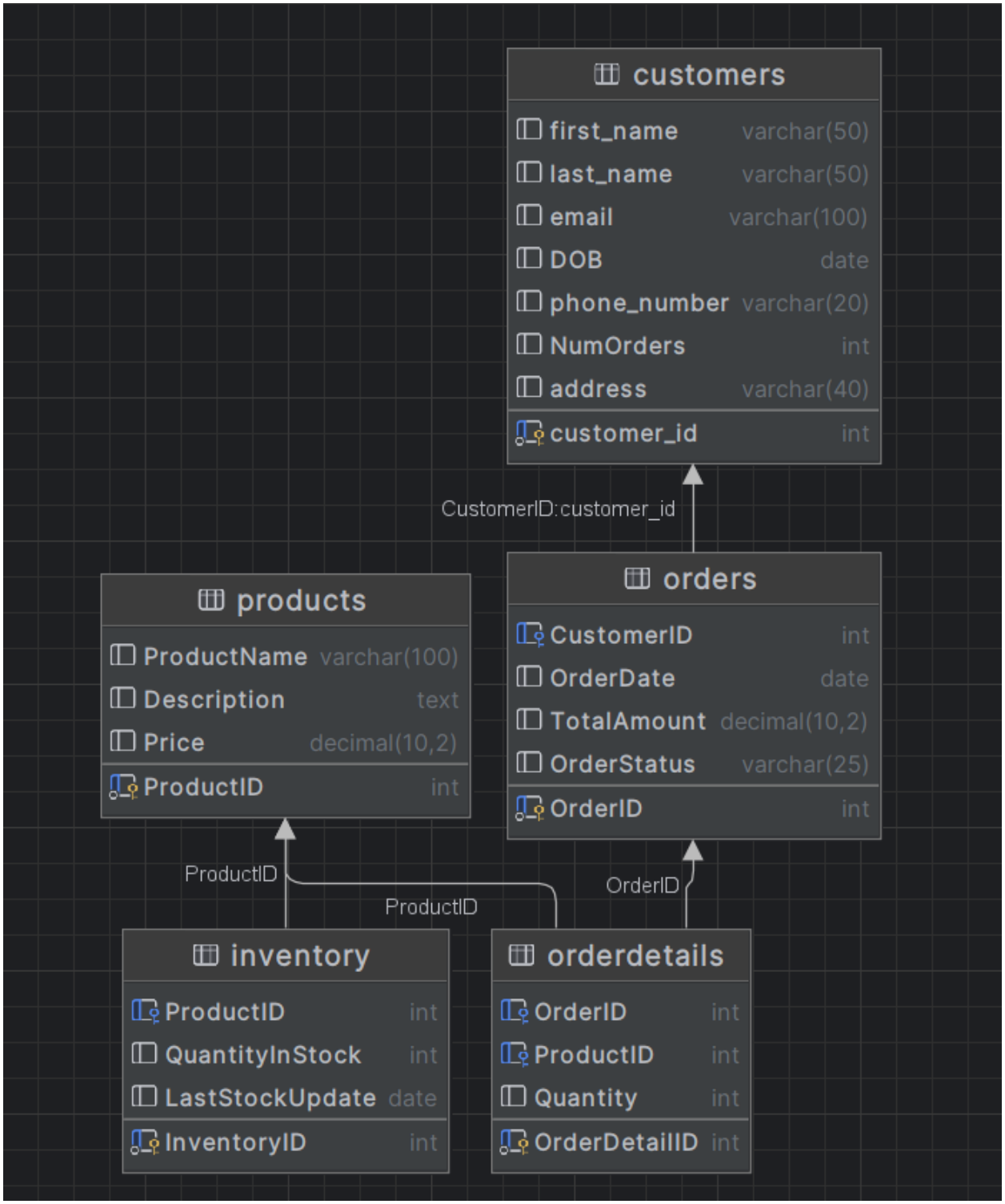
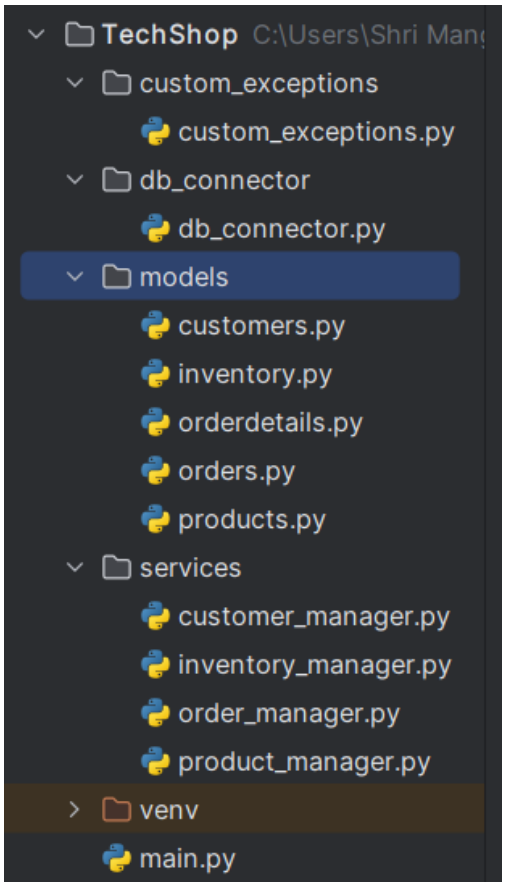
Tech Shop

Database Schema



File Structure



custom_exceptions.py

```python
3 usages  prakharmishra-cyber
class CustomerNotFoundException(Exception):
    pass

3 usages  prakharmishra-cyber
class ProductNotFoundException(Exception):
    pass

3 usages  prakharmishra-cyber
class OrderNotFoundException(Exception):
    pass
```

```python
import mysql.connector

21 usages  prakharmishra-cyber
def get_db_connection():
    # Replace the following values with your MySQL server credentials
    config = {
        'user': 'root',
        'password': 'prakhar123',
        'host': 'localhost',
        'database': 'techshop'
    }

    try:
        connection = mysql.connector.connect(**config)
        # print("Connected to the database")
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None
```

Customer.py

```
from db_connector.db_connector import get_db_connection
class Customers:
   def __init__(self, customer_id, first_name, last_name, email, dob, phone,
num_orders, address):
       self.connection = get_db_connection()
       self.__customer_id = customer_id
       self.__first_name = first_name
       self.__last_name = last_name
       self.__email = email
       self.__dob = dob
       self.__phone = phone
       self.__num_orders = num_orders
       self.__address = address

   def calculate_total_orders(self):
       cursor = self.connection.cursor()
       sql = 'select %s, Count(*) AS No_Of_Orders from orders where CustomerID
= %s;'
       para = (self.__customer_id, self.__customer_id)
       cursor.execute(sql, para)
       temp = list(cursor.fetchone())
       return temp[1]

   def get_customer_details(self):
       details = f"Customer ID: {self.__customer_id}\n"
       details += f"Name: {self.__first_name} {self.__last_name}\n"
       details += f"DOB: {self.__dob}\n"
       details += f"Email: {self.__email}\n"
       details += f"Phone: {self.__phone}\n"
       details += f"Address: {self.__address}\n"
       return details

   def update_customer_info(self, email=None, phone=None, address=None):
       if email:
           cursor = self.connection.cursor()
           sql = 'UPDATE Customers SET email = %s WHERE customer_id = %s'
           para = (email, self.__customer_id)
           cursor.execute(sql, para)
           self.__email = email
       if phone:
           cursor = self.connection.cursor()
           sql = 'UPDATE Customers SET phone = %s WHERE customer_id = %s'
           para = (phone, self.__customer_id)
           cursor.execute(sql, para)
           self.__phone = phone
       if address:
```

```
            cursor = self.connection.cursor()
            sql = 'UPDATE Customers SET address = %s WHERE customer_id = %s'
            para = (address, self.__customer_id)
            cursor.execute(sql, para)
            self.__address = address

    def get_num_orders(self):
        return self.__num_orders

    # Getter for customer_id
    def get_customer_id(self):
        return self.__customer_id

    # Getter for first_name
    def get_first_name(self):
        return self.__first_name

    # Getter for last_name
    def get_last_name(self):
        return self.__last_name

    # Getter for email
    def get_email(self):
        return self.__email

    # Getter for phone
    def get_phone(self):
        return self.__phone

    # Getter for address
    def get_address(self):
        return self.__address

    def get_dob(self):
        return self.__dob
```

Inventory.py

```
from models.products import Products
from db_connector.db_connector import get_db_connection
class Inventory:
    def __init__(self, inventory_id, product: Products, quantity_in_stock,
last_stock_update):
        self.connection = get_db_connection()
        self.__inventory_id = inventory_id
```

```
        self.__product = product
        self.__quantity_in_stock = quantity_in_stock
        self.__last_stock_update = last_stock_update

    # Getter for inventory_id
    def get_inventory_id(self):
        return self.__inventory_id

    # Getter for product
    def get_product(self):
        return self.__product

    # Setter for product
    def set_product(self, product:Products):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET ProductID = %s WHERE inventoryID = %s'
        para = (product.get_product_id(), self.__inventory_id)
        cursor.execute(sql, para)
        self.__product = product

    # Getter for quantity_in_stock
    def get_quantity_in_stock(self):
        return self.__quantity_in_stock

    # Setter for quantity_in_stock
    def set_quantity_in_stock(self, quantity_in_stock):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET QuantityInStock = %s WHERE inventoryID = %s'
        para = (quantity_in_stock, self.__inventory_id)
        cursor.execute(sql, para)
        self.__quantity_in_stock = quantity_in_stock

    # Getter for last_stock_update
    def get_last_stock_update(self):
        return self.__last_stock_update

    # Setter for last_stock_update
    def set_last_stock_update(self, last_stock_update):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET LastStockUpdate = %s WHERE inventoryID = %s'
        para = (last_stock_update, self.__inventory_id)
        cursor.execute(sql, para)
        self.__last_stock_update = last_stock_update

    def get_product(self):
        return self.__product

    def get_quantity_in_stock(self):
        return self.__quantity_in_stock
```

```
    def add_to_inventory(self, quantity):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET QuantityInStock = %s WHERE inventoryID = %s'
        para = (self.__quantity_in_stock + quantity, self.__inventory_id)
        cursor.execute(sql, para)
        self.__quantity_in_stock += quantity

    def remove_from_inventory(self, quantity):
        if self.__quantity_in_stock >= quantity:
            cursor = self.connection.cursor()
            sql = 'UPDATE inventory SET QuantityInStock = %s WHERE inventoryID =
%s'
            para = (self.__quantity_in_stock - quantity, self.__inventory_id)
            cursor.execute(sql, para)
            self.__quantity_in_stock -= quantity
        else:
            print("Error: Insufficient quantity in stock.")
            return

    def update_stock_quantity(self, new_quantity):
        cursor = self.connection.cursor()
        sql = 'UPDATE inventory SET QuantityInStock = %s WHERE inventoryID = %s'
        para = (new_quantity, self.__inventory_id)
        cursor.execute(sql, para)
        self.__quantity_in_stock = new_quantity

    def is_product_available(self, quantity_to_check):
        return self.__quantity_in_stock >= quantity_to_check

    def get_inventory_value(self):
        return self.__quantity_in_stock * self.__product.get_price()

    def list_low_stock_products(self, threshold):
        if self.__quantity_in_stock < threshold:
            return f"{self.__product.get_product_name()} is below the threshold
with {self.__quantity_in_stock} units."

    def list_out_of_stock_products(self):
        if self.__quantity_in_stock == 0:
            return f"{self.__product.get_product_name()} is out of stock."
        else:
            return f"No product is out of stock"

    def list_all_products(self):
        return f"Product: {self.__product.get_product_details()}, Quantity in
Stock: {self.__quantity_in_stock}"
```

Orderdetails.py

```
from datetime import datetime
from db_connector.db_connector import get_db_connection
from models.customers import Customers
from models.orders import Orders
from models.products import Products

class OrderDetails:
   def __init__(self, order_detail_id, order: Orders, product: Products,
quantity):
       self.connection = get_db_connection()
       self.__order_detail_id = order_detail_id
       self.__order = order
       self.__product = product
       self.__quantity = quantity

   # Getter for order_detail_id
   def get_order_detail_id(self):
       return self.__order_detail_id

   # Getter for order
   def get_order(self):
       return self.__order

   # Getter for product
   def get_product(self):
       return self.__product

   # Setter for product
   def set_product(self, product: Products):
       cursor = self.connection.cursor()
       sql = 'UPDATE orderdetails SET ProductID = %s WHERE OrderDetailID = %s'
       para = (product.get_product_id(), self.__order_detail_id)
       cursor.execute(sql, para)
       self.__product = product

   # Getter for quantity
   def get_quantity(self):
       return self.__quantity

   # Setter for quantity
   def set_quantity(self, quantity):
       if quantity < 0:
           print('Enter a valid quantity')
           return
       cursor = self.connection.cursor()
       sql = 'UPDATE orderdetails SET Quantity = %s WHERE OrderDetailID = %s'
       para = (quantity, self.__order_detail_id)
```

```
        cursor.execute(sql, para)
        self.__quantity = quantity

    def calculate_subtotal(self):
        return self.__product.get_price() * self.__quantity

    def get_order_detail_info(self):
        details = f"Order Detail ID: {self.__order_detail_id}\n"
        details += f"Order: {self.__order.get_order_details()}\n"
        details += f"Product: {self.__product.get_product_details()}\n"
        details += f"Quantity: {self.__quantity}\n"
        details += f"Subtotal: ${self.calculate_subtotal():.2f}\n"
        return details

    def update_quantity(self, new_quantity):
        cursor = self.connection.cursor()
        sql = 'UPDATE orderdetails SET Quantity = %s WHERE OrderDetailID = %s'
        para = (new_quantity, self.__order_detail_id)
        cursor.execute(sql, para)
        self.__quantity = new_quantity

    def add_discount(self, discount_percentage):
        discount_factor = 1 - (discount_percentage / 100)
        subtotal = self.calculate_subtotal()
        discounted_subtotal = subtotal * discount_factor
        print(f'Discounted Subtotal {discounted_subtotal}')
```

Order.py

```
from datetime import datetime
from models.customers import Customers
from db_connector.db_connector import get_db_connection

class Orders:
    def __init__(self, order_id, customer: Customers, order_date, total_amount,
order_status="Pending"):
        self.connection = get_db_connection()
        self.__order_id = order_id
        self.__customer = customer
        self.__order_date = order_date
        self.__total_amount = total_amount
        self.__order_status = order_status

    # Getter for order_id
    def get_order_id(self):
        return self.__order_id
```

```
    # Getter for customer
    def get_customer(self):
        return self.__customer

    # Getter for order_date
    def get_order_date(self):
        return self.__order_date

    # Setter for order_date
    def set_order_date(self, order_date):
        cursor = self.connection.cursor()
        sql = 'UPDATE Orders SET OrderDate = %s WHERE OrderID = %s'
        para = (order_date, self.__order_id)
        cursor.execute(sql, para)
        self.__order_date = order_date

    # Getter for total_amount
    def get_total_amount(self):
        return self.__total_amount

    # Setter for total_amount
    def set_total_amount(self, total_amount):
        cursor = self.connection.cursor()
        sql = 'UPDATE Orders SET TotalAmount = %s WHERE OrderID = %s'
        para = (total_amount, self.__order_id)
        cursor.execute(sql, para)
        self.__total_amount = total_amount

    # Getter for order_status
    def get_order_status(self):
        return self.__order_status

    # Setter for order_status
    def set_order_status(self, order_status):
        cursor = self.connection.cursor()
        sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
        para = (order_status, self.__order_id)
        cursor.execute(sql, para)
        self.__order_status = order_status

    def calculate_total_amount(self):
        cursor = self.connection.cursor()
        sql = 'SELECT SUM(orderdetails.Quantity*orders.TotalAmount) AS
TotalAmount FROM orderdetails JOIN orders on orderdetails.OrderID =
orders.OrderID WHERE orders.OrderID = %s'
        para = (self.__order_id,)
        cursor.execute(sql, para)
        totalamount = list(cursor.fetchone())[0]
        return totalamount
```

```
    def get_order_details(self):
        details = f"Order ID: {self.__order_id}\n"
        details += f"Customer: {self.__customer.get_customer_details()}\n"
        details += f"Order Date: {self.__order_date}\n"
        details += f"Total Amount: ${self.calculate_total_amount():.2f}\n"
        details += f"Order Status: {self.__order_status}\n"
        return details

    def update_order_status(self, new_status):
        cursor = self.connection.cursor()
        sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
        para = (new_status, self.__order_id)
        cursor.execute(sql, para)
        self.__order_status = new_status

    def cancel_order(self):
        cursor = self.connection.cursor()
        sql = 'UPDATE Orders SET OrderStatus= %s WHERE OrderID = %s'
        para = ('Cancelled', self.__order_id)
        cursor.execute(sql, para)
        self.__order_status = "Cancelled"
        print('Order successfully cancelled')
```

Products.py

```
from db_connector.db_connector import get_db_connection

class Products:
    def __init__(self, product_id, product_name, description, price):
        self.connection = get_db_connection()
        self.__product_id = product_id
        self.__product_name = product_name
        self.__description = description
        self.__price = price

    def get_product_id(self):
        return self.__product_id

    def set_product_id(self, product_id):
        sql = 'UPDATE Products SET productID = %s WHERE productID = %s'
        para = (product_id, self.__product_id)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.__product_id = product_id

    def get_product_name(self):
        return self.__product_name
```

```
    def set_product_name(self, product_name):
        sql = 'UPDATE Products SET productName = %s WHERE productID = %s'
        para = (product_name, self.__product_id)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.__product_name = product_name

    def get_description(self):
        return self.__description

    def set_description(self, description):
        sql = 'UPDATE Products SET description = %s WHERE productID = %s'
        para = (description, self.__product_id)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.__description = description

    def get_price(self):
        return self.__price

    def set_price(self, price):
        sql = 'UPDATE Products SET price = %s WHERE productID = %s'
        para = (price, self.__product_id)
        cursor = self.connection.cursor()
        cursor.execute(sql, para)
        self.__price = price

    def get_product_details(self):
        details = f"Product ID: {self.__product_id}\n"
        details += f"Product Name: {self.__product_name}\n"
        details += f"Description: {self.__description}\n"
        details += f"Price: ${self.__price:.2f}\n"
        return details

    def update_product_info(self, price=None, description=None):
        if price:
            cursor = self.connection.cursor()
            sql = 'UPDATE Products SET price = %s WHERE productID = %s'
            para = (price, self.__product_id)
            cursor.execute(sql, para)
            self.__price = price
        if description:
            cursor = self.connection.cursor()
            sql = 'UPDATE Products SET description = %s WHERE productID = %s'
            para = (description, self.__product_id)
            cursor.execute(sql, para)
            self.__description = description

    def is_product_in_stock(self):
```

```
        cursor = self.connection.cursor()
        sql = 'SELECT inventory.QuantityInStock FROM products JOIN inventory ON
products.ProductID = inventory.ProductID WHERE products.ProductID = %s'
        para = (self.__product_id,)
        cursor.execute(sql, para)
        x = list(cursor.fetchone())[0]
        return x > 0
```

Customer_manager.py

```
import re

from custom_exceptions.custom_exceptions import CustomerNotFoundException
from models.customers import Customers
from db_connector.db_connector import get_db_connection
class CustomerManager:
    def __init__(self):
        self.connection = get_db_connection()

    def register_customer(self, first_name, last_name, email, dob, phone,
num_orders, address):
        try:
            # Validating input data
            self.validate_customer_data(email)
            # Checking if the email already exists in the database
            if self.is_email_duplicate(email):
                raise ValueError("Email address is already registered.")

            cursor = self.connection.cursor()
            sql = 'SELECT * FROM Customers'
            cursor.execute(sql)
            x = list(cursor.fetchall())
            sql2 = 'INSERT INTO Customers(customer_id, first_name, last_name,
email, DOB, phone_number, NumOrders, address) VALUES(%s, %s, %s, %s, %s, %s,
%s, %s)'
            para = (len(x)+1, first_name, last_name, email, dob, phone,
num_orders, address)
            cursor.execute(sql2, para)
            self.connection.commit()
            self.connection.close()
            print("Customer registration successful.")
        except Exception as e:
            print(f"Error registering customer: {e}")

    def validate_customer_data(self, email):
        regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b'
        if not re.fullmatch(regex, email):
            raise ValueError("Invalid email address.")
```

```
    def is_email_duplicate(self, email):
        cursor = self.connection.cursor()
        sql = 'SELECT * FROM Customers WHERE email = %s'
        para = (email,)
        cursor.execute(sql, para)
        x = len(list(cursor.fetchall()))
        if (x > 0):
            return True
        return False

    def get_customer_by_id(self, customer_id):
        try:
            my_cursor = self.connection.cursor()
            sql = '''SELECT * FROM Customers WHERE customer_id = %s'''
            para = (customer_id,)
            my_cursor.execute(sql, para)
            x = my_cursor.fetchone()
            if x is None:
                raise CustomerNotFoundException('Invalid Customer ID')
            else:
                return Customers(*x)
        except CustomerNotFoundException as cnfe:
            print('An error occurred ',cnfe)
        except Exception as e:
            print('An error occurred ',e)


# Example usage:
# customer_manager = CustomerManager()
# customer_manager.register_customer("Prakhar", "Mishra",
"prakhar.mishra@example.com", '1990-05-01', "1234567890", 0,"Kanpur, UP,
India")
```

Inventory_manager.py

```
from models.inventory import Inventory
from models.products import Products
from services.product_manager import ProductManager
from db_connector.db_connector import get_db_connection

class InventoryManager:
    def __init__(self):
        self.connection = get_db_connection()

    def add_to_inventory(self, product_id, quantity):
        try:
            mycursor = self.connection.cursor()
            p1 = ProductManager()
```

```
            if p1.product_exists(product_id):
                sql = 'UPDATE Inventory SET QuantityInStock = QunatityInStock +
%s WHERE ProductID = %s'
                para = (quantity,)
                mycursor.execute(sql, para)
                self.connection.commit()
            else:
                print(f'Invalid ProductID')
                return

            print(f"Inventory updated successfully. Product ID: {product_id},
Quantity: {quantity}")
        except Exception as e:
            print(f"Error updating inventory: {e}")

    def remove_from_inventory(self, product_id, quantity):
        try:
            mycursor = self.connection.cursor()
            p1 = ProductManager()
            if p1.product_exists(product_id):
                sql = 'UPDATE Inventory SET QuantityInStock = QunatityInStock -
%s WHERE ProductID = %s'
                para = (quantity,)
                mycursor.execute(sql, para)
                self.connection.commit()
            else:
                print(f"Error updating inventory: Product ID {product_id} not
found")
        except Exception as e:
            print(f"Error updating inventory: {e}")

    def get_inventory_value(self):
        mycursor = self.connection.cursor()
        sql = '''SELECT SUM(Inventory.QuantityInStock*Products.Price)
                 AS TotalInventoryValue FROM Inventory
                 JOIN Products ON Inventory.ProductID = Products.ProductID'''
        mycursor.execute(sql)
        x = list(mycursor.fetchone())[0]
        return x

    def list_low_stock_products(self, threshold):
        mycursor = self.connection.cursor()
        sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory
                                JOIN Products ON Inventory.ProductID =
Products.ProductID
                                WHERE Inventory.QuantityInStock < %s'''
        para = (threshold,)
        mycursor.execute(sql, para)
        print("All Low Stock Products in Inventory:")
```

```
        print("ProductID | ProductName | Description | Price | QuantityInStock
\n")
        for item in mycursor.fetchall():
            print(item)

    def list_out_of_stock_products(self):
        mycursor = self.connection.cursor()
        sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory
                        JOIN Products ON Inventory.ProductID =
Products.ProductID
                        WHERE Inventory.QuantityInStock=0'''
        mycursor.execute(sql)
        print("All Out of Stock Products in Inventory:")
        print("ProductID | ProductName | Description | Price | QuantityInStock
\n")
        for item in mycursor.fetchall():
            print(item)

    def list_all_products(self):
        mycursor = self.connection.cursor()
        sql = '''SELECT Products.*, Inventory.QuantityInStock FROM Inventory
                JOIN Products ON Inventory.ProductID = Products.ProductID'''
        mycursor.execute(sql)
        print("All Products in Inventory:")
        print("ProductID | ProductName | Description | Price | QuantityInStock
\n")
        for item in mycursor.fetchall():
            print(item)

    def get_product_by_id(self, product_id) -> Products:
        mycursor = self.connection.cursor()
        sql = 'SELECT * FROM Products WHERE ProductID = %s'
        para = (product_id,)
        mycursor.execute(sql, para)
        x = list(mycursor.fetchone())
        return Products(x[0], x[1], x[2], x[3])
```

Order_manager.py

```
from datetime import datetime

from custom_exceptions.custom_exceptions import OrderNotFoundException
from models.orders import Orders
from models.products import Products
# from services.inventory_manager import InventoryManager
from db_connector.db_connector import get_db_connection

def getAllOrders() -> list:
```

```
    mydb = get_db_connection()
    mycursor = mydb.cursor()
    sql = 'SELECT * FROM Orders'
    mycursor.execute(sql)
    t = list(mycursor.fetchall())
    x = []
    for i in t: x.append(list(i))
    return x

class OrderManager:
    def __init__(self):
        self.connection = get_db_connection()

    def add_order(self, order: Orders):
        try:
            mycursor = self.connection.cursor()
            sql = 'INSERT INTO Orders(OrderID, CustomerID, OrderDate,
TotalAmount, OrderStatus) VALUES (%s, %s, %s, %s, %s, %s)'
            para = (order.get_order_id(),
order.get_customer().get_customer_id(), order.get_order_date(),
order.get_total_amount(), order.get_order_status())
            mycursor.execute(sql, para)
            self.connection.commit()
            print(f"Order {order.get_order_id()} added successfully.")
        except Exception as e:
            print(f"Error adding order: {e}")

    def update_order_status(self, order_id, new_status):
        try:
            mycursor = self.connection.cursor()
            sql = 'UPDATE Order SET OrderStatus = %s WHERE OrderID = %s'
            para = (new_status, order_id)
            mycursor.execute(sql, para)
            self.connection.commit()
        except Exception as e:
            print(f"Error updating order status: {e}")

    def remove_canceled_orders(self):
        try:
            mycursor = self.connection.cursor()
            sql1 = '''DELETE FROM OrderDetails WHERE OrderID IN
                    (SELECT OrderID FROM Orders WHERE OrderStatus IN
('Cancelled', 'cancelled')'''
            sql2 = '''DELETE FROM Orders WHERE OrderStatus IN ('Cancelled',
'cancelled')'''
            mycursor.execute(sql1)
            mycursor.execute(sql2)
            print(f"Orders removed successfully")
        except Exception as e:
```

```
            print(f"Error removing canceled orders: {e}")

    def sort_orders_by_date(self, ascending=True):
        mydb = get_db_connection()
        mycursor = mydb.cursor()
        sql = 'SELECT * FROM Orders ORDER BY OrderDate ASC'
        mycursor.execute(sql)
        t = list(mycursor.fetchall())
        x = []
        for i in t:
            x.append(list(i))
        if not ascending:
            x.reverse()
        return x

    def list_all_orders(self):
        return getAllOrders()

    def get_order_by_id(self, order_id):
        try:
            my_cursor = self.connection.cursor()
            sql = '''SELECT * FROM Orders WHERE orderID = %s'''
            para = (order_id,)
            my_cursor.execute(sql, para)
            x = my_cursor.fetchone()
            if x is None:
                raise OrderNotFoundException('Invalid Order ID')
            else:
                return Orders(*x)
        except OrderNotFoundException as onfe:
            print('An error occurred ',onfe)
        except Exception as e:
            print('An error occurred ',e)
```

Product_manager.py

```
from custom_exceptions.custom_exceptions import ProductNotFoundException
from models.products import Products
from db_connector.db_connector import get_db_connection

def getAllProducts() -> list:
    mydb = get_db_connection()
    mycursor = mydb.cursor()
    mycursor.execute('SELECT * FROM Products')
    t = list(mycursor.fetchall())
    x = [Products(*list(i)) for i in t]
    return x
```

```
class ProductManager:
    def __init__(self):
        self.connection = get_db_connection()

    def add_product(self, product: Products):
        try:
            # Check if the product with the same ProductID already exists
            if self.product_exists(product.get_product_id()):
                raise ValueError("Product with the same ProductID already
exists.")

            cursor = self.connection.cursor()
            sql = '''INSERT INTO Products(ProductID, ProductName, Description,
Price) VALUES(%s, %s, %s, %s)'''
            para = (product.get_product_id(), product.get_product_name(),
product.get_description(), product.get_price())
            cursor.execute(sql, para)
            self.connection.commit()
            print(f"Product {product.get_product_id()} added successfully.")
        except Exception as e:
            print(f"Error adding product: {e}")

    def update_product(self, product: Products, new_price, new_description):
        try:
            # Check if the product with the given ProductID exists
            if not self.product_exists(product.get_product_id()):
                raise ValueError("Product with the given ProductID does not
exist.")

            product.update_product_info(price = new_price, description =
new_description)
            print(f"Product {product.get_product_id()} updated successfully.")
        except Exception as e:
            print(f"Error updating product: {e}")

    def remove_product(self, product_id):
        try:
            # Check if the product with the given ProductID exists
            if not self.product_exists(product_id):
                raise ValueError("Product with the given ProductID does not
exist.")

            # Check if the product is associated with any existing orders
            if self.product_has_orders(product_id):
                raise ValueError("Product cannot be removed as it is associated
with existing orders.")

            cursor = self.connection.cursor()
            sql = 'DELETE FROM Products WHERE ProductID = %s'
```

```
            para = (product_id,)
            cursor.execute(sql, para)
            self.connection.commit()
            print(f"Product {product_id} removed successfully.")
        except Exception as e:
            print(f"Error removing product: {e}")

    def product_exists(self, product_id):
        cursor = self.connection.cursor()
        sql = '''SELECT COUNT(*) FROM Products WHERE ProductID = %s'''
        para = (product_id,)
        cursor.execute(sql, para)
        x = len(list(cursor.fetchall()))
        return x > 0

    def product_has_orders(self, product_id):
        cursor = self.connection.cursor()
        sql = '''SELECT COUNT(OrderDetails.OrderID) AS Order_Number FROM
Products
                JOIN OrderDetails
                ON Products.ProductID = OrderDetails.ProductID
                WHERE products.ProductID = %s GROUP BY products.ProductID'''
        para = (product_id, )
        cursor.execute(sql, para)
        x = len(list(cursor.fetchall()))
        return x > 0

    def list_all_products(self) -> list[Products]:
        return getAllProducts()

    def search_product(self, search_keyword):
        mycursor = self.connection.cursor()
        sql = '''SELECT * FROM Products WHERE UPPER(ProductName) LIKE UPPER(%s)
OR UPPER(Description) LIKE UPPER(%s)'''
        para = ('%'+search_keyword+'%', '%'+search_keyword+'%')
        mycursor.execute(sql, para)
        t = list(mycursor.fetchall())
        x = [list(i) for i in t]
        return x

    def get_product_by_id(self, product_id):
        try:
            my_cursor = self.connection.cursor()
            sql = '''
            SELECT * Products WHERE ProductID = %s
            '''
            para = (product_id,)
            my_cursor.execute(sql, para)
            x = my_cursor.fetchone()
```

```
            if x is None:
                raise ProductNotFoundException('Invalid product ID')
            else:
                return Products(*x)
        except ProductNotFoundException as pnfe:
            print('An error occurred: ', pnfe)
        except Exception as e:
            print( "An error occurred: ", e )
```

Main.py

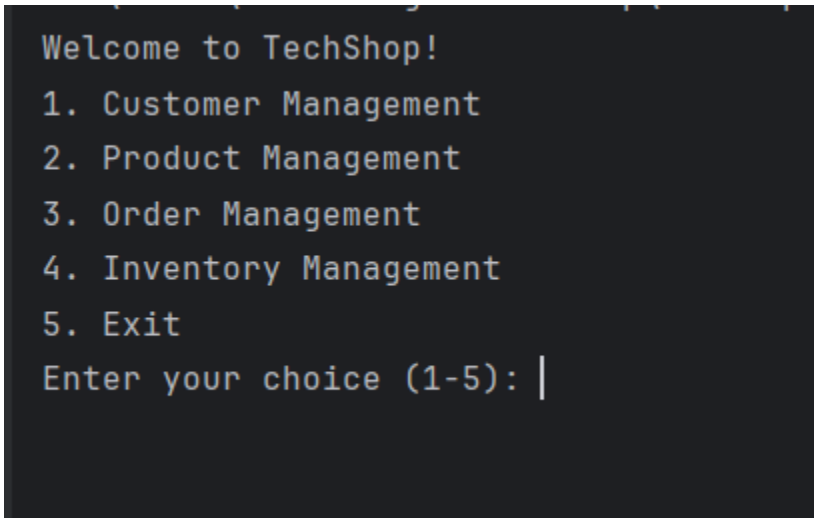
```
from models.orders import Orders
from models.products import Products
from services.customer_manager import CustomerManager
from services.product_manager import ProductManager
from services.order_manager import OrderManager
from services.inventory_manager import InventoryManager

def main_menu():
   print("Welcome to TechShop!")
   print("1. Customer Management")
   print("2. Product Management")
   print("3. Order Management")
   print("4. Inventory Management")
   print("5. Exit")

def customer_management_menu():
   print("\nCustomer Management Menu:")
   print("1. Register Customer")
   print("2. View Customer Details")
   print("3. Update Customer Information")
   print("4. Back to Main Menu")

def product_management_menu():
   print("\nProduct Management Menu:")
   print("1. Add Product")
   print("2. View Product Details")
   print("3. Update Product Information")
   print("4. Back to Main Menu")

def order_management_menu():
   print("\nOrder Management Menu:")
   print("1. Place Order")
   print("2. View Order Details")
   print("3. Update Order Status")
   print("4. Cancel Order")
   print("5. Back to Main Menu")

def inventory_management_menu():
```

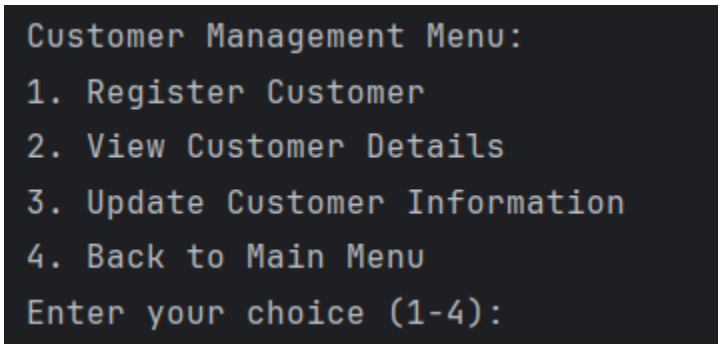
```
    print("\nInventory Management Menu:")
    print("1. Add to Inventory")
    print("2. Remove from Inventory")
    print("3. View Inventory Details")
    print("4. List Low Stock Products")
    print("5. List Out of Stock Products")
    print("6. List All Products in Inventory")
    print("7. Back to Main Menu")

if __name__ == "__main__":
    customer_manager = CustomerManager()
    product_manager = ProductManager()
    order_manager = OrderManager()
    inventory_manager = InventoryManager()

    while True:
        main_menu()
        choice = input("Enter your choice (1-5): ")

        if choice == "1":
            while True:
                customer_management_menu()
                customer_choice = input("Enter your choice (1-4): ")

                if customer_choice == "1":
                    customer_manager.register_customer(
                        input("Enter First Name: "),
                        input("Enter Last Name: "),
                        input("Enter Email: "),
                        input("Enter DOB: "),
                        input("Enter Phone: "),
                        0,
                        input("Enter Address: ")
                    )
                elif customer_choice == "2":
                    temp_customer =
customer_manager.get_customer_by_id(customer_id=input('Enter CustomerID: '))
                    print(temp_customer.get_customer_details())
                elif customer_choice == "3":
                    temp_customer =
customer_manager.get_customer_by_id(customer_id=input('Enter CustomerID: '))
                    temp_customer.update_customer_info(email=input('Enter new
email'),
                                                       phone=input('Enter new
phone'),
                                                       address=input('Enter new
address'))

                elif customer_choice == "4":
```

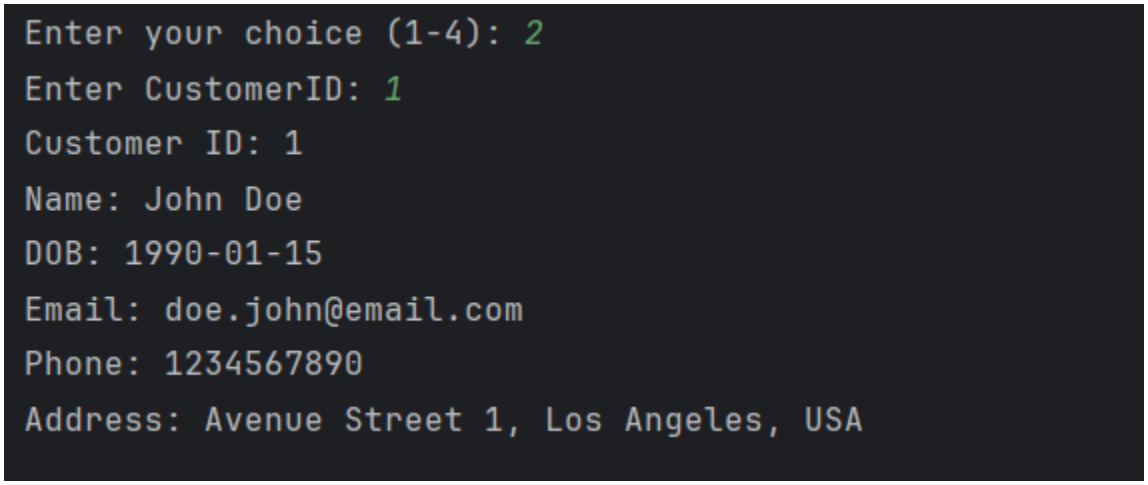
```
                break
            else:
                print("Invalid choice. Please enter a number between 1 and
4.")

        elif choice == "2":
            while True:
                product_management_menu()
                product_choice = input("Enter your choice (1-4): ")

                if product_choice == "1":
                    product_manager.add_product(
                        Products(input("Enter Product ID: "),
                        input("Enter Product Name: "),
                        input("Enter Description: "),
                        float(input("Enter Price: ")))
                    )
                elif product_choice == "2":
                    temp_product =
product_manager.get_product_by_id(input('Enter Product ID'))
                    print(temp_product.get_product_details())
                elif product_choice == "3":
                    temp_product =
product_manager.get_product_by_id(input('Enter Product ID'))
                    temp_product.update_product_info(price=input('Enter New
Product Price'), description=input('Enter New Product Description'))
                elif product_choice == "4":
                    break
                else:
                    print("Invalid choice. Please enter a number between 1 and
4.")

        elif choice == "3":
            while True:
                order_management_menu()
                order_choice = input("Enter your choice (1-5): ")
                cm = CustomerManager()

                if order_choice == "1":
                    order_manager.add_order(Orders(
                        input('Enter Order ID: '),
                        cm.get_customer_by_id(input('Enter Customer ID: ')),
                        input('Enter Order Date: '),
                        input('Enter Total Amount: '),
                        input('Enter Order Status: ')
                    ))
                elif order_choice == "2":
                    temp_Data = order_manager.get_order_by_id(input(
                        'Enter Order ID: '
```

```
                    ))
                    print(temp_Data.get_order_details())
                elif order_choice == "3":
                    temp_Data = order_manager.get_order_by_id(input(
                        'Enter Order ID: '
                    ))
                    temp_Data.update_order_status(input('Enter New Status'))
                elif order_choice == "4":
                    temp_Data = order_manager.get_order_by_id(input(
                        'Enter Order ID: '
                    ))
                    temp_Data.cancel_order()
                elif order_choice == "5":
                    break
                else:
                    print("Invalid choice. Please enter a number between 1 and
5.")

        elif choice == "4":
            while True:
                inventory_management_menu()
                inventory_choice = input("Enter your choice (1-7): ")

                if inventory_choice == "1":
                    inventory_manager.add_to_inventory(input('Enter Product ID:
'), input('Enter Product Quantity: '))
                elif inventory_choice == "2":
                    inventory_manager.remove_from_inventory(input('Enter Product
ID: '), input('Enter Product Quantity: '))
                elif inventory_choice == "3":
                    inventory_manager.list_all_products()
                elif inventory_choice == "4":
                    inventory_manager.list_low_stock_products(2)
                elif inventory_choice == "5":
                    inventory_manager.list_out_of_stock_products()
                elif inventory_choice == "6":
                    inventory_manager.list_all_products()
                elif inventory_choice == "7":
                    break
                else:
                    print("Invalid choice. Please enter a number between 1 and
7.")

        elif choice == "5":
            print("Exiting TechShop. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 5.")
```

Output


Welcome to TechShop!
1. Customer Management
2. Product Management
3. Order Management
4. Inventory Management
5. Exit
Enter your choice (1-5):


Customer Management Menu:
1. Register Customer
2. View Customer Details
3. Update Customer Information
4. Back to Main Menu
Enter your choice (1-4):


Enter your choice (1-4): 2
Enter CustomerID: 1
Customer ID: 1
Name: John Doe
DOB: 1990-01-15
Email: doe.john@email.com
Phone: 1234567890
Address: Avenue Street 1, Los Angeles, USA

```
Welcome to TechShop!
1. Customer Management
2. Product Management
3. Order Management
4. Inventory Management
5. Exit
Enter your choice (1-5): 5
Exiting TechShop. Goodbye!
```

```
Product Management Menu:
1. Add Product
2. View Product Details
3. Update Product Information
4. Back to Main Menu
Enter your choice (1-4):
```

```
Enter your choice (1-5): 3

Order Management Menu:
1. Place Order
2. View Order Details
3. Update Order Status
4. Cancel Order
5. Back to Main Menu
Enter your choice (1-5):
```

```
Inventory Management Menu:
1. Add to Inventory
2. Remove from Inventory
3. View Inventory Details
4. List Low Stock Products
5. List Out of Stock Products
6. List All Products in Inventory
7. Back to Main Menu
Enter your choice (1-7):
```