Dhamini Reddy

PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)

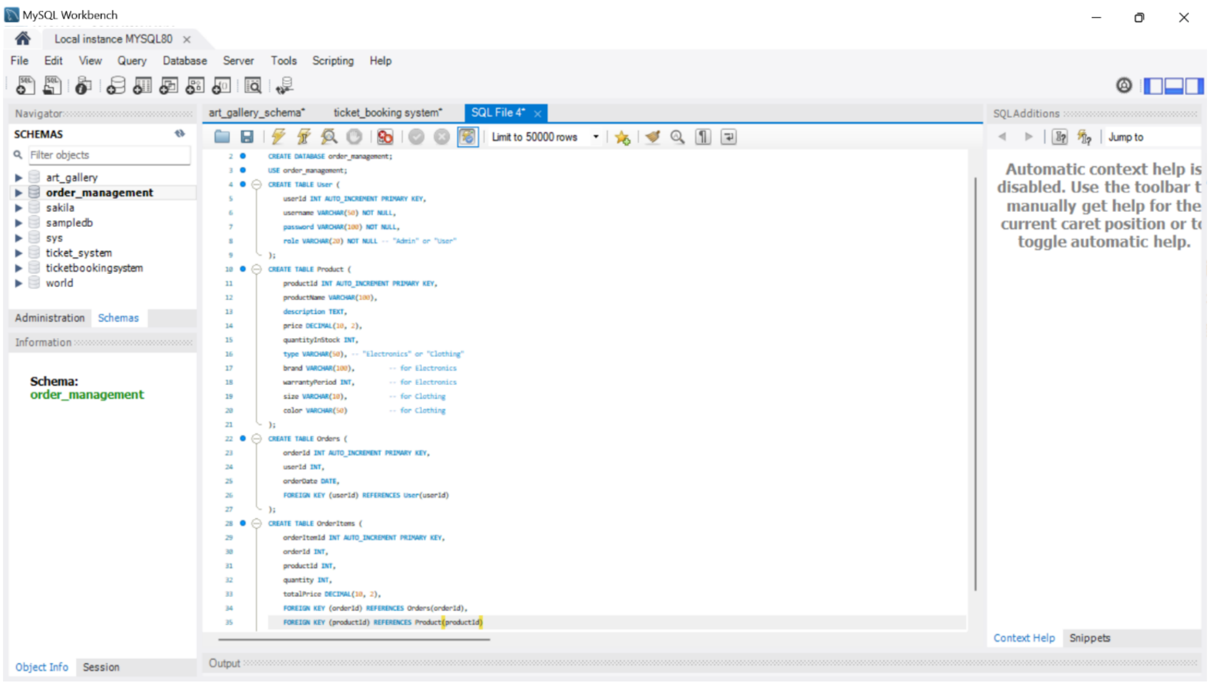
Order management system :

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)
);
```

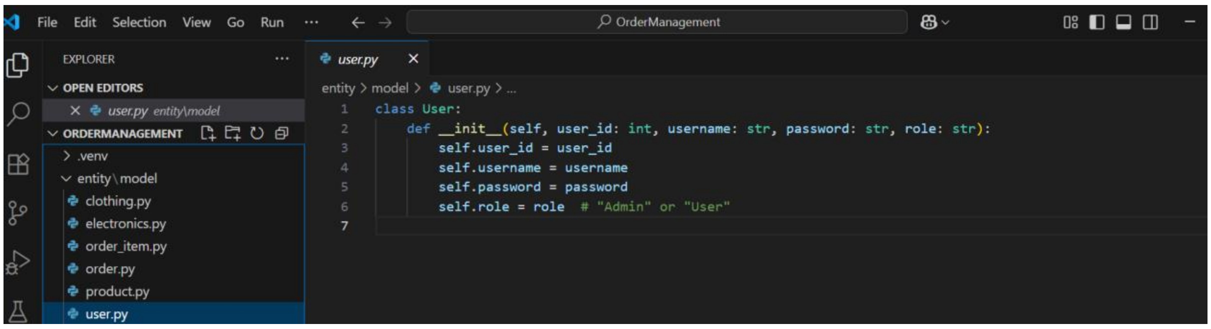


```sql
CREATE DATABASE order_management;
USE order_management;
CREATE TABLE User (
    userId INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL -- "Admin" or "User"
);
CREATE TABLE Product (
    productId INT AUTO_INCREMENT 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**

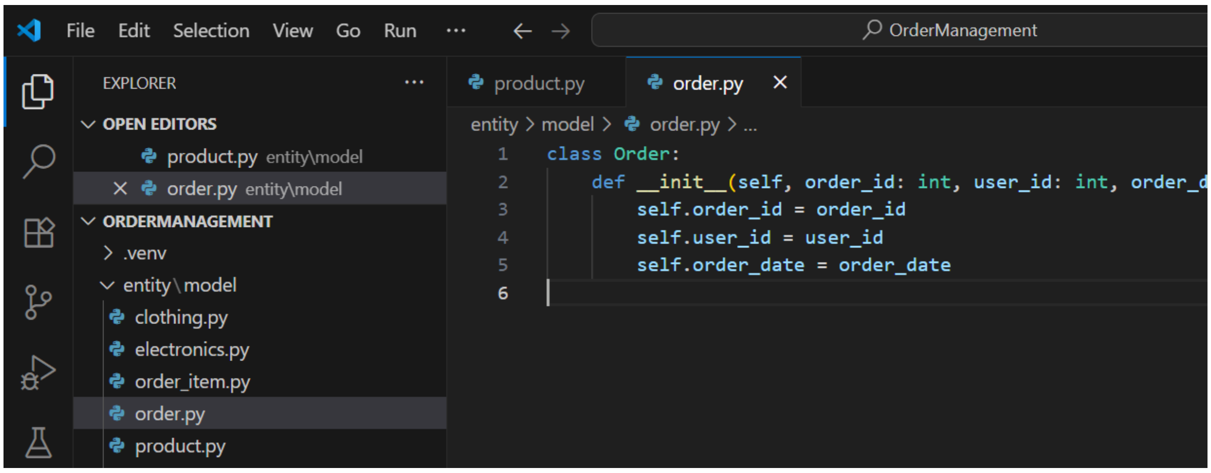
```python
class User:
    def __init__(self, user_id: int, username: str, password: str, role: str):
        self.user_id = user_id
        self.username = username
        self.password = password
        self.role = role  # "Admin" or "User"
```



```python
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"
```

2. **entity/order.py**

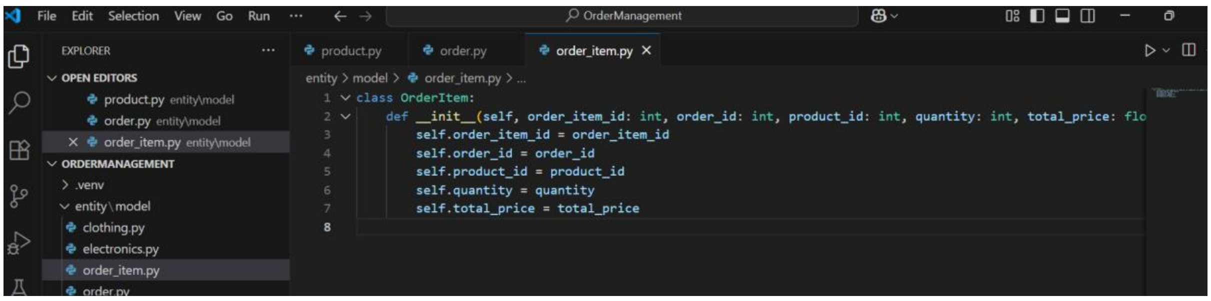
```
class Order:
    def __init__(self, order_id: int, user_id: int, order_date: str):
        self.order_id = order_id
        self.user_id = user_id
        self.order_date = order_date
```



```
class Order:
    def __init__(self, order_id: int, user_id: int, order_d
        self.order_id = order_id
        self.user_id = user_id
        self.order_date = order_date
```

3. **entity/order_item.py**

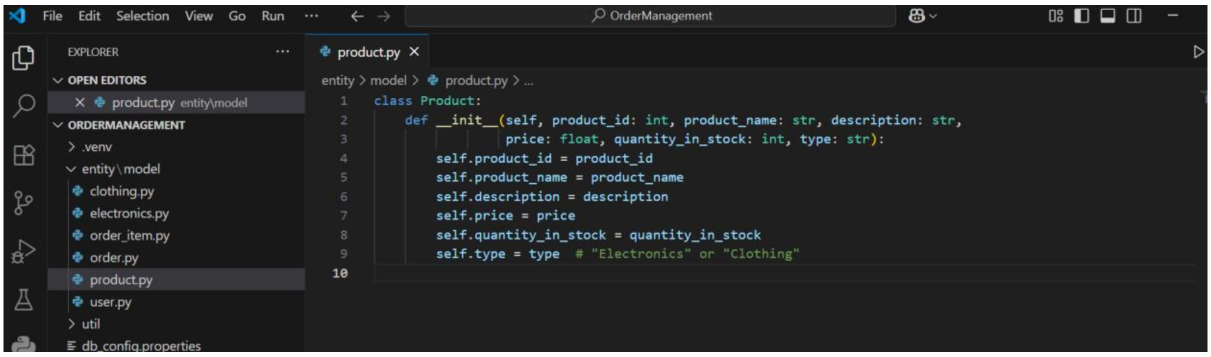
```
class OrderItem:

   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
```



```
class OrderItem:
    def __init__(self, order_item_id: int, order_id: int, product_id: int, quantity: int, total_price: flo
        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
```

4. **entity/model/product.py**

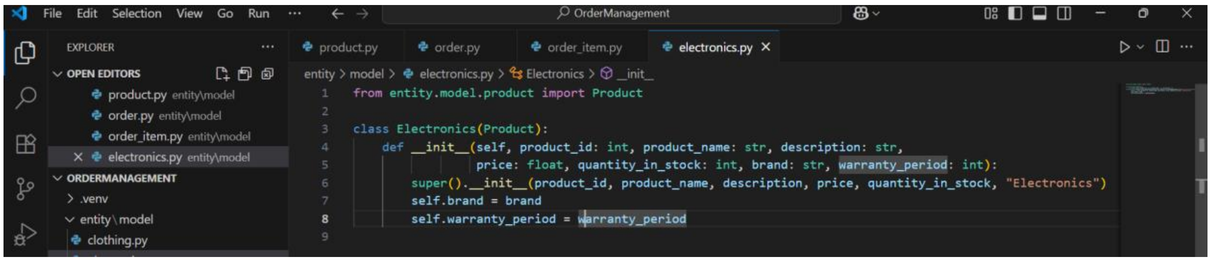
```
class Product:
    def __init__(self, product_id: int, product_name: str, description: str,
                 price: float, quantity_in_stock: int, type: str):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.quantity_in_stock = quantity_in_stock
        self.type = type  # "Electronics" or "Clothing"
```



```
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"
```

5. **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
```



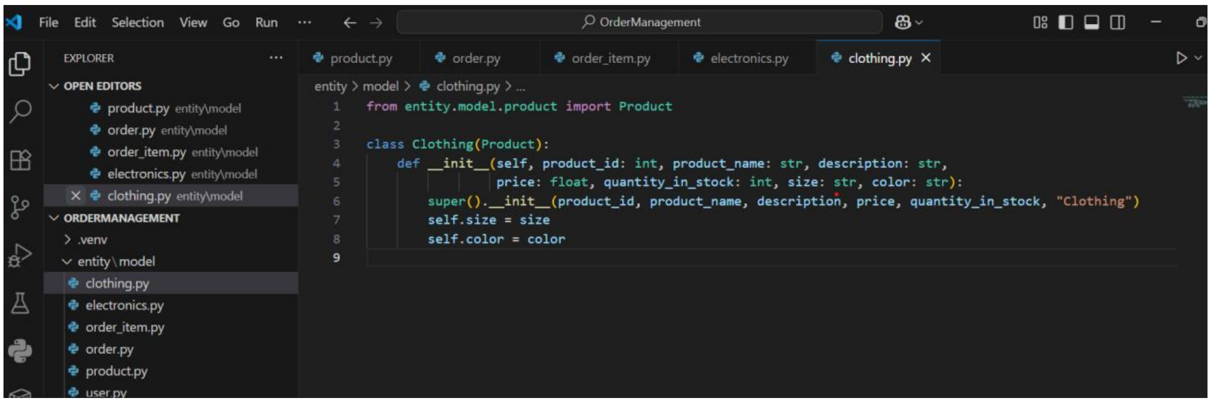
```
from entity.model.product import Product

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
```

6. **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
```

Dhamini Reddy

PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)



```python
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)

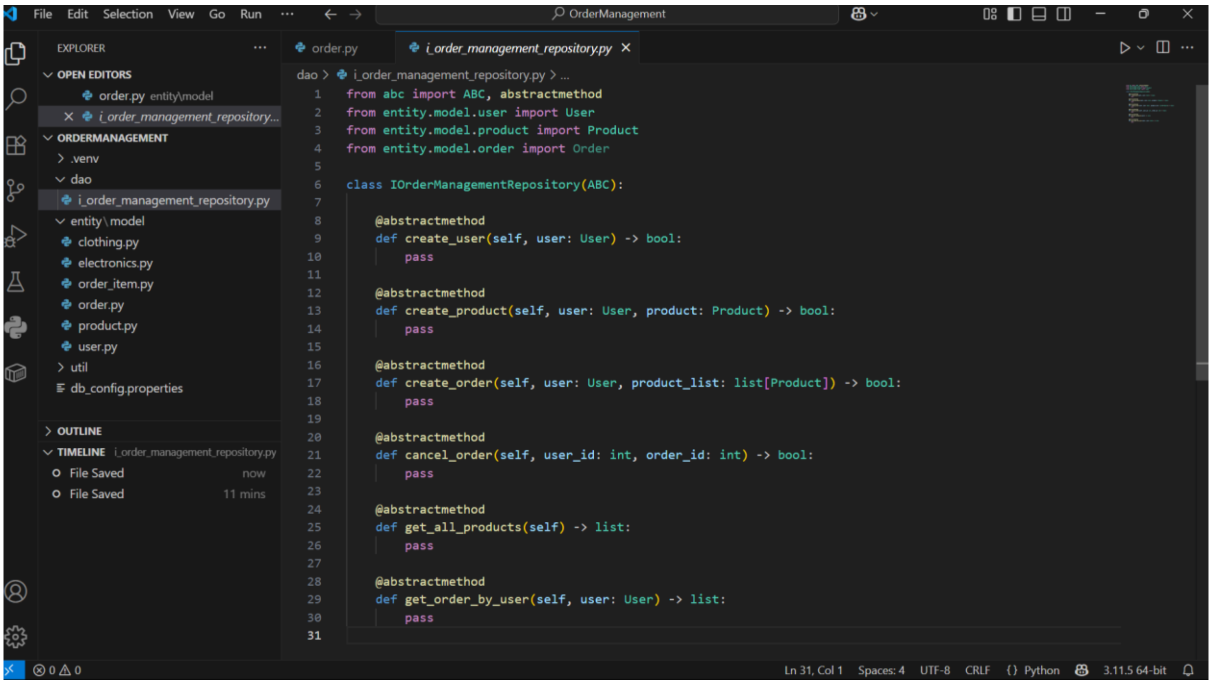
Defining the interface (abstract class) that declares all the service methods like:

- Add Product
- Place Order
- View Orders
- Cancel Order

1. **Create file: dao/i_order_management_repository.py**

```python
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
```


File Edit Selection View Go Run ...
OrderManagement
EXPLORER
OPEN EDITORS
order.py entity\model
i_order_management_repository...
ORDERMANAGEMENT
.venv
dao
i_order_management_repository.py
entity\model
clothing.py
electronics.py
order_item.py
order.py
product.py
user.py
util
db_config.properties
OUTLINE
TIMELINE i_order_management_repository.py
File Saved now
File Saved 11 mins
order.py
i_order_management_repository.py
dao > i_order_management_repository.py > ...
Ln 31, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.5 64-bit

```
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

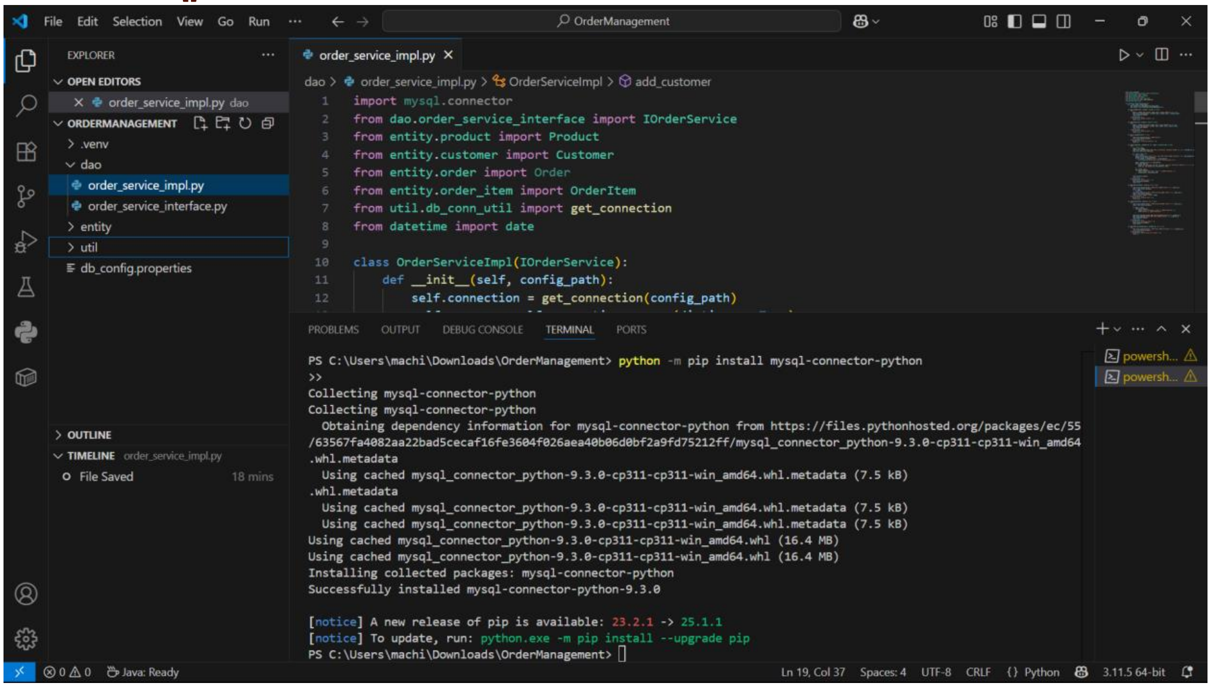
Step 4: Service Implementation (real DB code using MySQL)

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 []
```



```
import mysql.connector
from dao.order_service_interface import IOrderService
from entity.product import Product
from entity.customer import Customer
from entity.order import Order
from entity.order_item import OrderItem
from util.db_conn_util import get_connection
from datetime import date

class OrderServiceImpl(IOrderService):
    def __init__(self, config_path):
        self.connection = get_connection(config_path)
```

```
PS C:\Users\machi\Downloads\OrderManagement> python -m pip install mysql-connector-python
>>
Collecting mysql-connector-python
Collecting mysql-connector-python
  Obtaining dependency information for mysql-connector-python from https://files.pythonhosted.org/packages/ec/55
/63567fa4082aa22bad5cecaf16fe3604f026aea40b06d0bf2a9fd75212ff/mysql_connector_python-9.3.0-cp311-cp311-win_amd64
.whl.metadata
  Using cached mysql_connector_python-9.3.0-cp311-cp311-win_amd64.whl.metadata (7.5 kB)
.whl.metadata
  Using cached mysql_connector_python-9.3.0-cp311-cp311-win_amd64.whl.metadata (7.5 kB)
  Using cached mysql_connector_python-9.3.0-cp311-cp311-win_amd64.whl.metadata (7.5 kB)
Using cached mysql_connector_python-9.3.0-cp311-cp311-win_amd64.whl (16.4 MB)
Using cached mysql_connector_python-9.3.0-cp311-cp311-win_amd64.whl (16.4 MB)
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.3.0

[notice] A new release of pip is available: 23.2.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\machi\Downloads\OrderManagement>
```

OrderManagement
order_service_impl.py
dao > order_service_impl.py > OrderServiceImpl > add_customer
EXPLORER
OPEN EDITORS
ORDERMANAGEMENT
.venv
dao
order_service_impl.py
order_service_interface.py
entity
util
db_config.properties
OUTLINE
TIMELINE order_service_impl.py
File Saved 18 mins
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 19, Col 37 Spaces: 4 UTF-8 CRLF {} Python 3.11.5 64-bit

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.

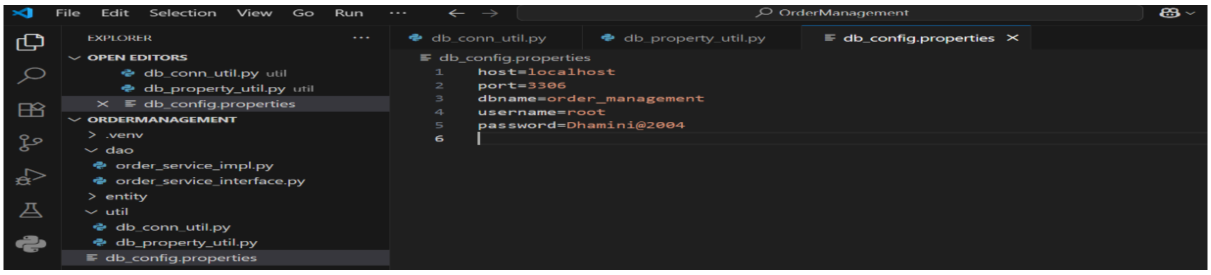
Step 5 – Database Connection Utilities

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.

Dhamini Reddy
PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)

1. db_config.properties

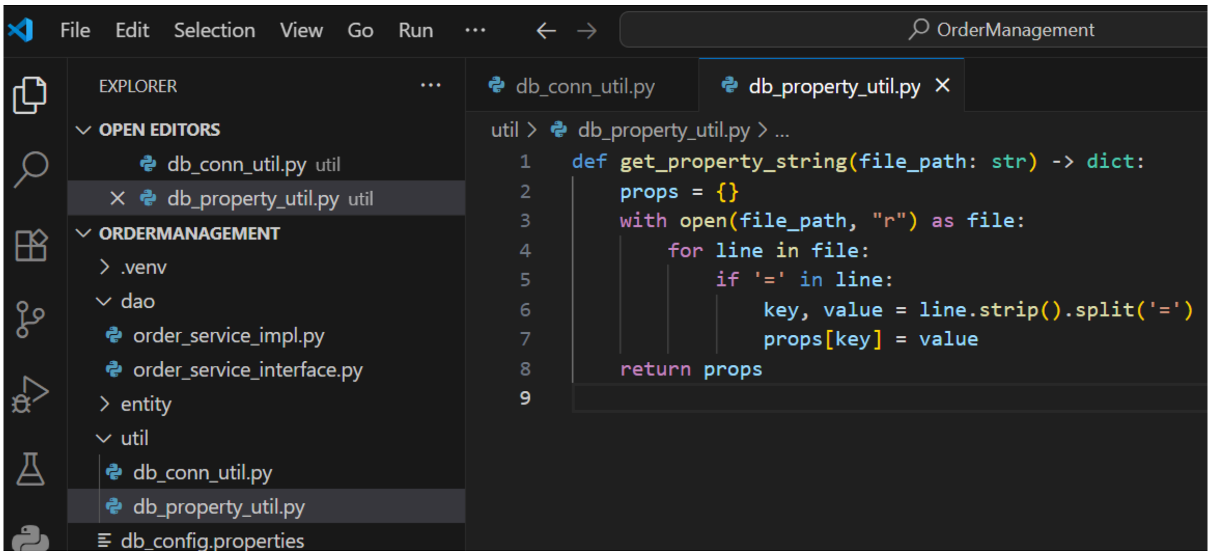
```
host=localhost
port=3306
dbname=order_management
username=root
password=password
```



```
host=localhost
port=3306
dbname=order_management
username=root
password=Dhamini@2004
```

2. 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
```

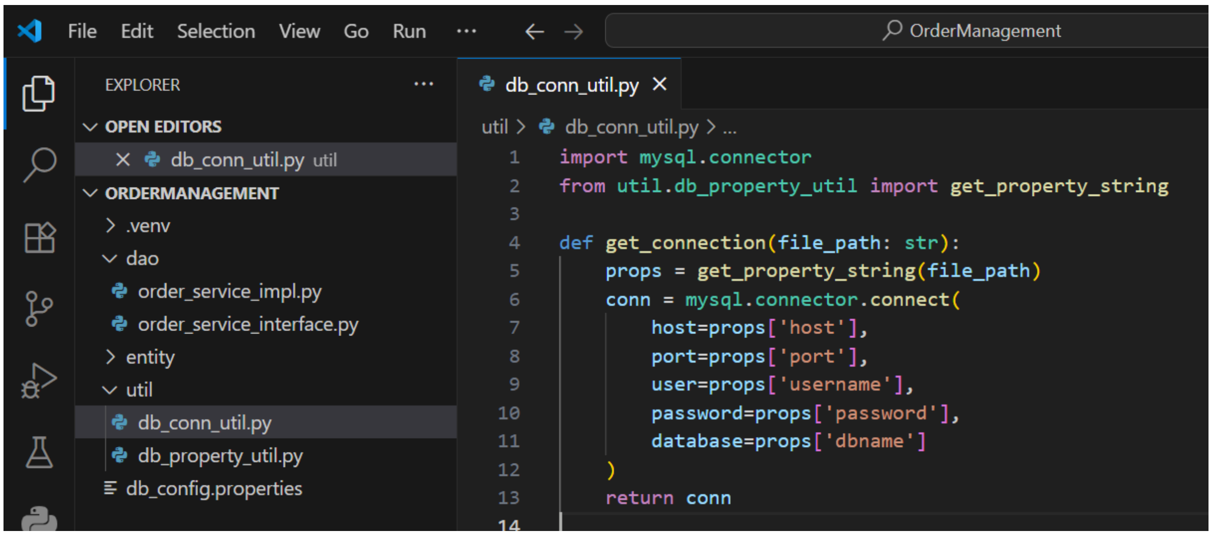


```
def get_property_string(file_path: str) -> dict:
    props = {}
    with open(file_path, "r") as file:
        for line in file:
            if '=' in line:
                key, value = line.strip().split('=')
                props[key] = value
    return props
```

3. 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
```



```
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)

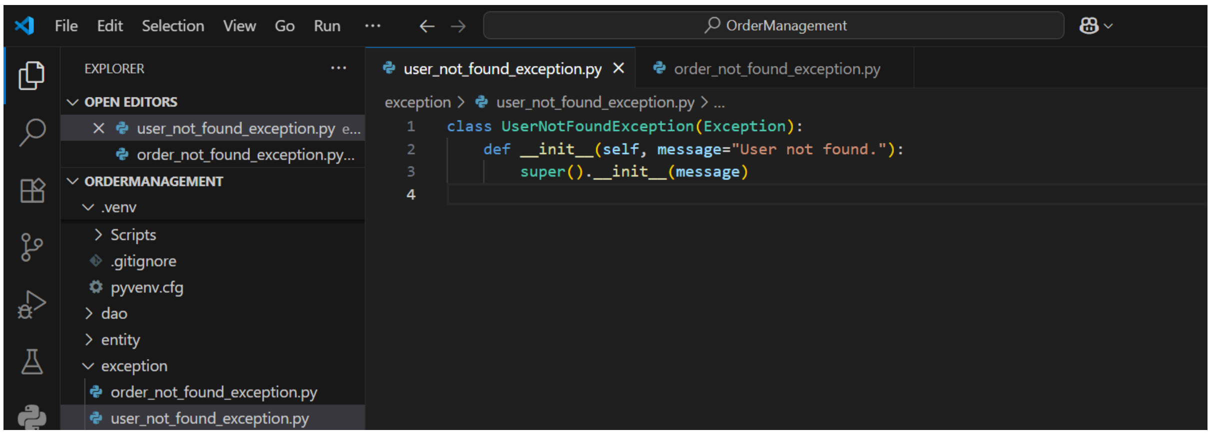
To follow good object-oriented practices, custom exceptions were created for user and order not found scenarios. These make the application easier to debug and maintain. File:

1. exception/user_not_found_exception.py

```
class UserNotFoundException(Exception):

 def __init__(self, message="User not found."):

     super().__init__(message)
```

2. File: exception/order_not_found_exception.py

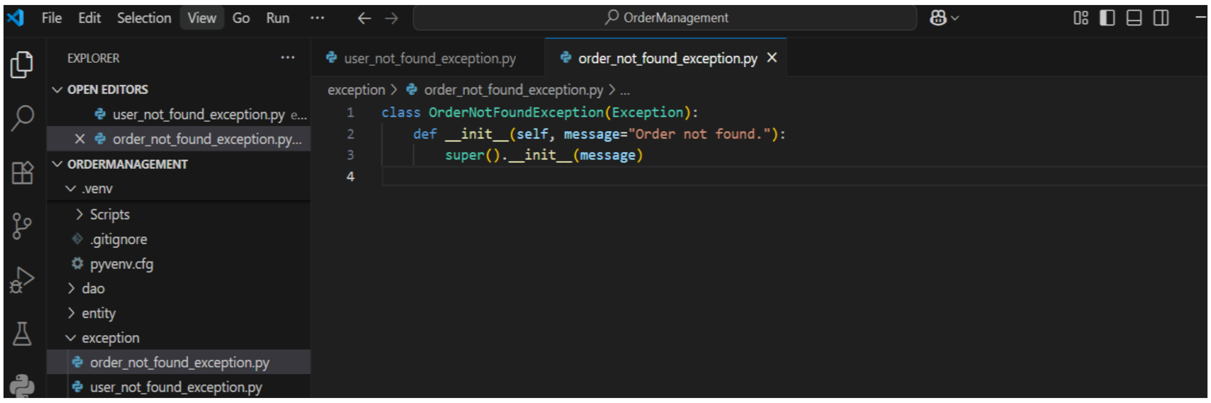
```
class OrderNotFoundException(Exception):
    def __init__(self, message="Order not found."):
        super().__init__(message)
```



exception > user_not_found_exception.py > ...

```
class UserNotFoundException(Exception):
    def __init__(self, message="User not found."):
        super().__init__(message)
```

File Edit Selection View Go Run ...
OrderManagement
EXPLORER
OPEN EDITORS
user_not_found_exception.py e...
order_not_found_exception.py...
ORDERMANAGEMENT
.venv
Scripts
.gitignore
pyvenv.cfg
dao
entity
exception
order_not_found_exception.py
user_not_found_exception.py



exception > order_not_found_exception.py > ...

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

File Edit Selection View Go Run ...
OrderManagement
EXPLORER
OPEN EDITORS
user_not_found_exception.py e...
order_not_found_exception.py...
ORDERMANAGEMENT
.venv
Scripts
.gitignore
pyvenv.cfg
dao
entity
exception
order_not_found_exception.py
user_not_found_exception.py

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

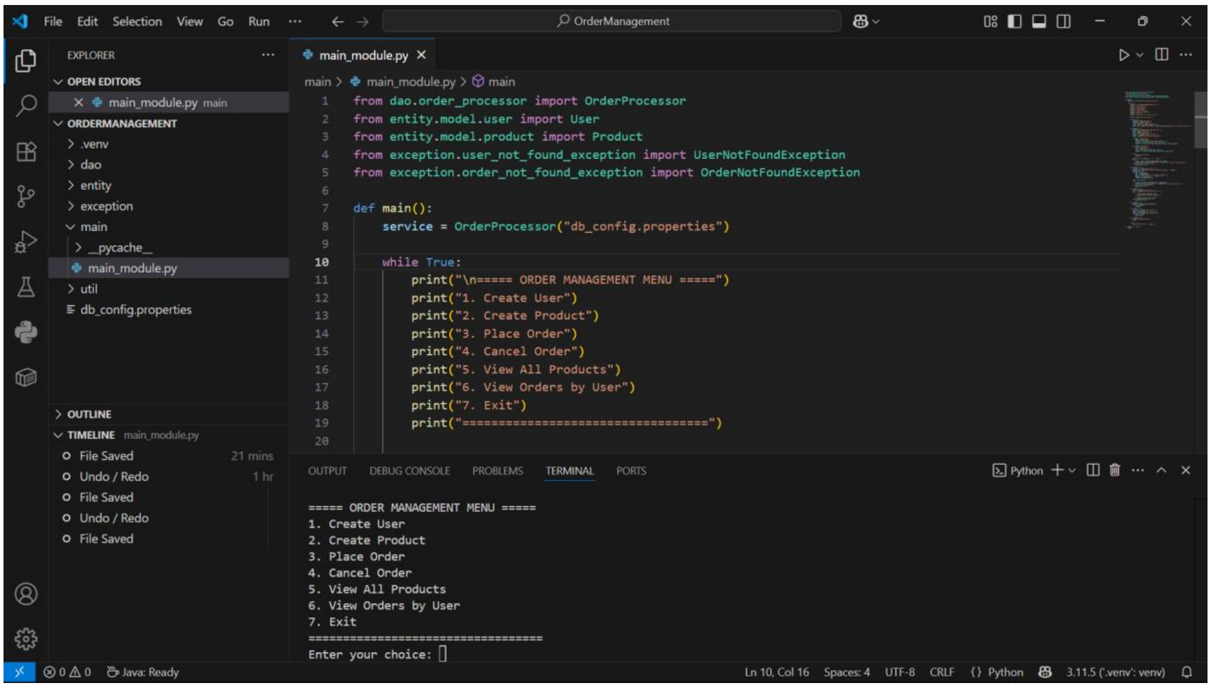
```
from dao.order_processor import OrderProcessor
from entity.model.user import User

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()
```

Dhamini Reddy

PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)

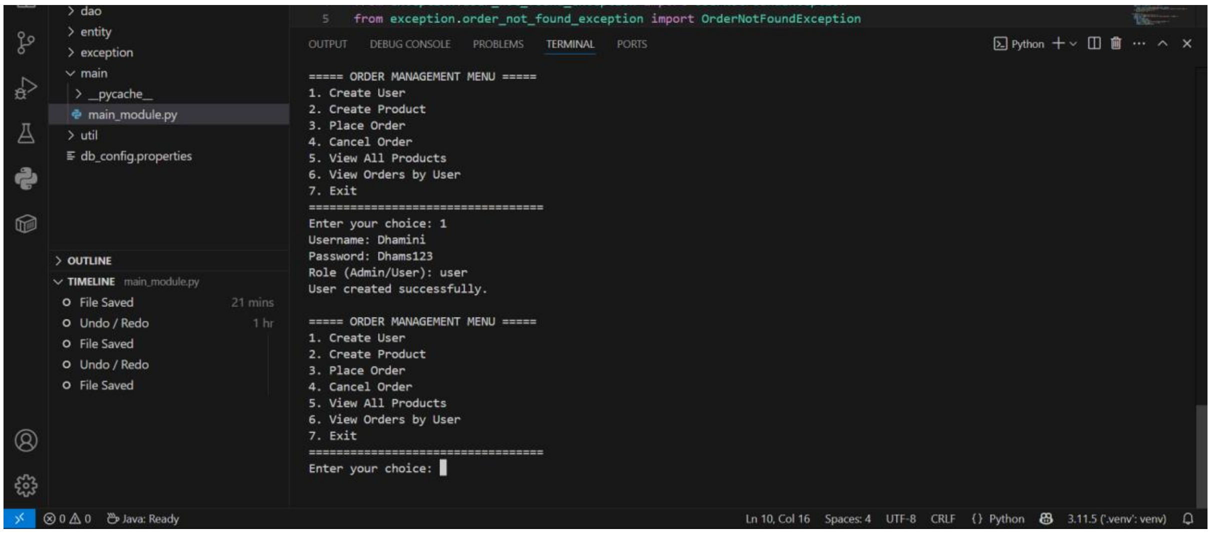


```python
from dao.order_processor import OrderProcessor
from entity.model.user import User
from entity.model.product import Product
from exception.user_not_found_exception import UserNotFoundException
from exception.order_not_found_exception import OrderNotFoundException

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("=================================")
```

```
===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice:
```

Output :

1. User creation

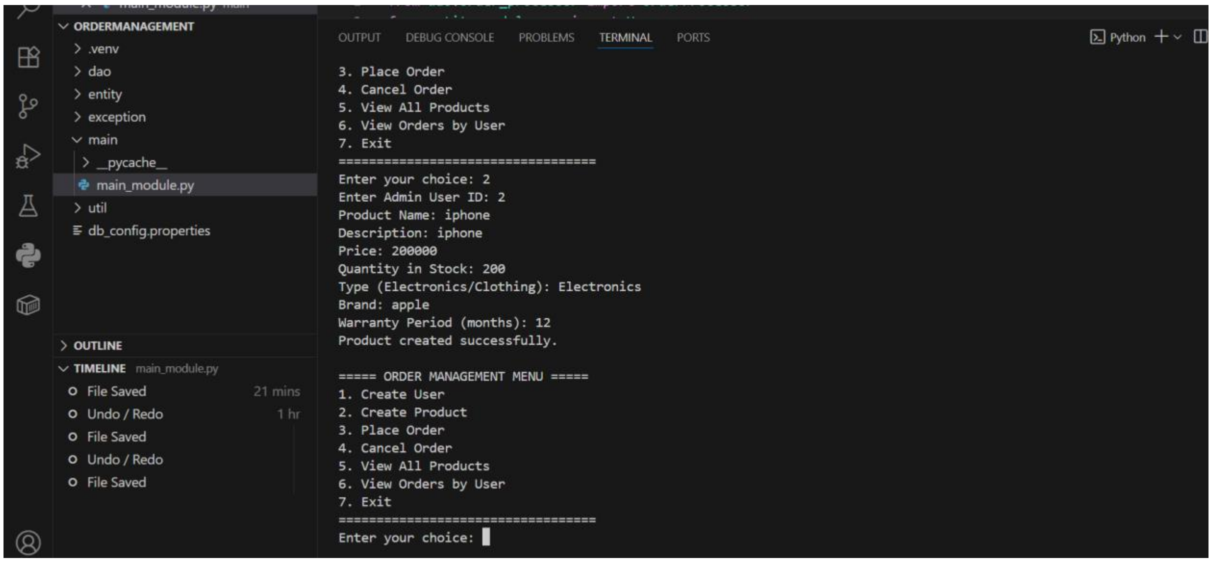


```
===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 1
Username: Dhamini
Password: Dhams123
Role (Admin/User): user
User created successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice:
```

2. Product creation

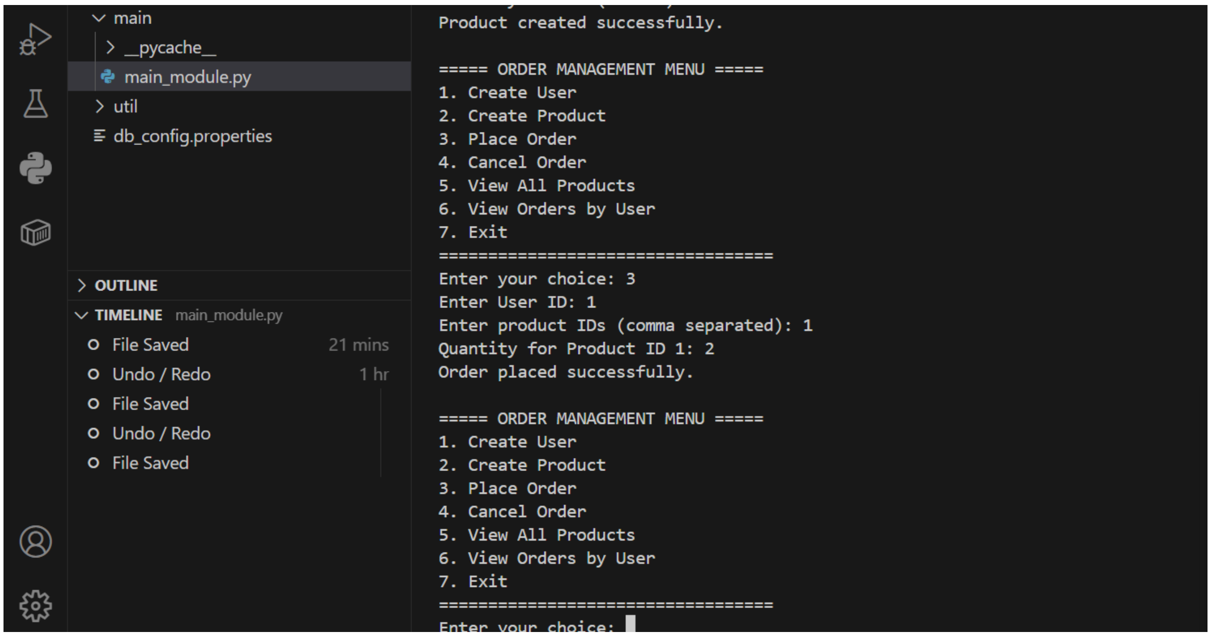
Dhamini Reddy

PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)



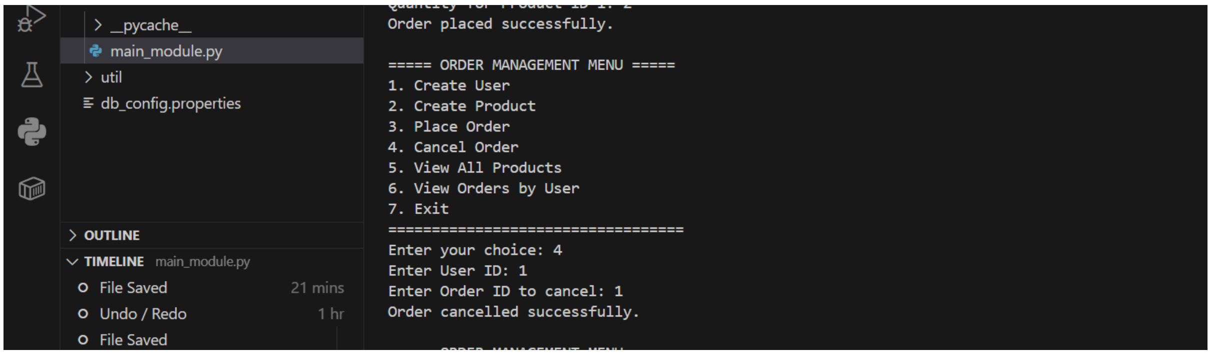
```
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 2
Enter Admin User ID: 2
Product Name: iphone
Description: iphone
Price: 200000
Quantity in Stock: 200
Type (Electronics/Clothing): Electronics
Brand: apple
Warranty Period (months): 12
Product created successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice:
```

3. Place order



```
Product created successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 3
Enter User ID: 1
Enter product IDs (comma separated): 1
Quantity for Product ID 1: 2
Order placed successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice:
```

4. cancel order

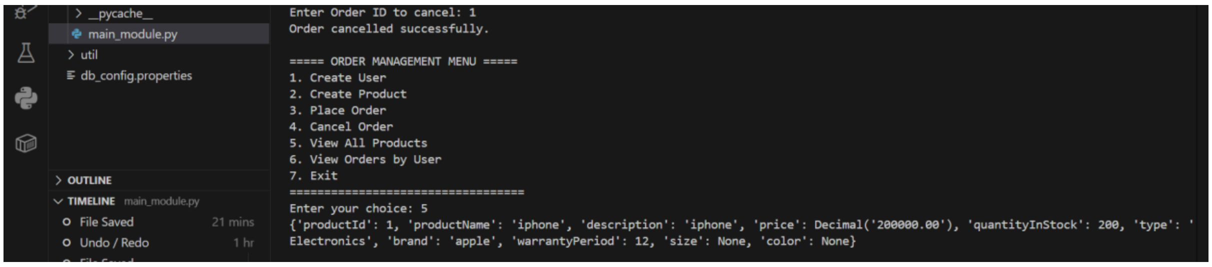


```
Order placed successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 4
Enter User ID: 1
Enter Order ID to cancel: 1
Order cancelled successfully.
```

Dhamini Reddy

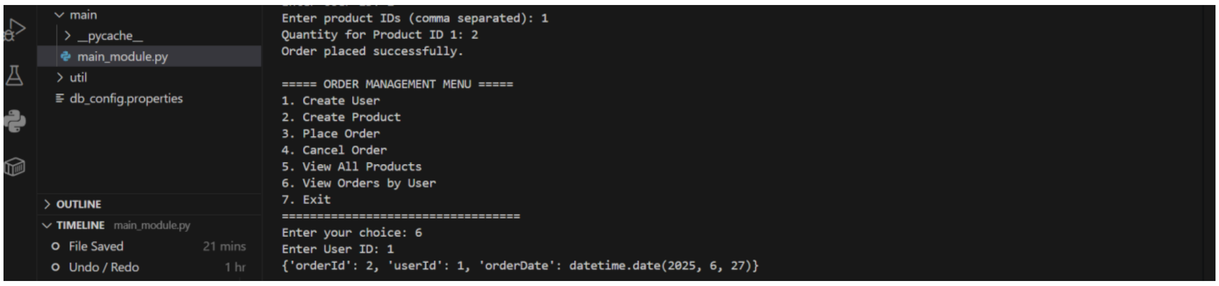
PYTHON CODING CHALLENGE _(ORDER MANAGEMENT)

5. View all products



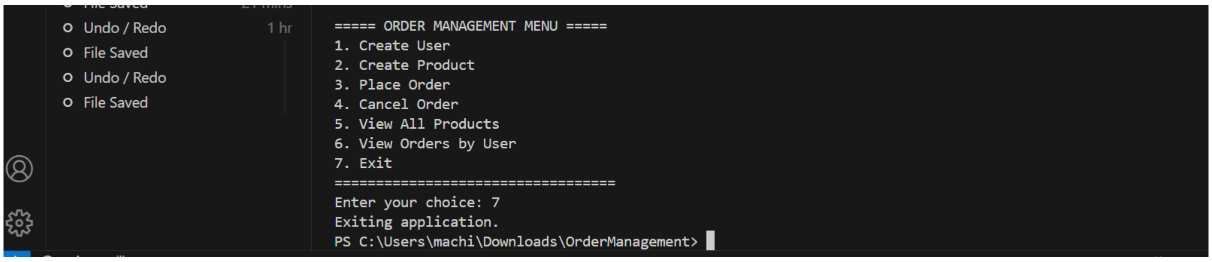
```
Enter Order ID to cancel: 1
Order cancelled successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 5
{'productId': 1, 'productName': 'iphone', 'description': 'iphone', 'price': Decimal('200000.00'), 'quantityInStock': 200, 'type': '
Electronics', 'brand': 'apple', 'warrantyPeriod': 12, 'size': None, 'color': None}
```

6. View order by user



```
Enter product IDs (comma separated): 1
Quantity for Product ID 1: 2
Order placed successfully.

===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 6
Enter User ID: 1
{'orderId': 2, 'userId': 1, 'orderDate': datetime.date(2025, 6, 27)}
```

7. Exit



```
===== ORDER MANAGEMENT MENU =====
1. Create User
2. Create Product
3. Place Order
4. Cancel Order
5. View All Products
6. View Orders by User
7. Exit
=================================
Enter your choice: 7
Exiting application.
PS C:\Users\machi\Downloads\OrderManagement>
```