Order management system :

Step 1: Database Design:

The schema was created based on the updated challenge instructions. It models users, products (with subtypes), orders, and order items. Foreign keys ensure referential integrity between user orders and products

1. Create the **Order Management DB schema** in MySQL Workbench.

CREATE DATABASE order\_management;

USE order\_management;

CREATE TABLE User (

userId INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password VARCHAR(100) NOT NULL,

role VARCHAR(20) NOT NULL -- "Admin" or "User"

);

CREATE TABLE Product (

productId INT AUTO\_INCREMENT PRIMARY KEY,

productName VARCHAR(100),

description TEXT,

price DECIMAL(10, 2),

quantityInStock INT,

type VARCHAR(50), -- "Electronics" or "Clothing"

brand VARCHAR(100), -- for Electronics

warrantyPeriod INT, -- for Electronics

size VARCHAR(10), -- for Clothing

color VARCHAR(50) -- for Clothing

);

CREATE TABLE Orders (

orderId INT AUTO\_INCREMENT PRIMARY KEY,

userId INT,

orderDate DATE,

FOREIGN KEY (userId) REFERENCES User(userId)

);

CREATE TABLE OrderItems (

orderItemId INT AUTO\_INCREMENT PRIMARY KEY,

orderId INT,

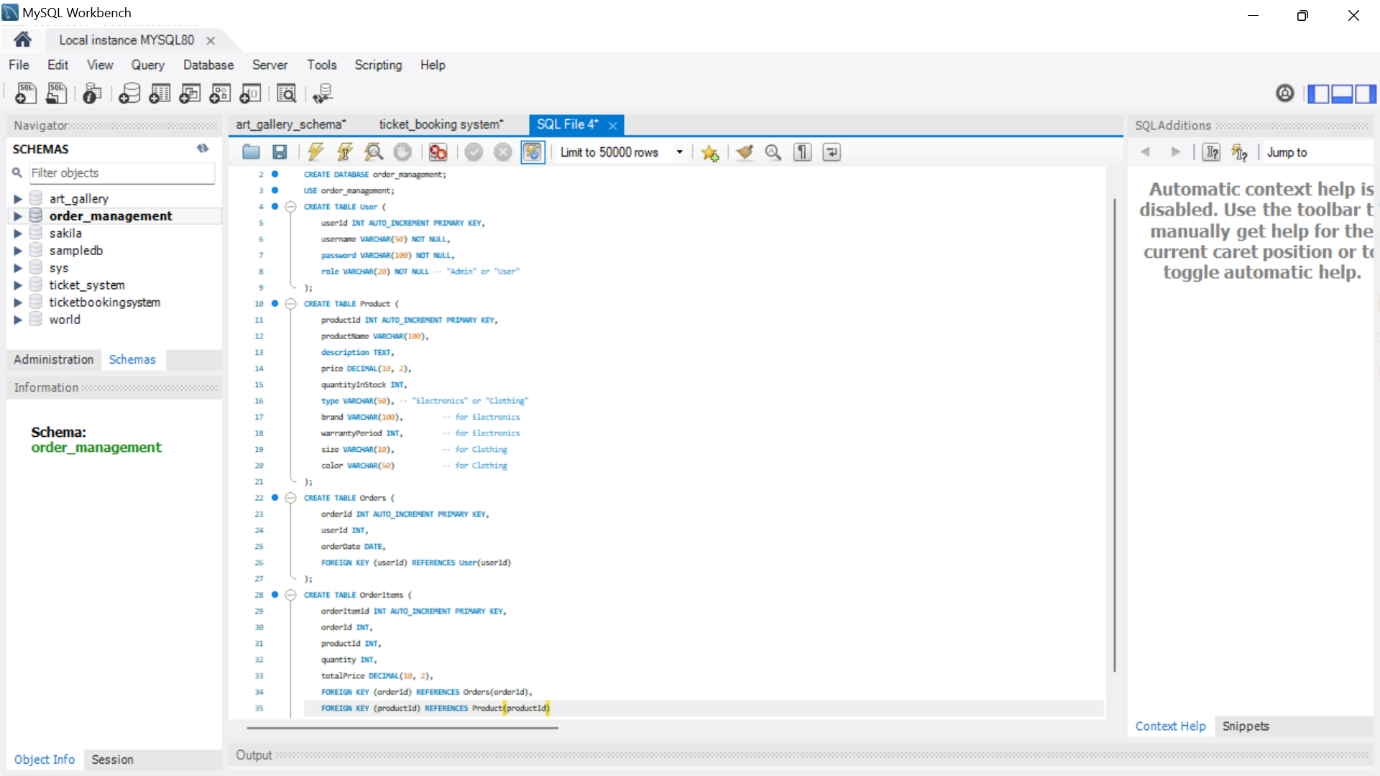
productId INT,

quantity INT,

totalPrice DECIMAL(10, 2),

FOREIGN KEY (orderId) REFERENCES Orders(orderId),

FOREIGN KEY (productId) REFERENCES Product(productId)

);

**Step 2: Entity Classes in Python (vs code) customer, product , order, order\_item**

1. **entity/user.py**

class User:

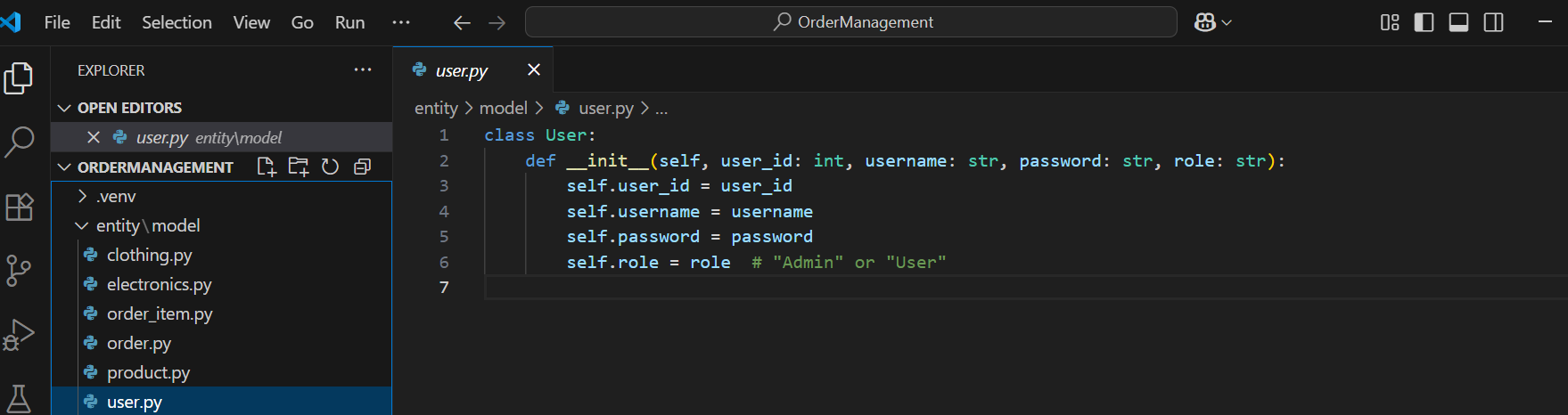
def \_\_init\_\_(self, user\_id: int, username: str, password: str, role: str):

self.user\_id = user\_id

self.username = username

self.password = password

self.role = role # "Admin" or "User"



1. **entity/order.py**

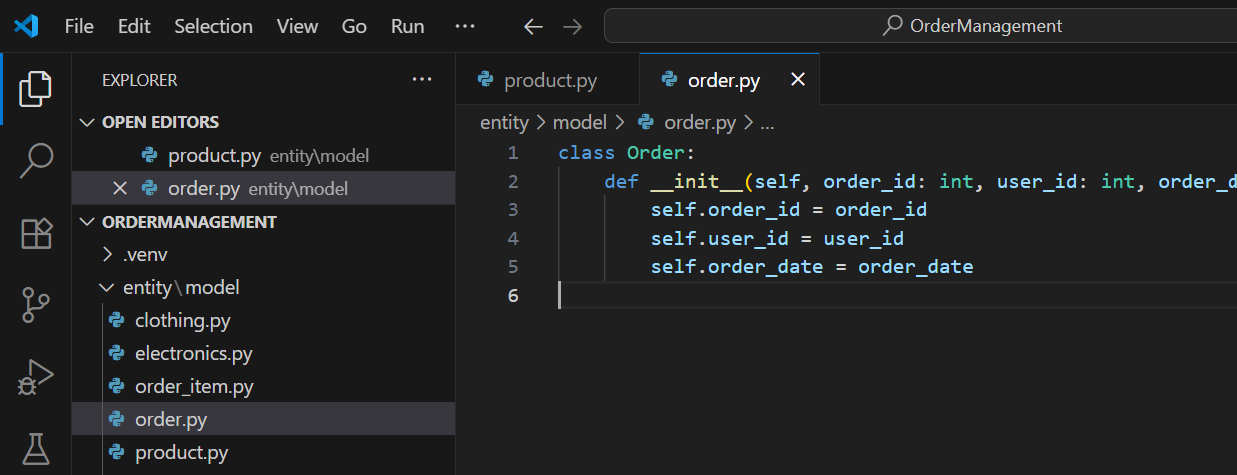
class Order:

def \_\_init\_\_(self, order\_id: int, user\_id: int, order\_date: str):

self.order\_id = order\_id

self.user\_id = user\_id

self.order\_date = order\_date



1. **entity/order\_item.py**

class OrderItem:

    def \_\_init\_\_(self, order\_item\_id: int, order\_id: int, product\_id: int, quantity: int, total\_price: float):

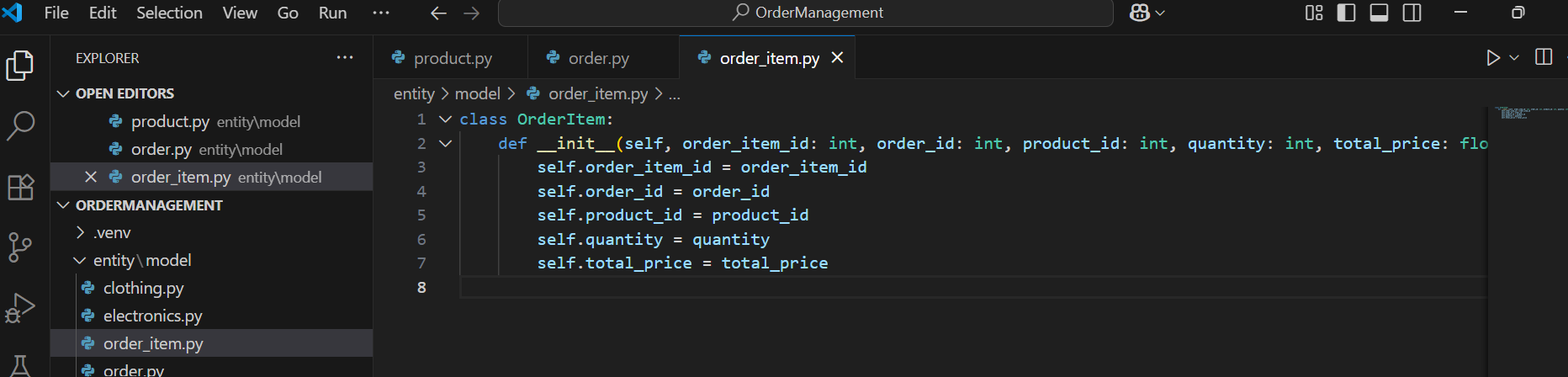
        self.order\_item\_id = order\_item\_id

        self.order\_id = order\_id

        self.product\_id = product\_id

        self.quantity = quantity

        self.total\_price = total\_price



1. **entity/model/product.py**

class Product:

def \_\_init\_\_(self, product\_id: int, product\_name: str, description: str,

price: float, quantity\_in\_stock: int, type: str):

self.product\_id = product\_id

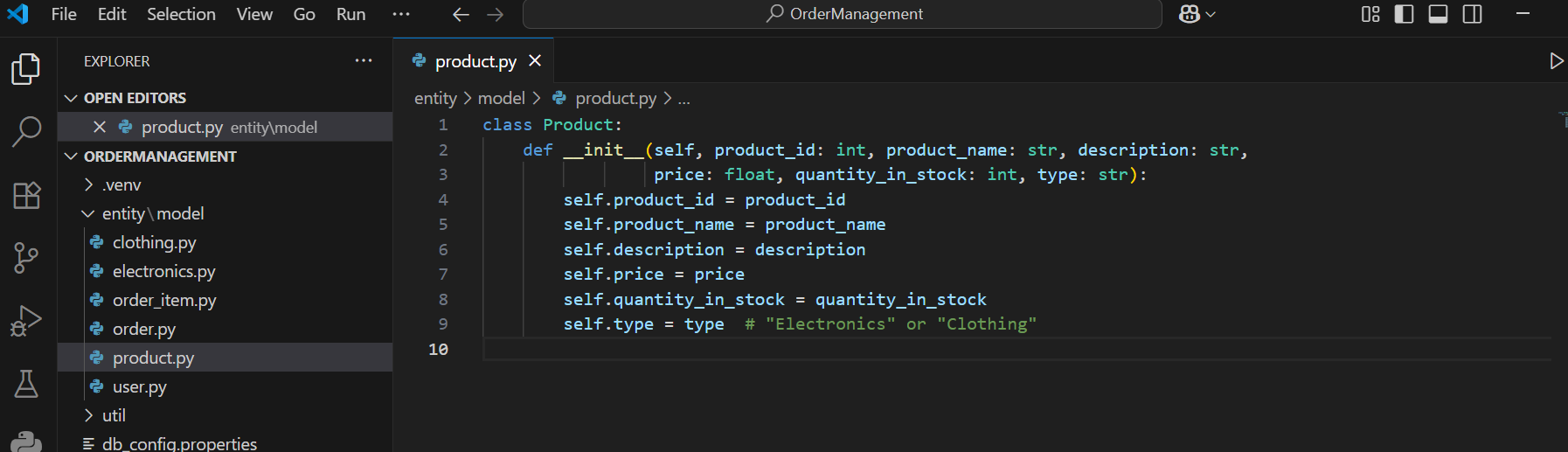
self.product\_name = product\_name

self.description = description

self.price = price

self.quantity\_in\_stock = quantity\_in\_stock

self.type = type # "Electronics" or "Clothing"



1. **entity/model/electronics.py**

from entity.model.product import Product

class Electronics(Product):

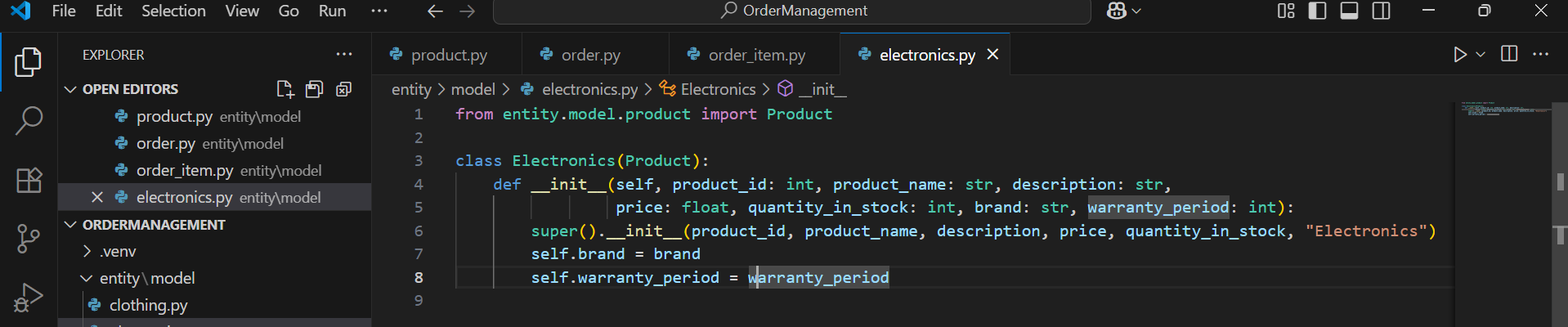
def \_\_init\_\_(self, product\_id: int, product\_name: str, description: str,

price: float, quantity\_in\_stock: int, brand: str, warranty\_period: int):

super().\_\_init\_\_(product\_id, product\_name, description, price, quantity\_in\_stock, "Electronics")

self.brand = brand

self.warranty\_period = warranty\_period



1. **entity/model/clothing.py**

from entity.model.product import Product

class Clothing(Product):

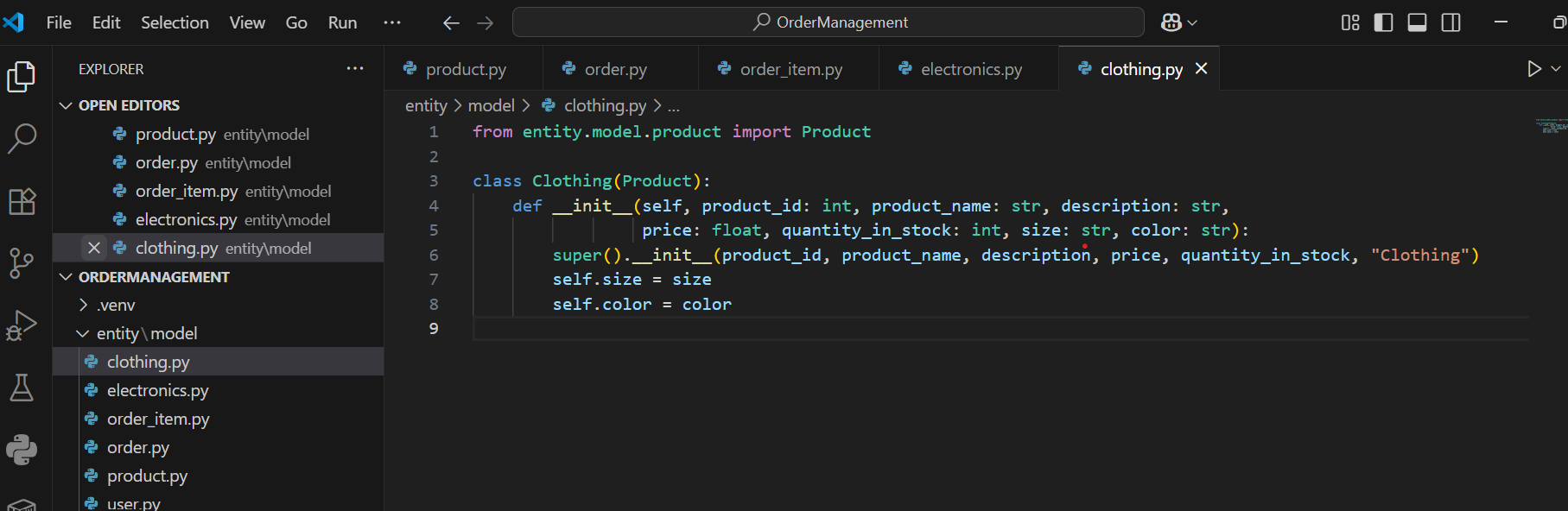
def \_\_init\_\_(self, product\_id: int, product\_name: str, description: str,

price: float, quantity\_in\_stock: int, size: str, color: str):

super().\_\_init\_\_(product\_id, product\_name, description, price, quantity\_in\_stock, "Clothing")

self.size = size

self.color = color

****

**STEP 3: Create IOrderManagementRepository Interface (dao/ folder)**

Defining the interface (abstract class) that declares all the service methods like:

* Add Product
* Place Order
* View Orders
* Cancel Order
  1. **Create file: dao/i\_order\_management\_repository.py**

from abc import ABC, abstractmethod

from entity.model.user import User

from entity.model.product import Product

from entity.model.order import Order

class IOrderManagementRepository(ABC):

@abstractmethod

def create\_user(self, user: User) -> bool:

pass

@abstractmethod

def create\_product(self, user: User, product: Product) -> bool:

pass

@abstractmethod

def create\_order(self, user: User, product\_list: list[Product]) -> bool:

pass

@abstractmethod

def cancel\_order(self, user\_id: int, order\_id: int) -> bool:

pass

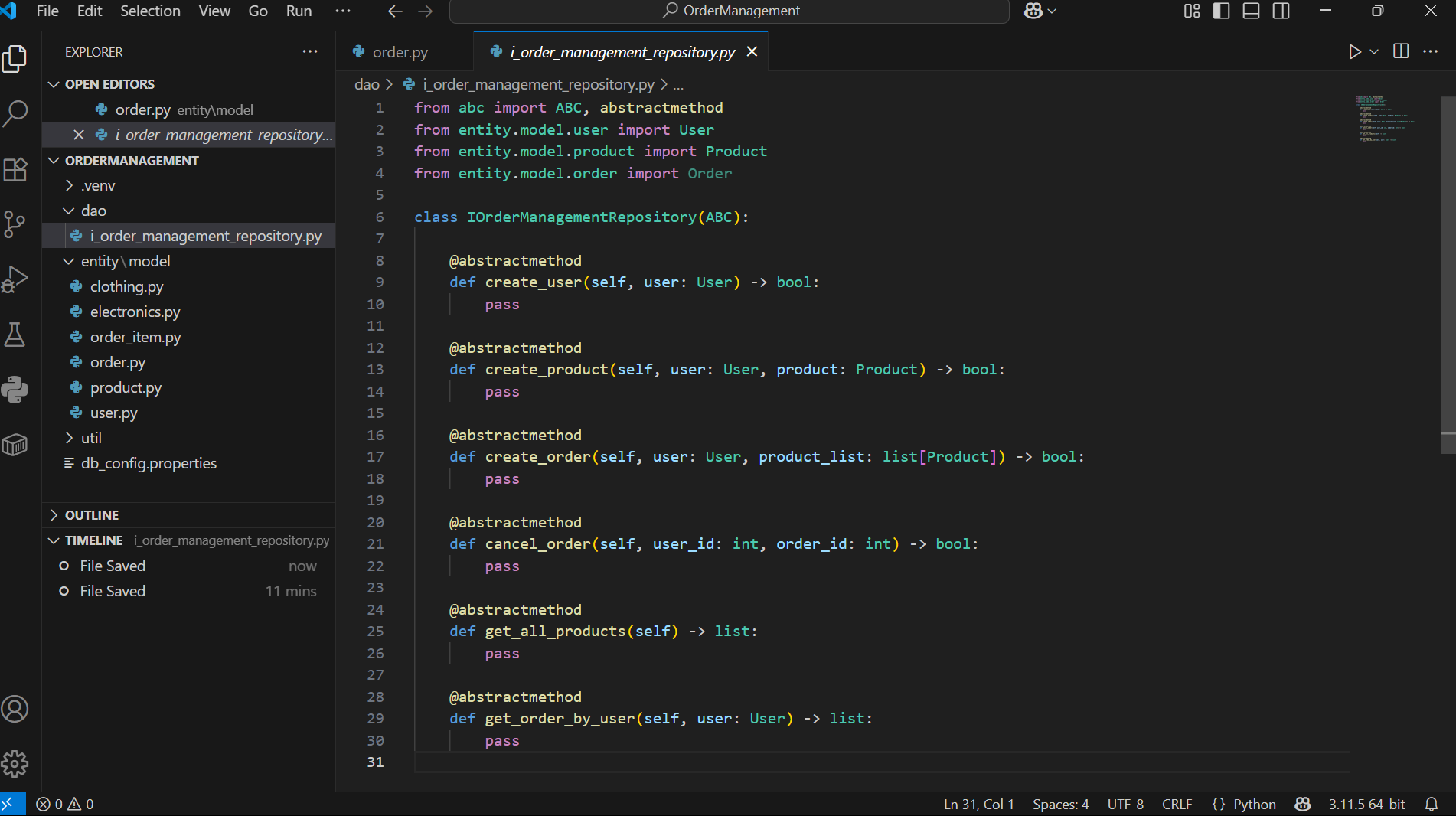
@abstractmethod

def get\_all\_products(self) -> list:

pass

@abstractmethod

def get\_order\_by\_user(self, user: User) -> list:

pass pass

**Step 4: Service Implementation (real DB code using MySQL)**

This class implements all the methods declared in the service interface. It connects to MySQL and performs all core operations like placing orders, adding products/customers, viewing and cancelling orders.

**Code : 1. OrderProcessor Service Implementation dao/ order\_processor**

import mysql.connector

from datetime import date

from dao.i\_order\_management\_repository import IOrderManagementRepository

from entity.model.user import User

from entity.model.product import Product

from entity.model.order import Order

from exception.user\_not\_found\_exception import UserNotFoundException

from exception.order\_not\_found\_exception import OrderNotFoundException

from util.db\_conn\_util import get\_connection

class OrderProcessor(IOrderManagementRepository):

def \_\_init\_\_(self, config\_path):

self.connection = get\_connection(config\_path)

self.cursor = self.connection.cursor(dictionary=True)

def create\_user(self, user: User) -> bool:

try:

self.cursor.execute("INSERT INTO User (username, password, role) VALUES (%s, %s, %s)",

(user.username, user.password, user.role))

self.connection.commit()

return True

except Exception as e:

print("Error creating user:", e)

return False

def create\_product(self, user: User, product: Product) -> bool:

try:

# Check if user is admin

self.cursor.execute("SELECT \* FROM User WHERE userId = %s AND role = 'Admin'", (user.user\_id,))

if not self.cursor.fetchone():

raise UserNotFoundException("Admin user not found.")

self.cursor.execute("""INSERT INTO Product

(productName, description, price, quantityInStock, type, brand, warrantyPeriod, size, color)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)""",

(product.product\_name, product.description, product.price, product.quantity\_in\_stock,

product.type, getattr(product, 'brand', None), getattr(product, 'warranty\_period', None),

getattr(product, 'size', None), getattr(product, 'color', None)))

self.connection.commit()

return True

except Exception as e:

print("Error creating product:", e)

return False

def create\_order(self, user: User, product\_list: list[Product]) -> bool:

try:

self.cursor.execute("SELECT \* FROM User WHERE userId = %s", (user.user\_id,))

if not self.cursor.fetchone():

raise UserNotFoundException("User not found.")

today = date.today()

self.cursor.execute("INSERT INTO Orders (userId, orderDate) VALUES (%s, %s)",

(user.user\_id, today))

order\_id = self.cursor.lastrowid

for product in product\_list:

self.cursor.execute("SELECT price FROM Product WHERE productId = %s", (product.product\_id,))

result = self.cursor.fetchone()

if result:

total\_price = result['price'] \* product.quantity\_in\_stock

self.cursor.execute("""INSERT INTO OrderItems (orderId, productId, quantity, totalPrice)

VALUES (%s, %s, %s, %s)""",

(order\_id, product.product\_id, product.quantity\_in\_stock, total\_price))

self.connection.commit()

return True

except Exception as e:

print("Error creating order:", e)

self.connection.rollback()

return False

def cancel\_order(self, user\_id: int, order\_id: int) -> bool:

try:

self.cursor.execute("SELECT \* FROM Orders WHERE orderId = %s AND userId = %s",

(order\_id, user\_id))

if not self.cursor.fetchone():

raise OrderNotFoundException("Order not found for given user.")

self.cursor.execute("DELETE FROM OrderItems WHERE orderId = %s", (order\_id,))

self.cursor.execute("DELETE FROM Orders WHERE orderId = %s", (order\_id,))

self.connection.commit()

return True

except Exception as e:

print("Error cancelling order:", e)

self.connection.rollback()

return False

def get\_all\_products(self) -> list:

try:

self.cursor.execute("SELECT \* FROM Product")

return self.cursor.fetchall()

except Exception as e:

print("Error retrieving products:", e)

return []

def get\_order\_by\_user(self, user: User) -> list:

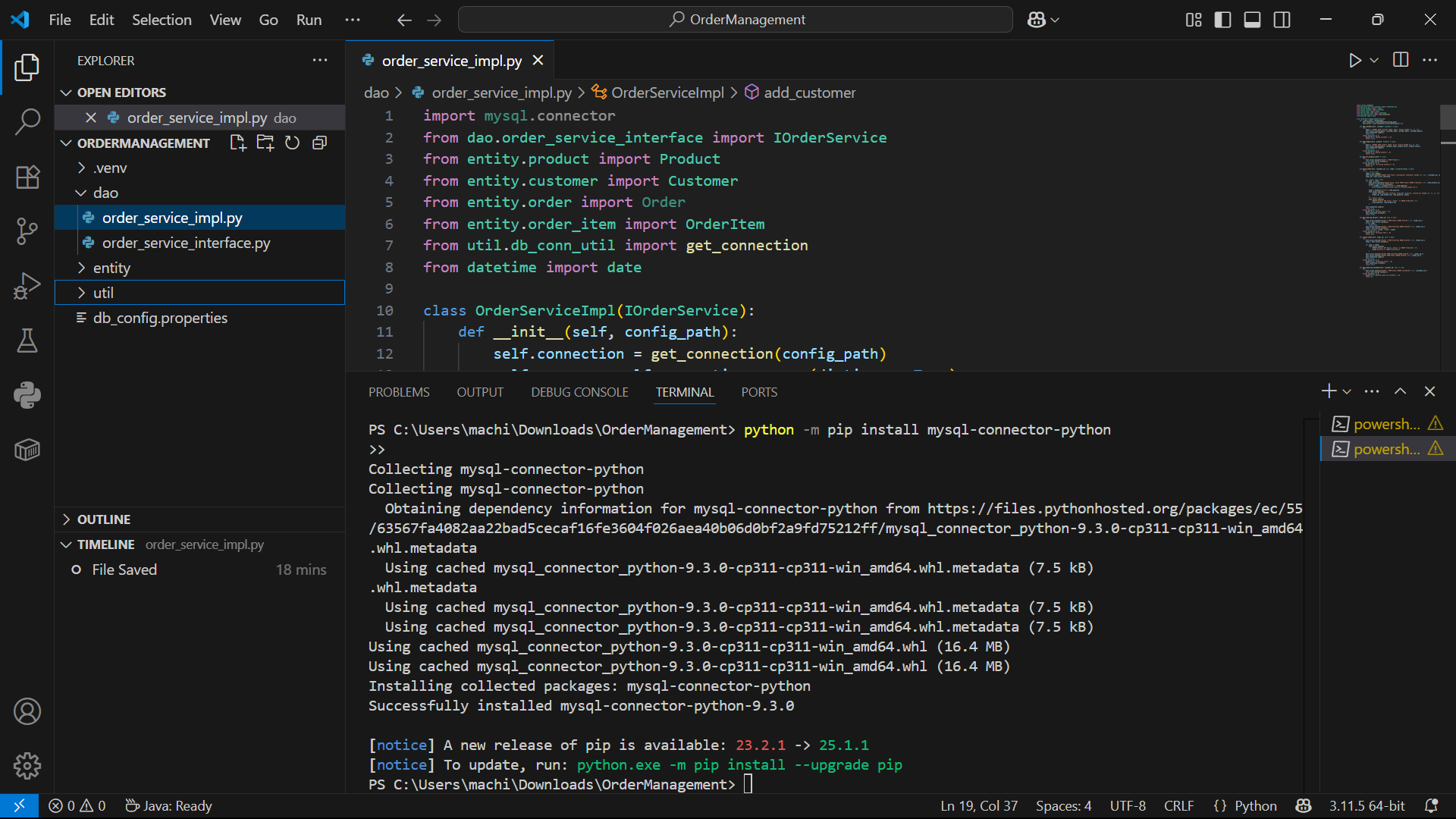
try:

self.cursor.execute("SELECT \* FROM Orders WHERE userId = %s", (user.user\_id,))

return self.cursor.fetchall()

except Exception as e:

print("Error fetching user orders:", e)

return []

**This class implements all the methods declared in the service interface. It connects to MySQL and performs all core operations like placing orders, adding products/customers, viewing and cancelling orders.**

**Step 5 – Database Connection Utilities**

**To ensure reusable and modular code, utility classes were created for loading DB credentials and establishing MySQL connections. This follows best practices in clean architecture.**

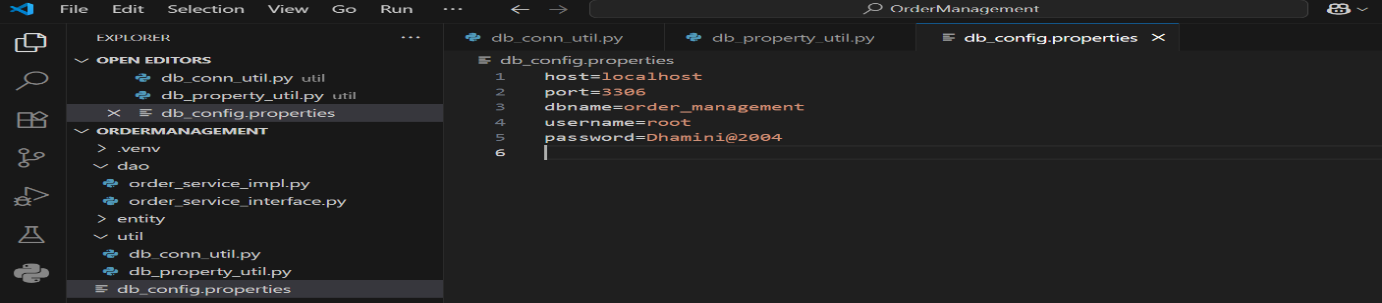
1. db\_config.properties

host=localhost

port=3306

dbname=order\_management

username=root

password=password

1. util/db\_property\_util.py

def get\_property\_string(file\_path: str) -> dict:

props = {}

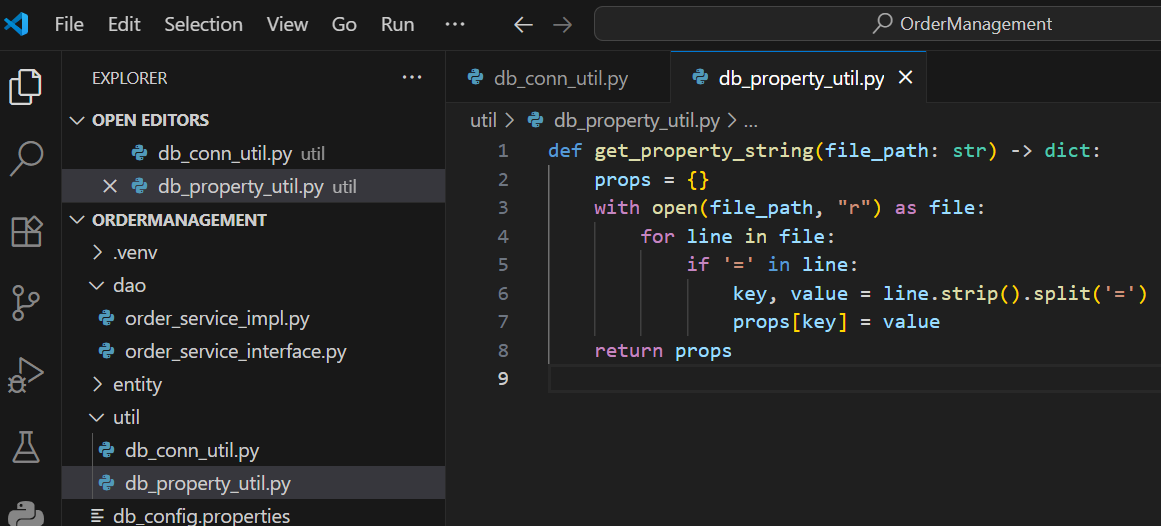
with open(file\_path, "r") as file:

for line in file:

if '=' in line:

key, value = line.strip().split('=')

props[key] = value

 return props

1. util/db\_conn\_util.py

import mysql.connector

from util.db\_property\_util import get\_property\_string

def get\_connection(file\_path: str):

props = get\_property\_string(file\_path)

conn = mysql.connector.connect(

host=props['host'],

port=props['port'],

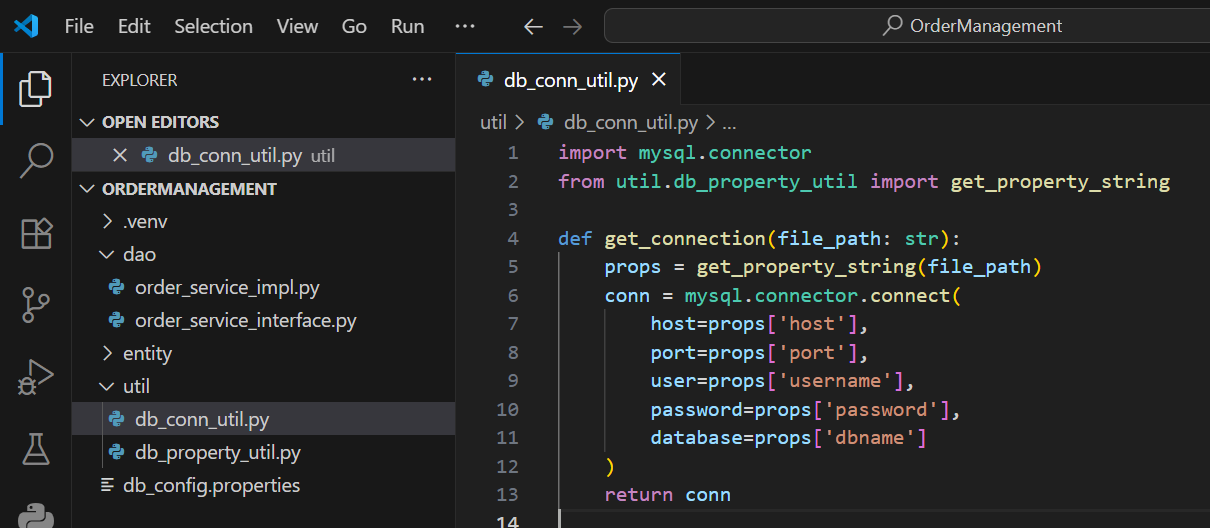
user=props['username'],

password=props['password'],

database=props['dbname']

)

return conn

****

**STEP 6: Custom Exceptions (exception/ folder)**

**To follow good object-oriented practices, custom exceptions were created for user and order not found scenarios. These make the application easier to debug and maintain.File: 1. exception/user\_not\_found\_exception.py**

class UserNotFoundException(Exception):

def \_\_init\_\_(self, message="User not found."):

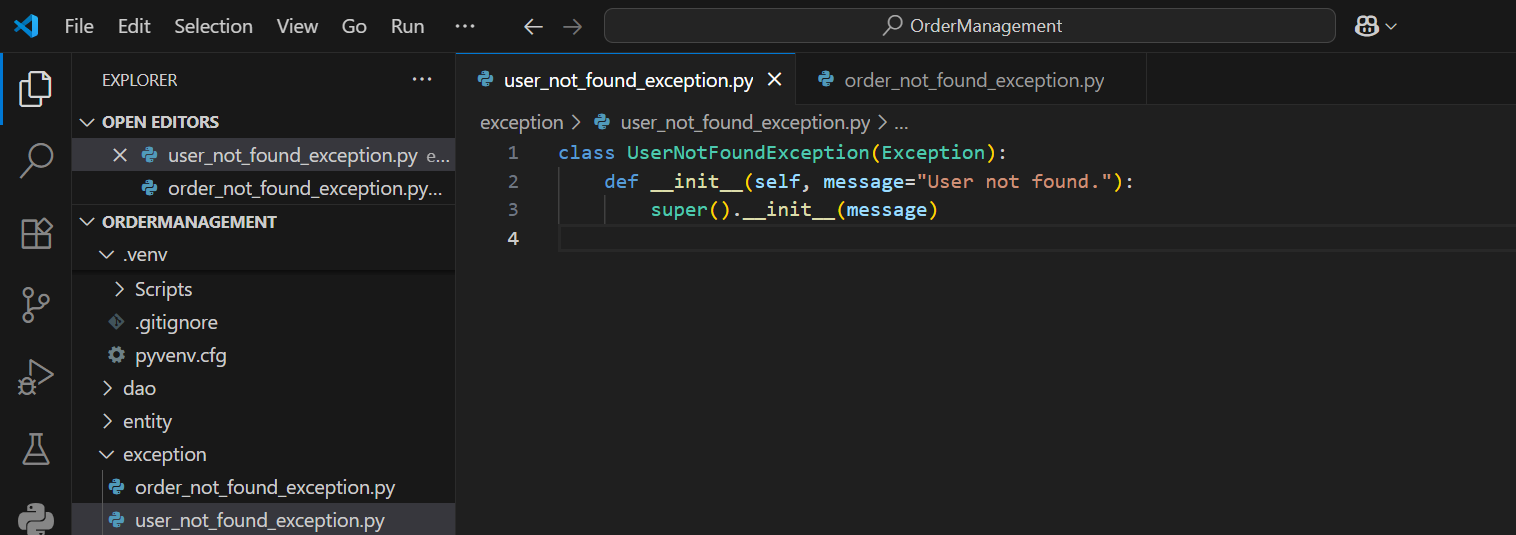
super().\_\_init\_\_(message)

* 1. **File: exception/order\_not\_found\_exception.py**

class OrderNotFoundException(Exception):

def \_\_init\_\_(self, message="Order not found."):

super().\_\_init\_\_(message)





**STEP 7: CLI Menu Interface (main/main\_module.py)**

This is the menu where users (admin or regular) can:

* Add users or products
* Place orders
* Cancel orders
* View all products
* View user orders

1. **main/main\_module.py**

from dao.order\_processor import OrderProcessor

from entity.model.user import User

from entity.model.product import Product

from exception.user\_not\_found\_exception import UserNotFoundException

from exception.order\_not\_found\_exception import OrderNotFoundException

def main():

service = OrderProcessor("db\_config.properties")

while True:

print("\n===== ORDER MANAGEMENT MENU =====")

print("1. Create User")

print("2. Create Product")

print("3. Place Order")

print("4. Cancel Order")

print("5. View All Products")

print("6. View Orders by User")

print("7. Exit")

print("==================================")

choice = input("Enter your choice: ")

if choice == "1":

username = input("Username: ")

password = input("Password: ")

role = input("Role (Admin/User): ")

user = User(0, username, password, role)

print("User created successfully." if service.create\_user(user) else "Failed to create user.")

elif choice == "2":

admin\_id = int(input("Enter Admin User ID: "))

name = input("Product Name: ")

desc = input("Description: ")

price = float(input("Price: "))

stock = int(input("Quantity in Stock: "))

ptype = input("Type (Electronics/Clothing): ")

if ptype == "Electronics":

brand = input("Brand: ")

warranty = int(input("Warranty Period (months): "))

from entity.model.electronics import Electronics

product = Electronics(0, name, desc, price, stock, brand, warranty)

elif ptype == "Clothing":

size = input("Size: ")

color = input("Color: ")

from entity.model.clothing import Clothing

product = Clothing(0, name, desc, price, stock, size, color)

else:

print("Invalid type.")

continue

admin\_user = User(admin\_id, "", "", "Admin")

try:

result = service.create\_product(admin\_user, product)

print("Product created successfully." if result else "Failed to create product.")

except UserNotFoundException as e:

print("Error:", e)

elif choice == "3":

user\_id = int(input("Enter User ID: "))

product\_ids = input("Enter product IDs (comma separated): ").split(',')

product\_list = []

for pid in product\_ids:

pid = int(pid.strip())

qty = int(input(f"Quantity for Product ID {pid}: "))

product = Product(pid, "", "", 0, qty, "")

product\_list.append(product)

user = User(user\_id, "", "", "")

try:

success = service.create\_order(user, product\_list)

print("Order placed successfully." if success else "Failed to place order.")

except UserNotFoundException as e:

print("Error:", e)

elif choice == "4":

uid = int(input("Enter User ID: "))

oid = int(input("Enter Order ID to cancel: "))

try:

if service.cancel\_order(uid, oid):

print("Order cancelled successfully.")

else:

print("Failed to cancel order.")

except OrderNotFoundException as e:

print("Error:", e)

elif choice == "5":

products = service.get\_all\_products()

for p in products:

print(p)

elif choice == "6":

user\_id = int(input("Enter User ID: "))

user = User(user\_id, "", "", "")

orders = service.get\_order\_by\_user(user)

for order in orders:

print(order)

elif choice == "7":

print("Exiting application.")

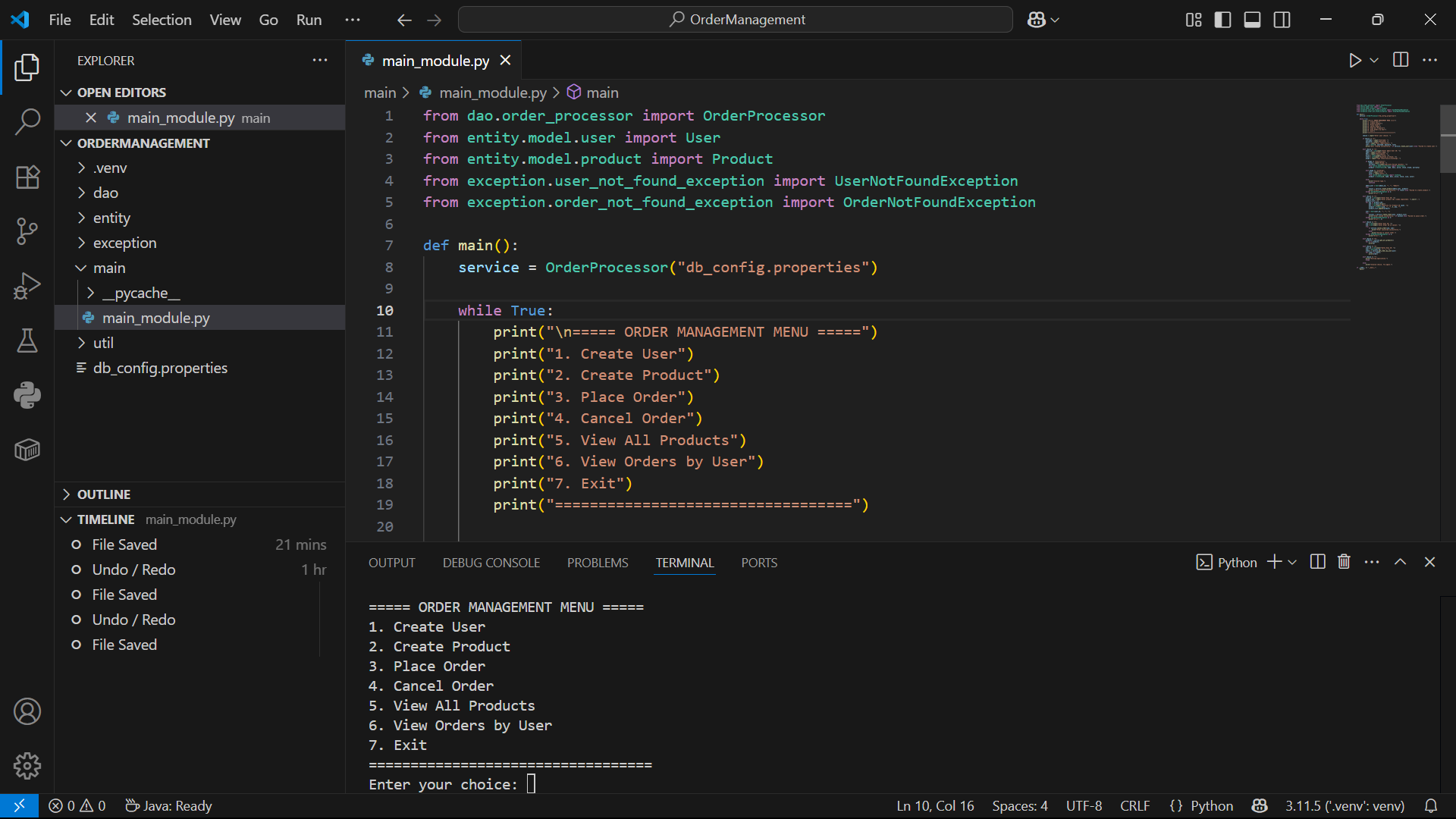
break

else:

print("Invalid choice. Try again.")

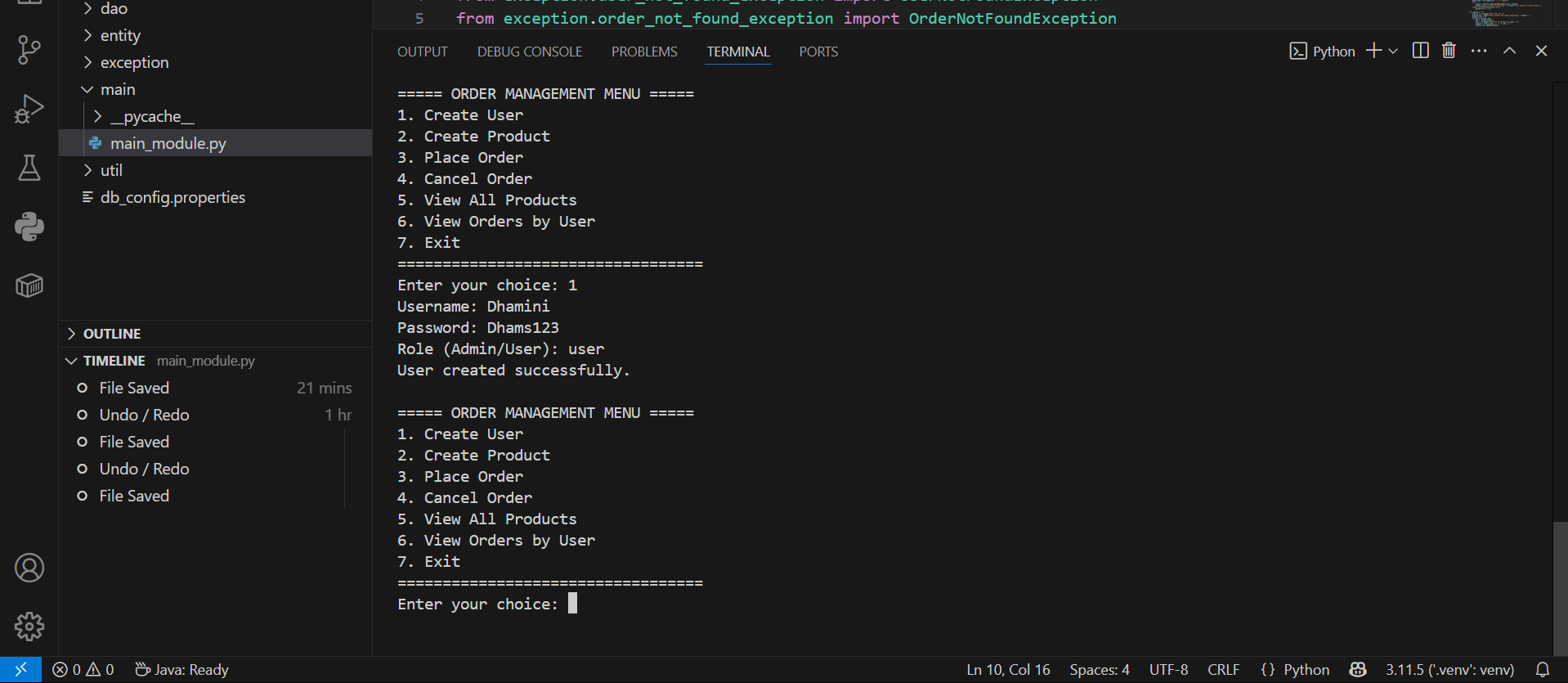
if \_\_name\_\_ == "\_\_main\_\_":

main()

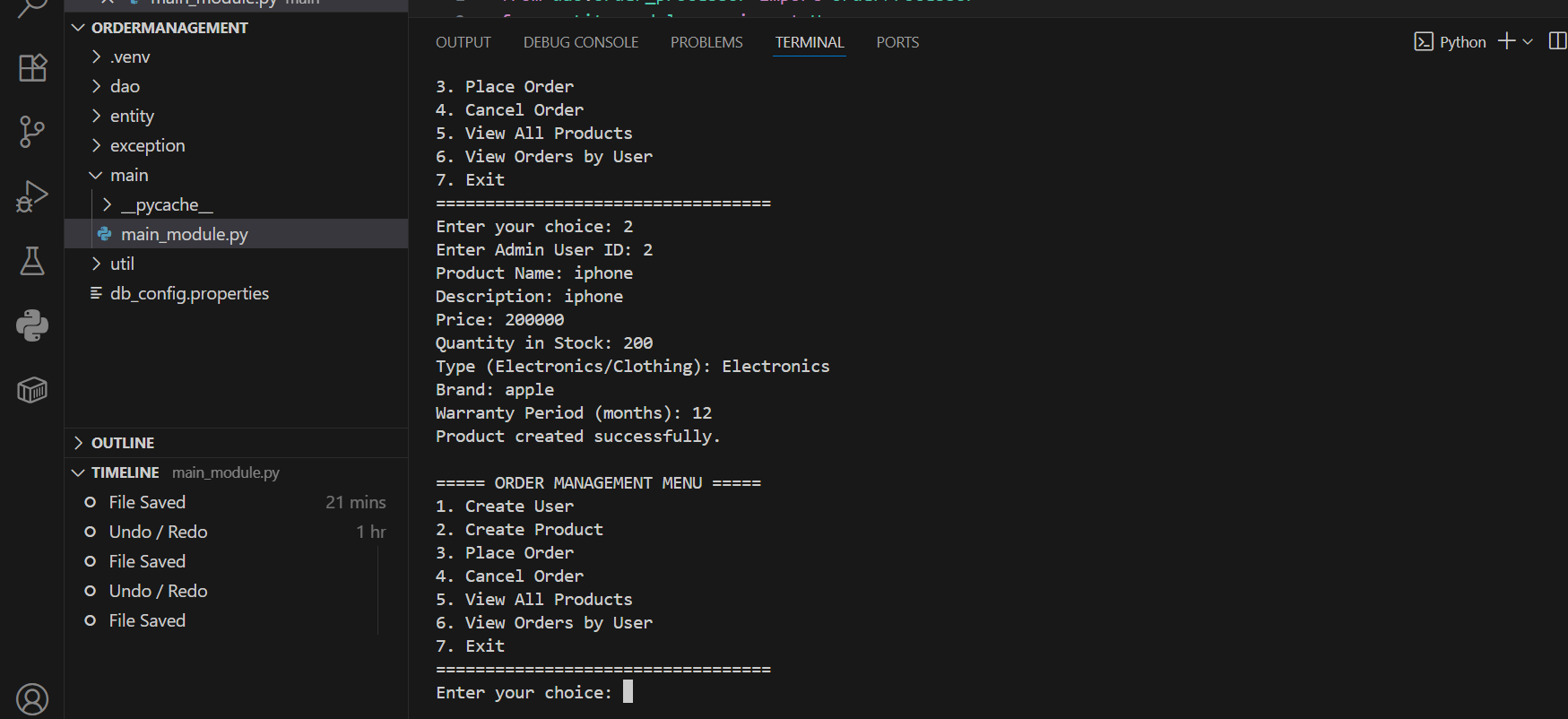


Output :

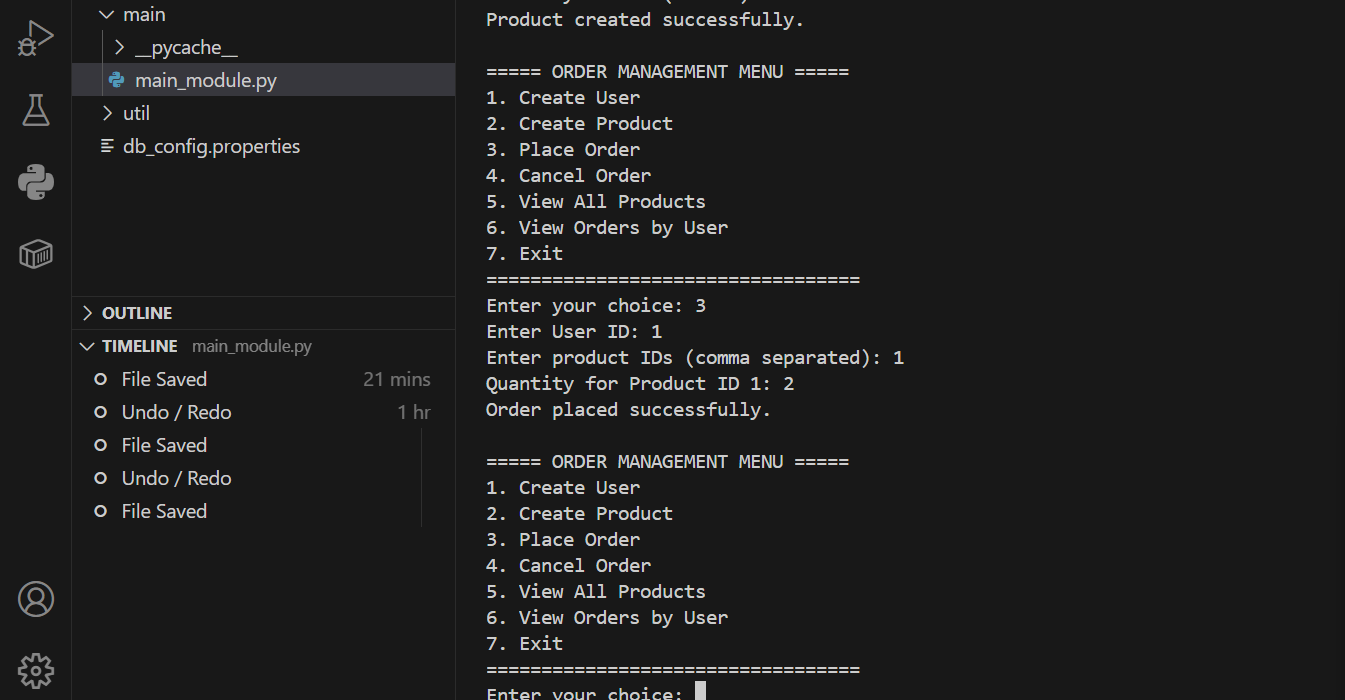
* 1. User creation

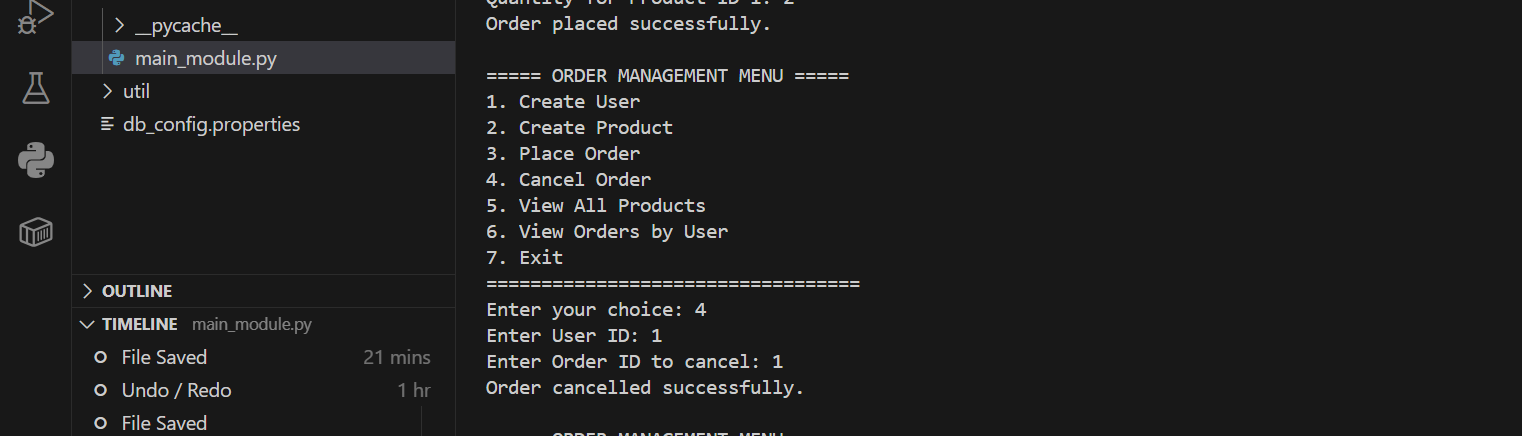
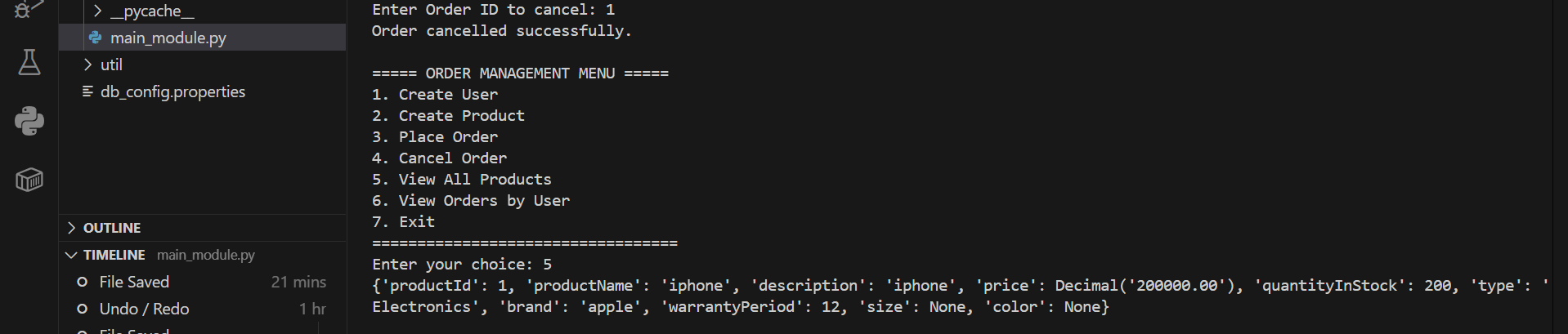
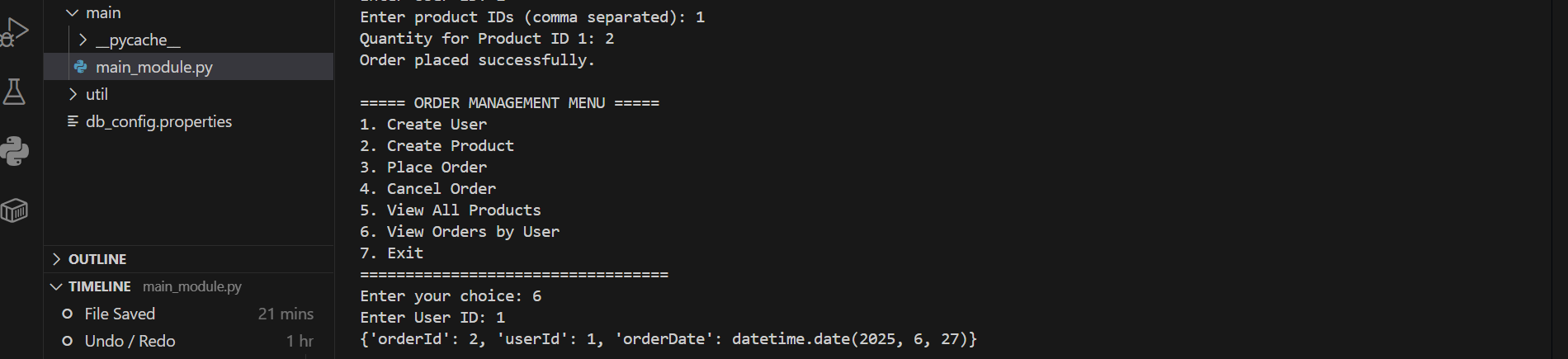
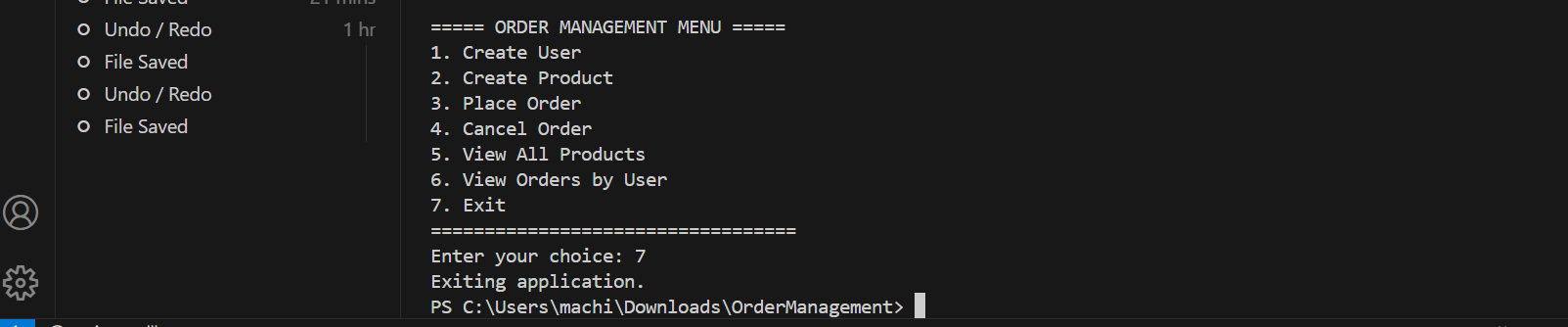


* 1. Product creation



* 1. Place order



* 1. cancel order
  2. View all products
  3. View order by user
  4. Exit