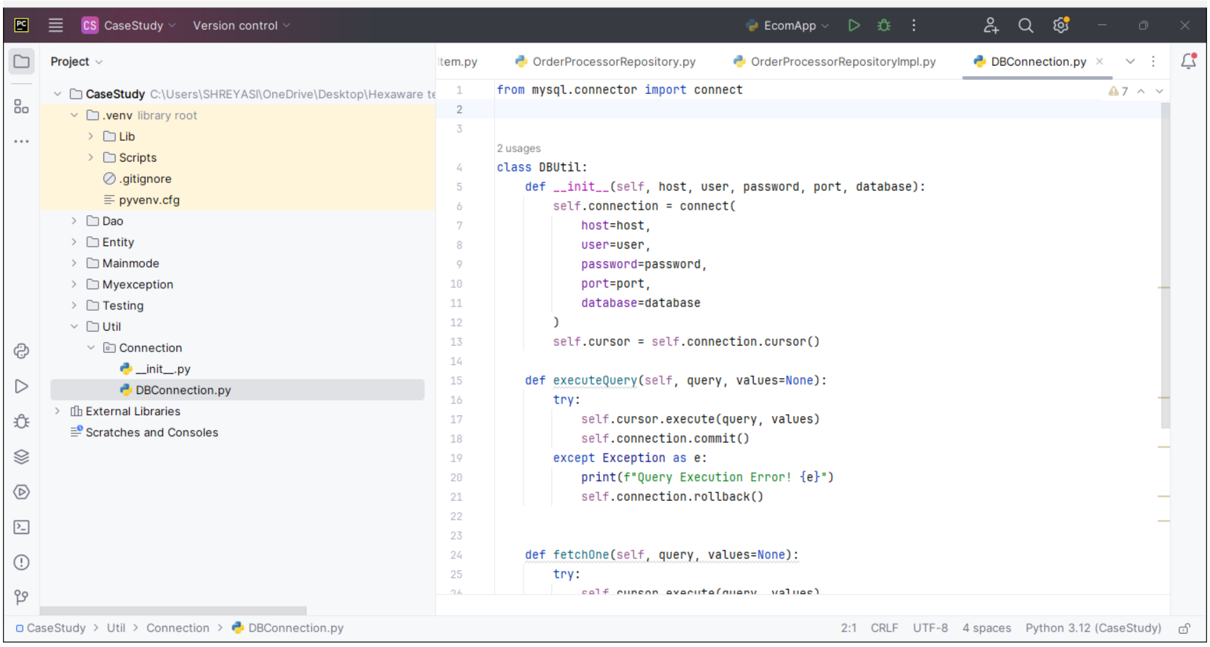
Connect your application to the SQL database:

8. Write code to establish a connection to your SQL database.

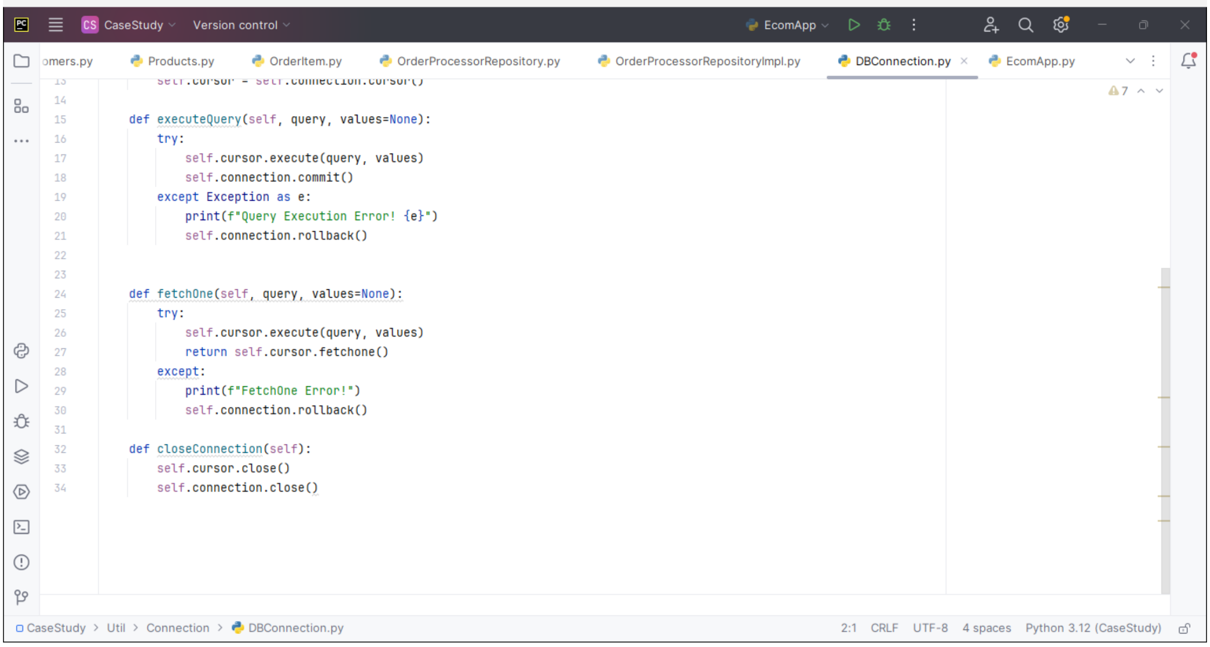
Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.

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.



```
from mysql.connector import connect


class DBUtil:
    def __init__(self, host, user, password, port, database):
        self.connection = connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    def executeQuery(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            self.connection.commit()
        except Exception as e:
            print(f"Query Execution Error! {e}")
            self.connection.rollback()


    def fetchOne(self, query, values=None):
        try:
            self.cursor.execute(query, values)
```


CaseStudy
Version control
EcomApp
omers.py
Products.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
EcomApp.py
CaseStudy › Util › Connection › DBConnection.py
2:1 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

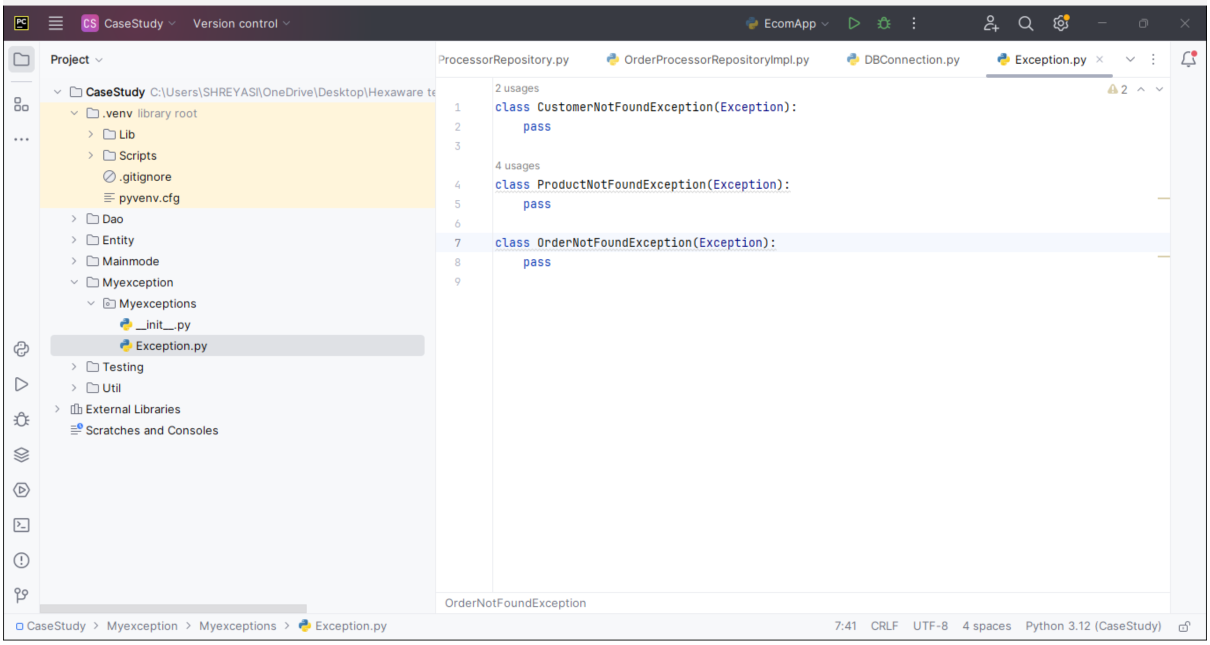
```
        self.cursor = self.connection.cursor()

    def executeQuery(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            self.connection.commit()
        except Exception as e:
            print(f"Query Execution Error! {e}")
            self.connection.rollback()


    def fetchOne(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchone()
        except:
            print(f"FetchOne Error!")
            self.connection.rollback()

    def closeConnection(self):
        self.cursor.close()
        self.connection.close()
```

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



```python
class CustomerNotFoundException(Exception):
    pass


class ProductNotFoundException(Exception):
    pass

class OrderNotFoundException(Exception):
    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.

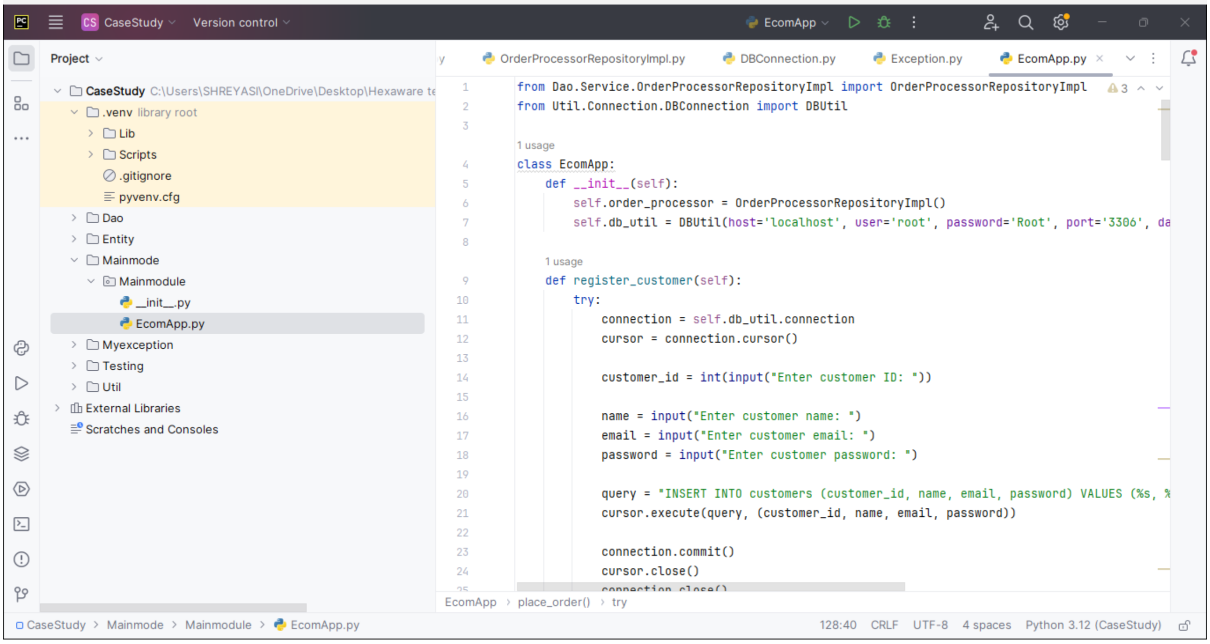
3. Delete Product.

4. Add to cart.

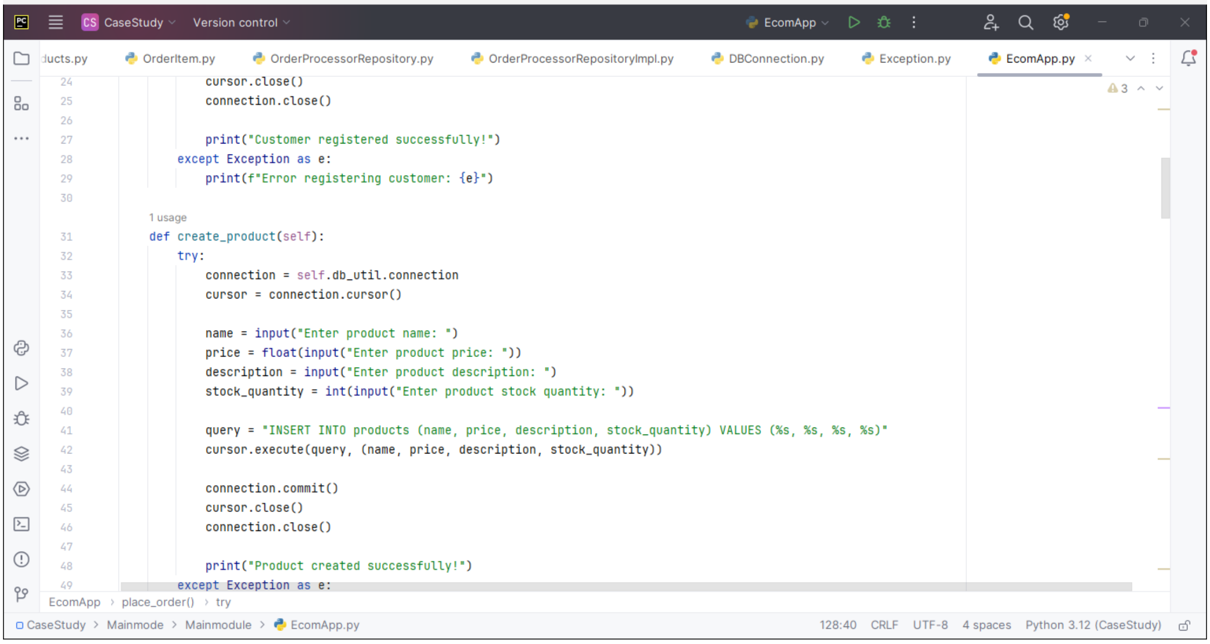
5. View cart.

6. Place order.

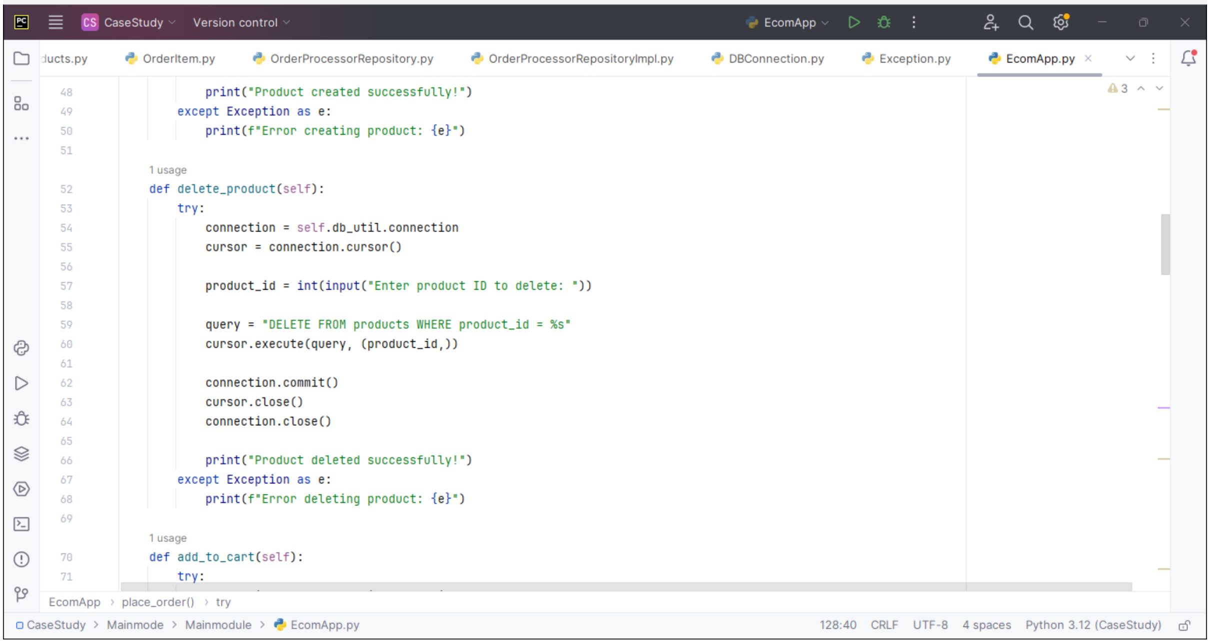
7. View Customer Order



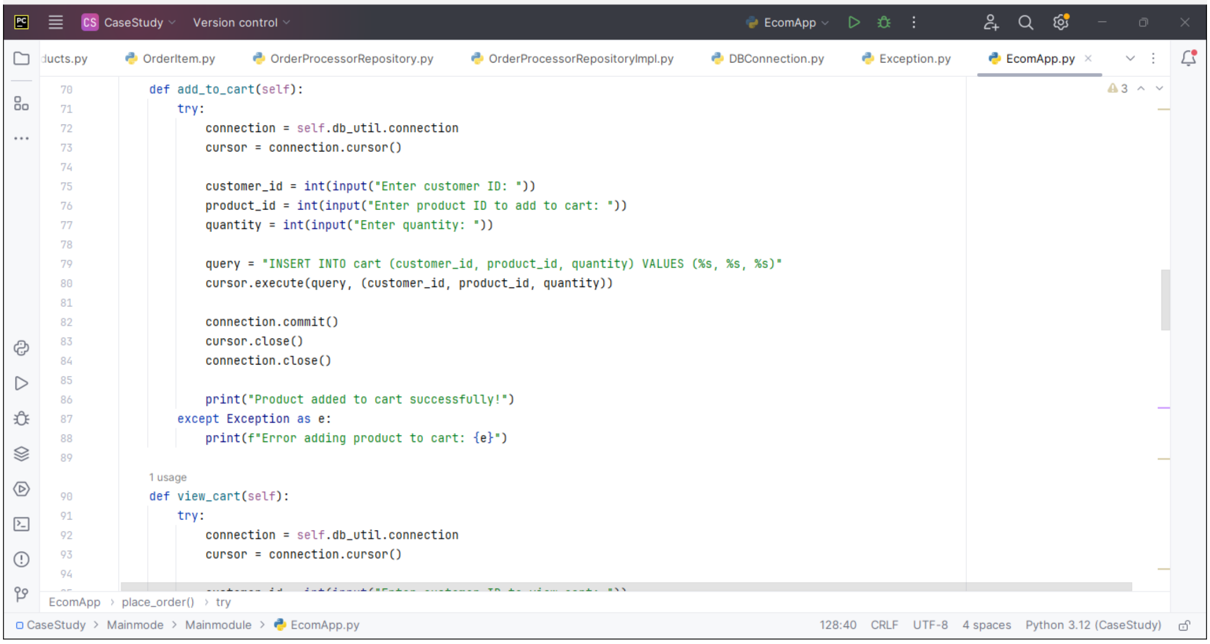
```python
from Dao.Service.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
from Util.Connection.DBConnection import DBUtil

class EcomApp:
    def __init__(self):
        self.order_processor = OrderProcessorRepositoryImpl()
        self.db_util = DBUtil(host='localhost', user='root', password='Root', port='3306', da

    def register_customer(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            customer_id = int(input("Enter customer ID: "))

            name = input("Enter customer name: ")
            email = input("Enter customer email: ")
            password = input("Enter customer password: ")

            query = "INSERT INTO customers (customer_id, name, email, password) VALUES (%s, %
            cursor.execute(query, (customer_id, name, email, password))

            connection.commit()
            cursor.close()
            connection.close()
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

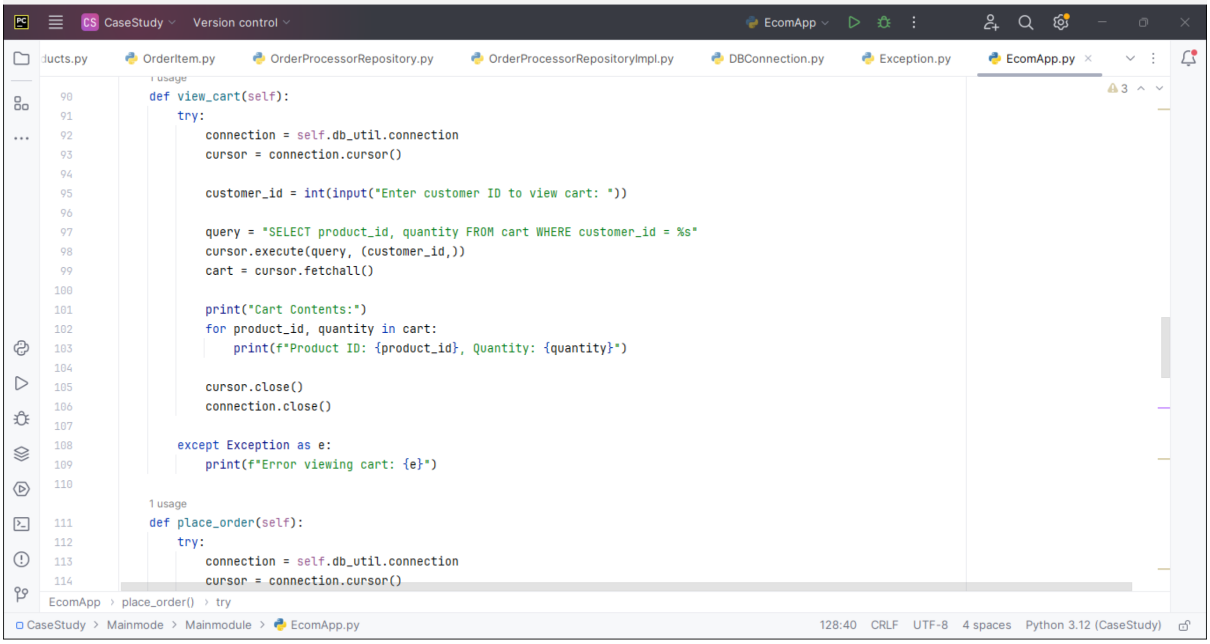
```python
            cursor.close()
            connection.close()

            print("Customer registered successfully!")
        except Exception as e:
            print(f"Error registering customer: {e}")

    def create_product(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            name = input("Enter product name: ")
            price = float(input("Enter product price: "))
            description = input("Enter product description: ")
            stock_quantity = int(input("Enter product stock quantity: "))

            query = "INSERT INTO products (name, price, description, stock_quantity) VALUES (%s, %s, %s, %s)"
            cursor.execute(query, (name, price, description, stock_quantity))

            connection.commit()
            cursor.close()
            connection.close()

            print("Product created successfully!")
        except Exception as e:
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

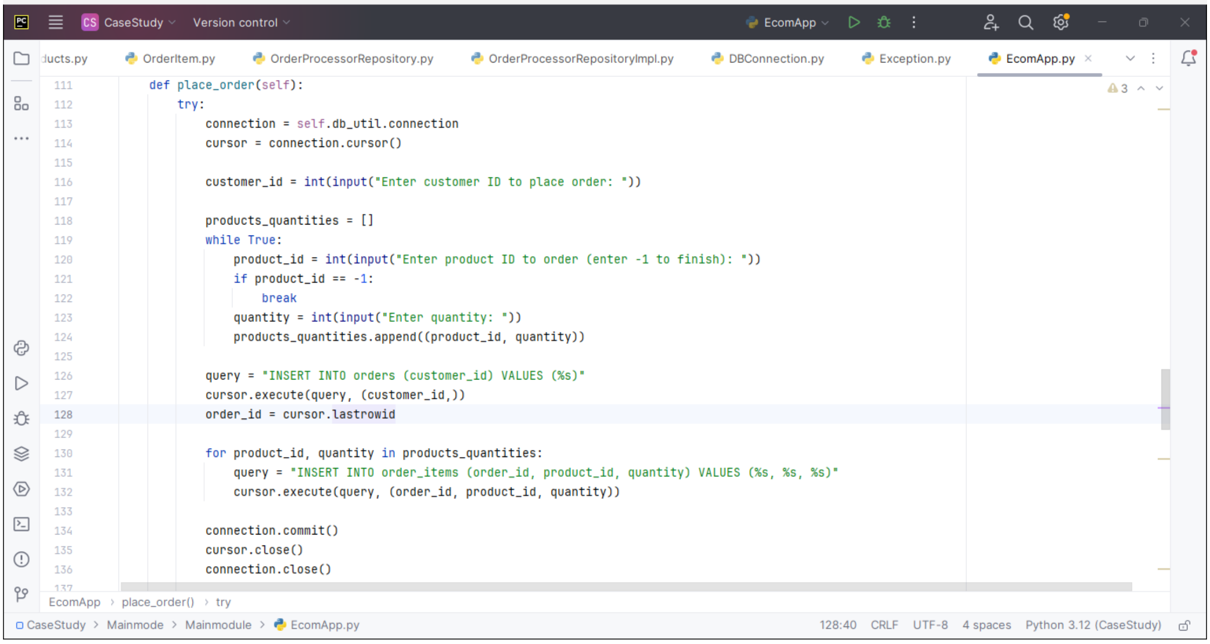
```
                print("Product created successfully!")
            except Exception as e:
                print(f"Error creating product: {e}")

        def delete_product(self):
            try:
                connection = self.db_util.connection
                cursor = connection.cursor()

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

                query = "DELETE FROM products WHERE product_id = %s"
                cursor.execute(query, (product_id,))

                connection.commit()
                cursor.close()
                connection.close()

                print("Product deleted successfully!")
            except Exception as e:
                print(f"Error deleting product: {e}")

        def add_to_cart(self):
            try:
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

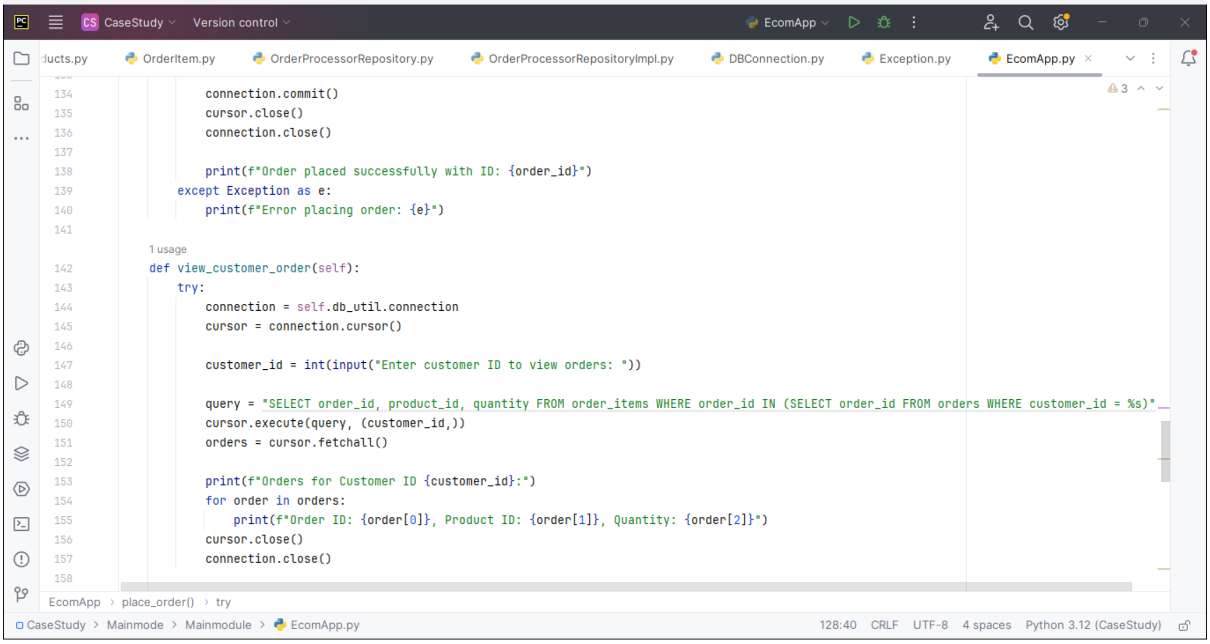
```
        def add_to_cart(self):
            try:
                connection = self.db_util.connection
                cursor = connection.cursor()

                customer_id = int(input("Enter customer ID: "))
                product_id = int(input("Enter product ID to add to cart: "))
                quantity = int(input("Enter quantity: "))

                query = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
                cursor.execute(query, (customer_id, product_id, quantity))

                connection.commit()
                cursor.close()
                connection.close()

                print("Product added to cart successfully!")
            except Exception as e:
                print(f"Error adding product to cart: {e}")

        def view_cart(self):
            try:
                connection = self.db_util.connection
                cursor = connection.cursor()

                customer_id = int(input("Enter customer ID to view cart: "))
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

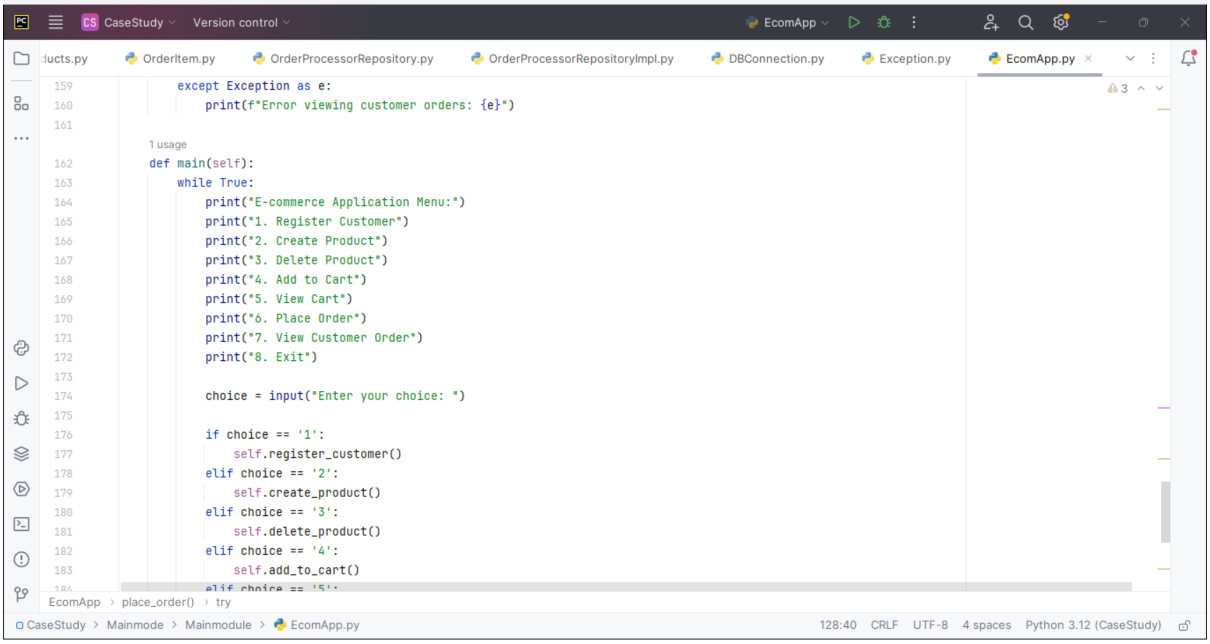
```
    def view_cart(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            customer_id = int(input("Enter customer ID to view cart: "))

            query = "SELECT product_id, quantity FROM cart WHERE customer_id = %s"
            cursor.execute(query, (customer_id,))
            cart = cursor.fetchall()

            print("Cart Contents:")
            for product_id, quantity in cart:
                print(f"Product ID: {product_id}, Quantity: {quantity}")

            cursor.close()
            connection.close()

        except Exception as e:
            print(f"Error viewing cart: {e}")

    def place_order(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

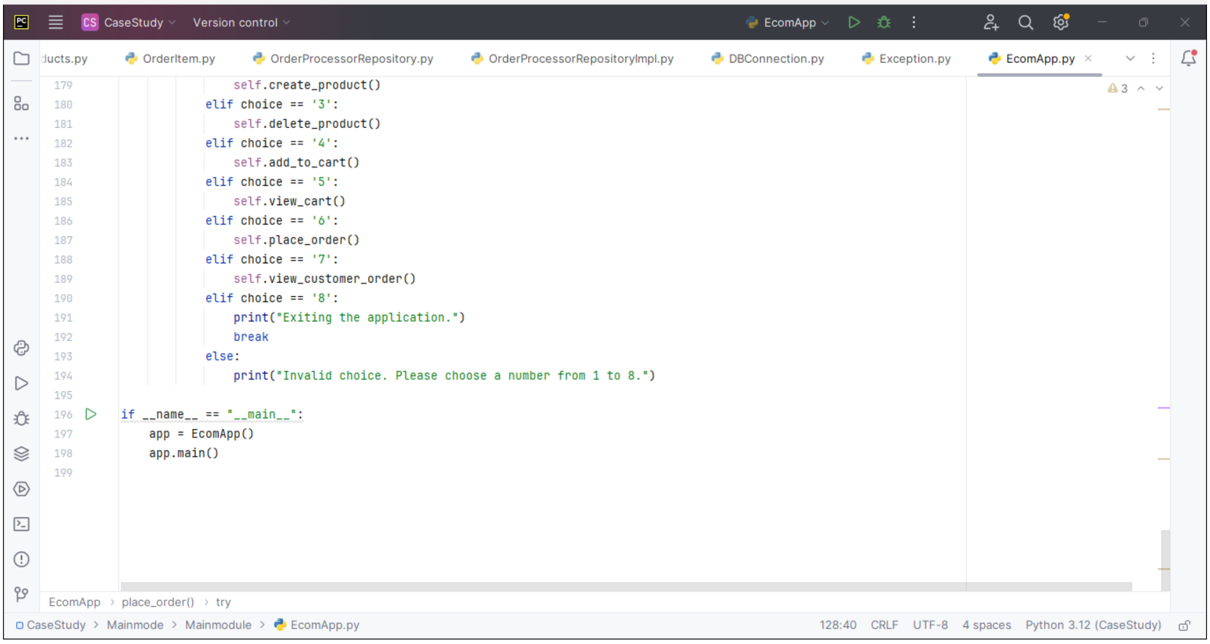
```
    def place_order(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            customer_id = int(input("Enter customer ID to place order: "))

            products_quantities = []
            while True:
                product_id = int(input("Enter product ID to order (enter -1 to finish): "))
                if product_id == -1:
                    break
                quantity = int(input("Enter quantity: "))
                products_quantities.append((product_id, quantity))

            query = "INSERT INTO orders (customer_id) VALUES (%s)"
            cursor.execute(query, (customer_id,))
            order_id = cursor.lastrowid

            for product_id, quantity in products_quantities:
                query = "INSERT INTO order_items (order_id, product_id, quantity) VALUES (%s, %s, %s)"
                cursor.execute(query, (order_id, product_id, quantity))

            connection.commit()
            cursor.close()
            connection.close()
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
⚠3
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```python
            connection.commit()
            cursor.close()
            connection.close()

            print(f"Order placed successfully with ID: {order_id}")
        except Exception as e:
            print(f"Error placing order: {e}")

    def view_customer_order(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            customer_id = int(input("Enter customer ID to view orders: "))

            query = "SELECT order_id, product_id, quantity FROM order_items WHERE order_id IN (SELECT order_id FROM orders WHERE customer_id = %s)"
            cursor.execute(query, (customer_id,))
            orders = cursor.fetchall()

            print(f"Orders for Customer ID {customer_id}:")
            for order in orders:
                print(f"Order ID: {order[0]}, Product ID: {order[1]}, Quantity: {order[2]}")
            cursor.close()
            connection.close()

```

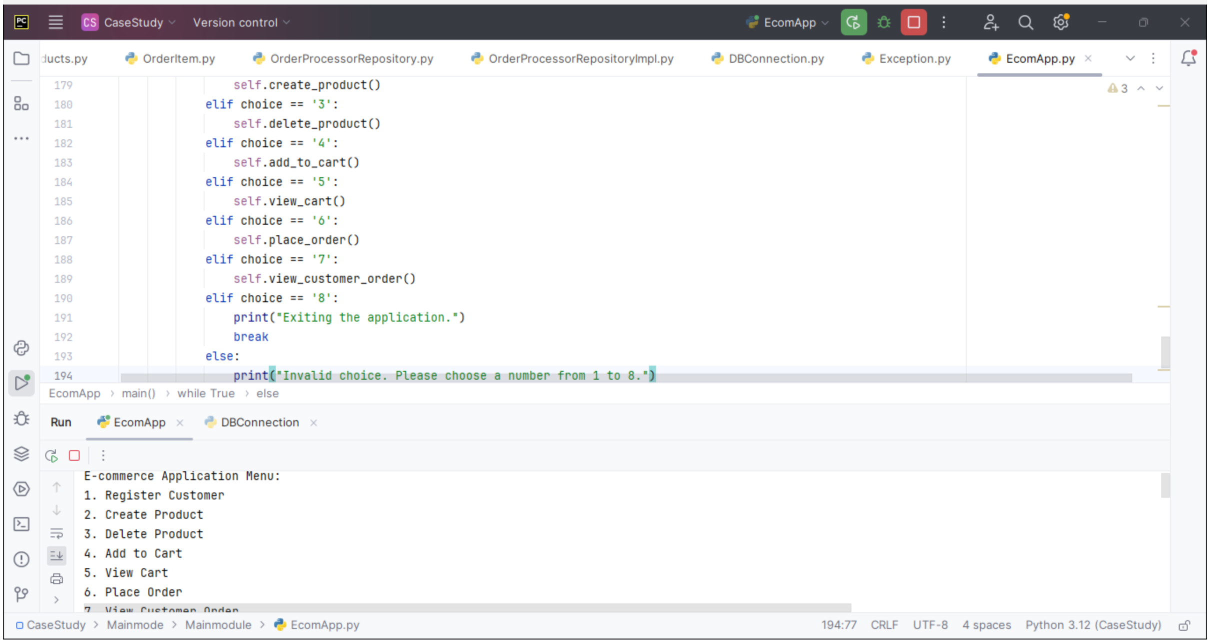

CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
⚠3
1 usage
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```python
        except Exception as e:
            print(f"Error viewing customer orders: {e}")

    def main(self):
        while True:
            print("E-commerce Application Menu:")
            print("1. Register Customer")
            print("2. Create Product")
            print("3. Delete Product")
            print("4. Add to Cart")
            print("5. View Cart")
            print("6. Place Order")
            print("7. View Customer Order")
            print("8. Exit")

            choice = input("Enter your choice: ")

            if choice == '1':
                self.register_customer()
            elif choice == '2':
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.add_to_cart()
            elif choice == '5':
```


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
EcomApp › place_order() › try
CaseStudy › Mainmode › Mainmodule › EcomApp.py
128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

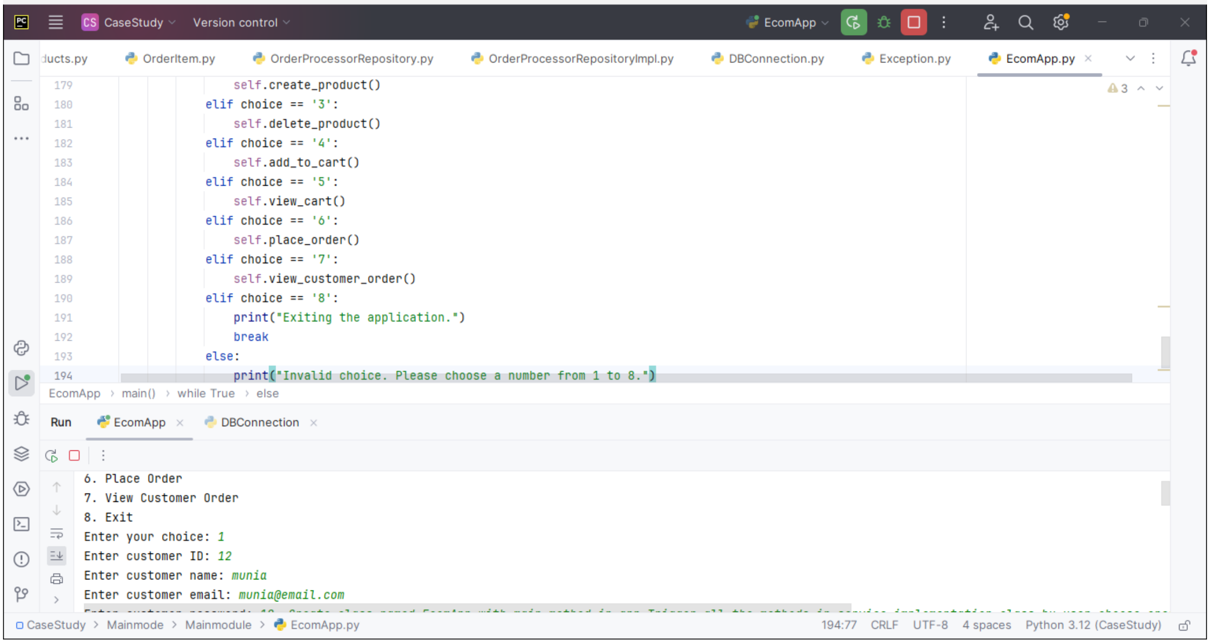
```
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.add_to_cart()
            elif choice == '5':
                self.view_cart()
            elif choice == '6':
                self.place_order()
            elif choice == '7':
                self.view_customer_order()
            elif choice == '8':
                print("Exiting the application.")
                break
            else:
                print("Invalid choice. Please choose a number from 1 to 8.")

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

Output:-


CaseStudy
Version control
EcomApp
ducts.py
OrderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
EcomApp › main() › while True › else
Run
EcomApp
DBConnection
CaseStudy › Mainmode › Mainmodule › EcomApp.py
194:77 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

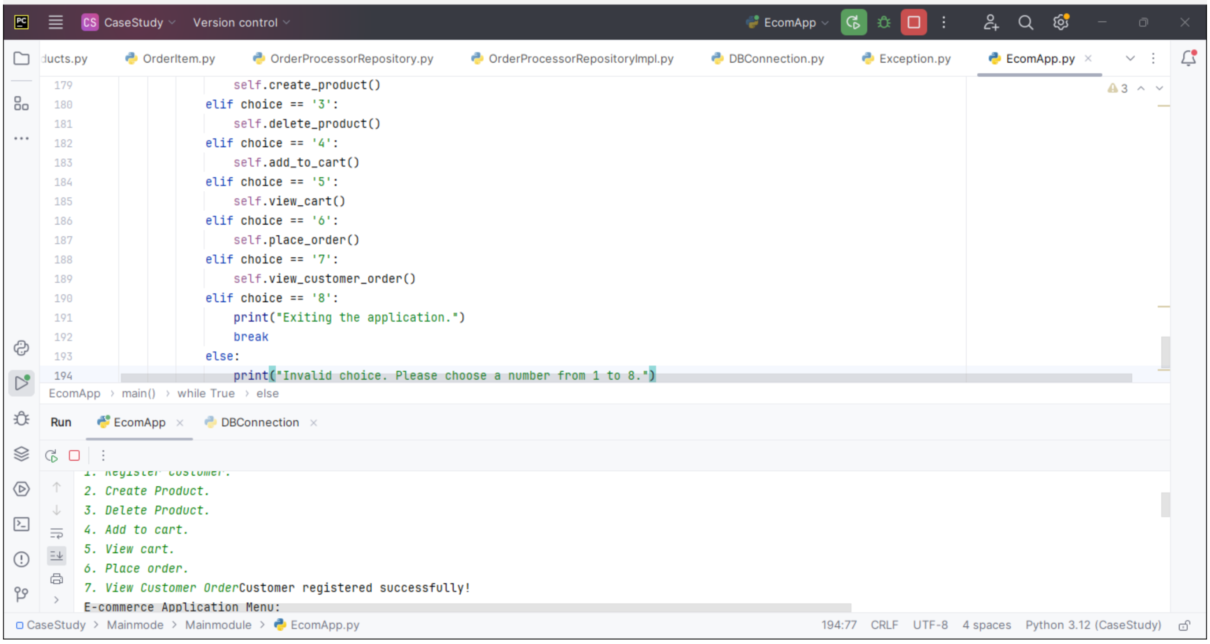
```
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.add_to_cart()
            elif choice == '5':
                self.view_cart()
            elif choice == '6':
                self.place_order()
            elif choice == '7':
                self.view_customer_order()
            elif choice == '8':
                print("Exiting the application.")
                break
            else:
                print("Invalid choice. Please choose a number from 1 to 8.")
```

```
E-commerce Application Menu:
1. Register Customer
2. Create Product
3. Delete Product
4. Add to Cart
5. View Cart
6. Place Order
7. View Customer Order
```



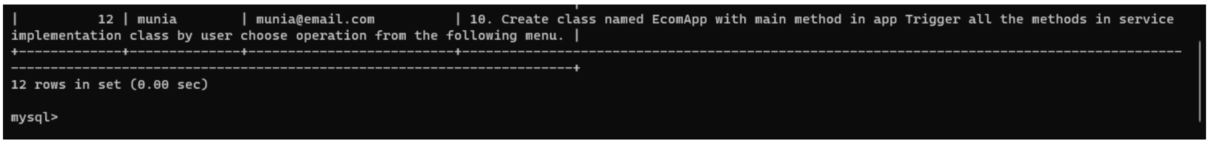
```
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.add_to_cart()
            elif choice == '5':
                self.view_cart()
            elif choice == '6':
                self.place_order()
            elif choice == '7':
                self.view_customer_order()
            elif choice == '8':
                print("Exiting the application.")
                break
            else:
                print("Invalid choice. Please choose a number from 1 to 8.")
```

```
6. Place Order
7. View Customer Order
8. Exit
Enter your choice: 1
Enter customer ID: 12
Enter customer name: munia
Enter customer email: munia@email.com
Enter customer password: 10. Create class named EcomApp with main method in app Trigger all the methods in service implementation class by user choose one
```



```
                self.create_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.add_to_cart()
            elif choice == '5':
                self.view_cart()
            elif choice == '6':
                self.place_order()
            elif choice == '7':
                self.view_customer_order()
            elif choice == '8':
                print("Exiting the application.")
                break
            else:
                print("Invalid choice. Please choose a number from 1 to 8.")
```

```
1. Register Customer.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer OrderCustomer registered successfully!
E-commerce Application Menu:
```



```
|          12 | munia        | munia@email.com          | 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. |
+-------------+--------------+--------------------------+---------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+

12 rows in set (0.00 sec)

mysql>
```

Unit Testing

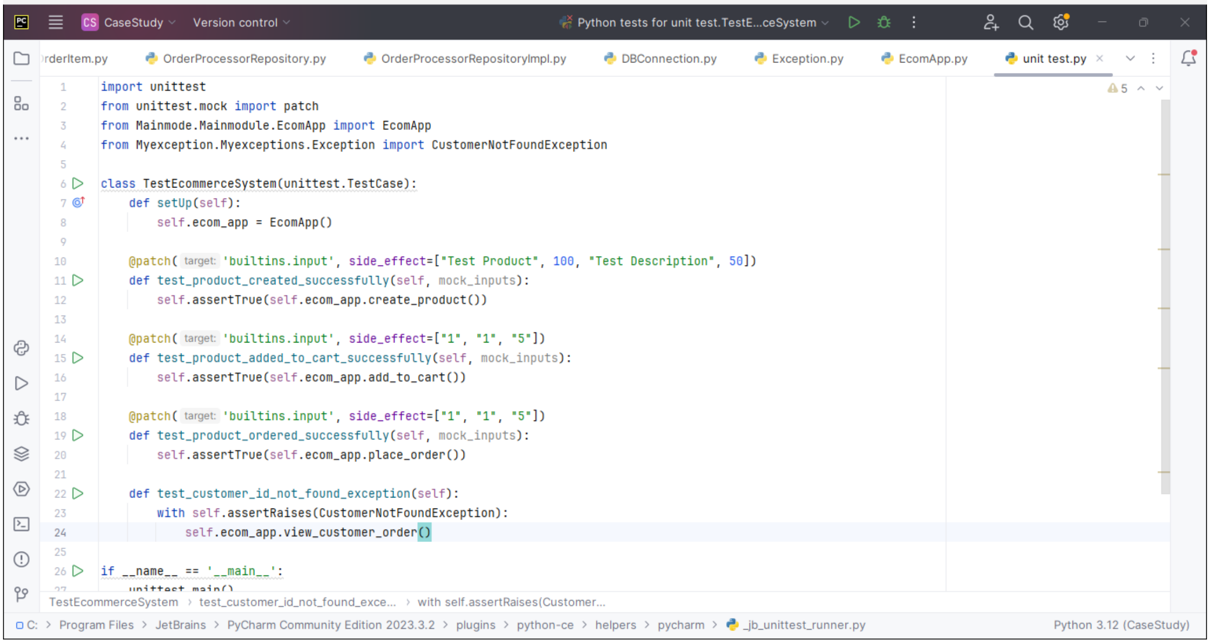
11. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

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.

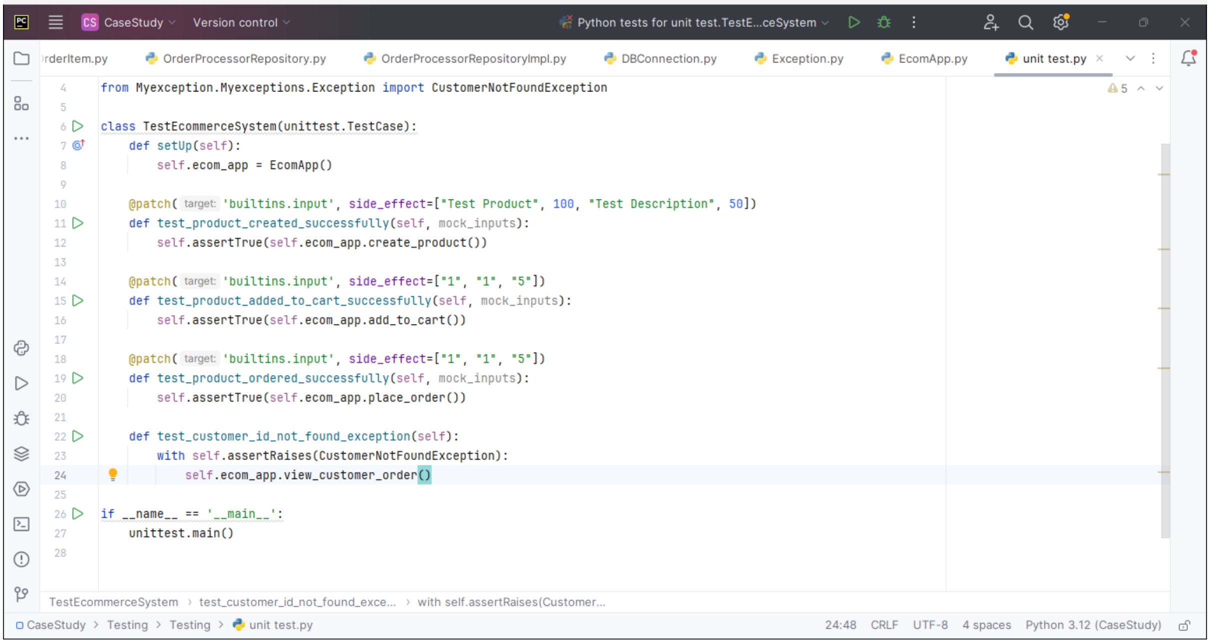
write test case to test exception is thrown correctly or not when customer id or product id not found in database.



CS CaseStudy
Version control
Python tests for unit test.TestE...ceSystem
rderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
unit test.py
5

```
import unittest
from unittest.mock import patch
from Mainmode.Mainmodule.EcomApp import EcomApp
from Myexception.Myexceptions.Exception import CustomerNotFoundException

class TestEcommerceSystem(unittest.TestCase):
    def setUp(self):
        self.ecom_app = EcomApp()

    @patch( target: 'builtins.input', side_effect=["Test Product", 100, "Test Description", 50])
    def test_product_created_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.create_product())

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_product_added_to_cart_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.add_to_cart())

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_product_ordered_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.place_order())

    def test_customer_id_not_found_exception(self):
        with self.assertRaises(CustomerNotFoundException):
            self.ecom_app.view_customer_order()

if __name__ == '__main__':
    unittest.main()
```

TestEcommerceSystem › test_customer_id_not_found_exce... › with self.assertRaises(Customer...
C: › Program Files › JetBrains › PyCharm Community Edition 2023.3.2 › plugins › python-ce › helpers › pycharm › _jb_unittest_runner.py
Python 3.12 (CaseStudy)



CS CaseStudy
Version control
Python tests for unit test.TestE...ceSystem
rderItem.py
OrderProcessorRepository.py
OrderProcessorRepositoryImpl.py
DBConnection.py
Exception.py
EcomApp.py
unit test.py
5

```
from Myexception.Myexceptions.Exception import CustomerNotFoundException

class TestEcommerceSystem(unittest.TestCase):
    def setUp(self):
        self.ecom_app = EcomApp()

    @patch( target: 'builtins.input', side_effect=["Test Product", 100, "Test Description", 50])
    def test_product_created_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.create_product())

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_product_added_to_cart_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.add_to_cart())

    @patch( target: 'builtins.input', side_effect=["1", "1", "5"])
    def test_product_ordered_successfully(self, mock_inputs):
        self.assertTrue(self.ecom_app.place_order())

    def test_customer_id_not_found_exception(self):
        with self.assertRaises(CustomerNotFoundException):
            self.ecom_app.view_customer_order()

if __name__ == '__main__':
    unittest.main()
```

TestEcommerceSystem › test_customer_id_not_found_exce... › with self.assertRaises(Customer...
CaseStudy › Testing › Testing › unit test.py
24:48
CRLF
UTF-8
4 spaces
Python 3.12 (CaseStudy)