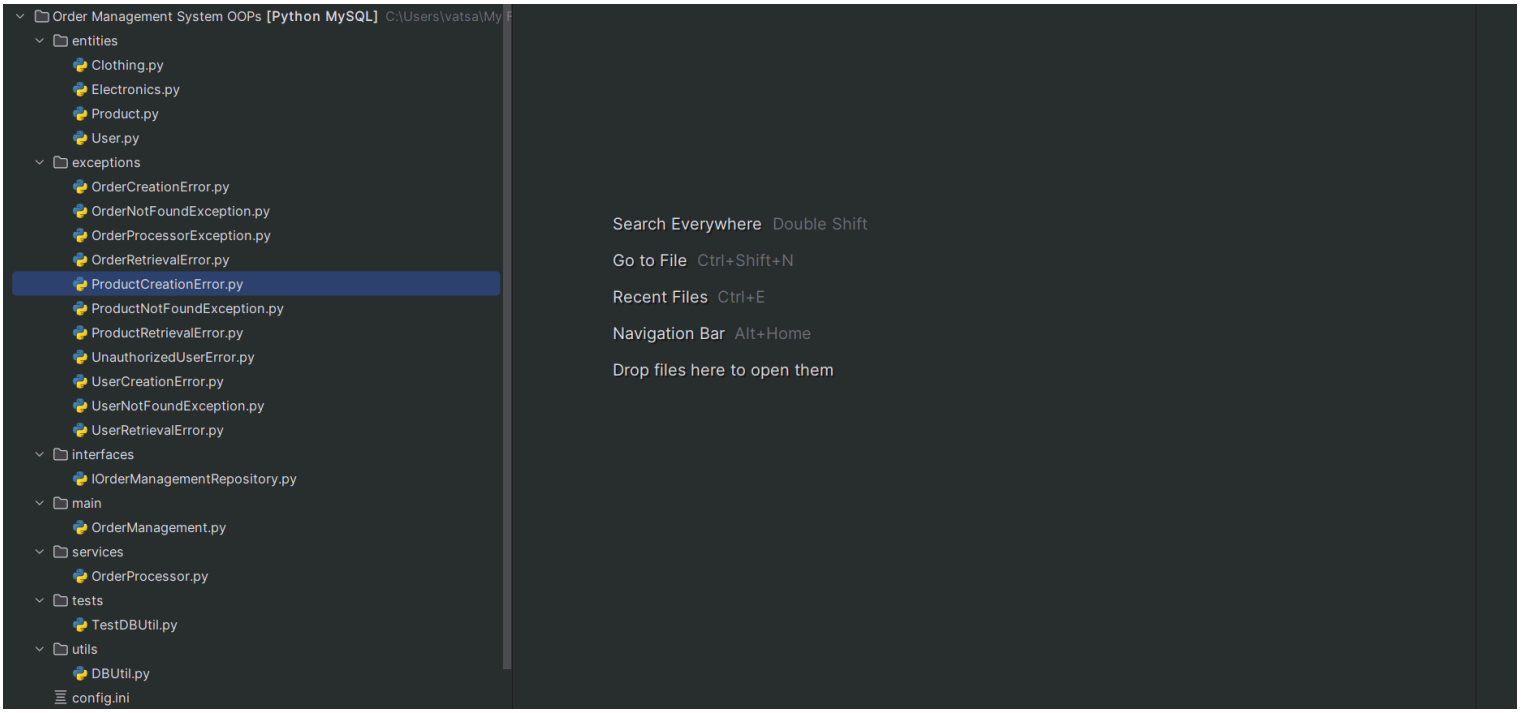
6 - Coding Challenge - Order Management System

Folder Structure



Create SQL Schema from the product and user class, and use the class attributes for table column names

SQL Tables

Users

```sql
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    role ENUM('Admin', 'User') NOT NULL
);
```

Products

```sql
CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    description TEXT,
    price DOUBLE NOT NULL,
    quantity_in_stock INT NOT NULL,
    type ENUM('Electronics', 'Clothing') NOT NULL
);
```

Orders

```sql
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    order_status ENUM('Placed', 'Cancelled', 'Delivered') DEFAULT 'Placed',
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

Order Products

```sql
CREATE TABLE order_products (
    order_id INT,
    product_id INT,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Classes

Product

```python
class Product:
    def __init__(self, productName, description, price, quantityInStock, productType):
        # self.__product_id = productId  #  Made ID auto-increment
        self.__product_name = productName
        self.__description = description
        self.__price = price
        self.__quantity_in_stock = quantityInStock
        self.__type = productType

    #  Made ID auto-increment
    # @property
    # def product_id(self):
    #     return self.__product_id
    #
    # @product_id.setter
    # def product_id(self, value):
    #     self.__product_id = value

    3 usages (2 dynamic)
    @property
    def product_name(self):
        return self.__product_name

    2 usages (2 dynamic)
    @product_name.setter
    def product_name(self, value):
        self.__product_name = value

    2 usages (1 dynamic)
    @property
    def description(self):
```

Electronics

```python
from entities.Product import Product


class Electronics(Product):
    def __init__(self, productName, description, price, quantityInStock, productType, brand, warrantyPeriod):
        super().__init__(productName, description, price, quantityInStock, productType)
        self.__brand = brand
        self.__warranty_period = warrantyPeriod

    1 usage
    @property
    def brand(self):
        return self.__brand

    @brand.setter
    def brand(self, value):
        self.__brand = value

    1 usage
    @property
    def warranty_period(self):
        return self.__warranty_period

    @warranty_period.setter
    def warranty_period(self, value):
        self.__warranty_period = value
```

Clothing

```python
from entities.Product import Product


class Clothing(Product):
    def __init__(self, productName, description, price, quantityInStock, productType, size, color):
        super().__init__(productName, description, price, quantityInStock, productType)
        self.__size = size
        self.__color = color

    1 usage
    @property
    def size(self):
        return self.__size

    @size.setter
    def size(self, value):
        self.__size = value

    1 usage
    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, value):
        self.__color = value
```

User

```python
class User:
    def __init__(self, username, password, role):
        # self.__user_id = userId  #  Made ID auto-increment
        self.__username = username
        self.__password = password
        self.__role = role

    #  Made ID auto-increment
    # @property
    # def user_id(self):
    #     return self.__user_id
    #
    # @user_id.setter
    # def user_id(self, value):
    #     self.__user_id = value

    3 usages (2 dynamic)
    @property
    def username(self):
        return self.__username

    2 usages (2 dynamic)
    @username.setter
    def username(self, value):
        self.__username = value

    3 usages (2 dynamic)
    @property
    def password(self):
        return self.__password
```

IOrderManagementRepository.py

```python
class IOrderManagementRepository(ABC):
    @abstractmethod
    def createOrder(self, user, products):
        pass

    @abstractmethod
    def cancelOrder(self, userId, orderId):
        pass

    @abstractmethod
    def createProduct(self, product):
        pass

    @abstractmethod
    def createUser(self, user):
        pass

    @abstractmethod
    def getAllProducts(self):
        pass

    @abstractmethod
    def getOrderByUser(self, user):
        pass
```

Services

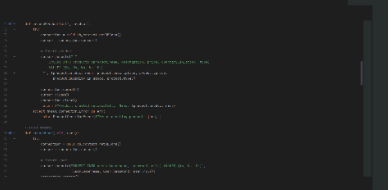
- I managed authorisation and authentication using username and password with role check wherever required like in creating a product.
- I also created an extra service called create order for creating order.

OrderProcessor

```python
class OrderProcessor(IOrderManagementRepository):
    def __init__(self, db_context):
        self.db_context = db_context

    def createOrder(self, user, products):
        try:
            connection = self.db_context.getDBConn()
            cursor = connection.cursor()

            # Insert user if not exists
            cursor.execute("INSERT IGNORE INTO users (user_id, username, password, role) VALUES (%s, %s, %s, %s)",
                           (user.user_id, user.username, user.password, user.role))

            # Insert order
            cursor.execute("INSERT INTO orders (user_id) VALUES (%s)", (user.user_id,))
            order_id = cursor.lastrowid  # Get the ID of the last inserted row

            # Insert products in the order
            for product in products:
                cursor.execute("INSERT INTO order_products (order_id, product_id) VALUES (%s, %s)",
                               (order_id, product.productId))

            connection.commit()
            cursor.close()
            connection.close()
            return f"Order created successfully with ID: {order_id}"
```

```
1 usage (1 dynamic)
def cancelOrder(self, userId, orderId):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor()

        # Check if the order with this user_id and order_id exist
        cursor.execute("SELECT * FROM orders WHERE user_id = %s AND order_id = %s", (userId, orderId))
        order = cursor.fetchone()

        if order:
            # Update the order status to "Canceled"
            cursor.execute("UPDATE orders SET order_status = 'Cancelled' WHERE order_id = %s", (orderId,))

            connection.commit()
            cursor.close()
            connection.close()
            return f"Order canceled successfully"
        else:
            raise OrderNotFoundException("Order not found")

    except mysql.connector.Error as err:
        raise OrderProcessorException(f"Error canceling order: {err}")
```

```
def createProduct(self, product):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor()

        # Insert product
        cursor.execute("""
            INSERT INTO products (product_name, description, price, quantity_in_stock, type)
            VALUES (%s, %s, %s, %s, %s)
        """, (product.product_name, product.description, product.price,
              product.quantity_in_stock, product.type))

        connection.commit()
        cursor.close()
        connection.close()
        return f"Product created successfully. Name: {product.product_name}"
    except mysql.connector.Error as err:
        raise ProductCreationError(f"Error creating product: {err}")

1 usage (1 dynamic)
def createUser(self, user):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor()

        # Insert user
        cursor.execute("INSERT INTO users (username, password, role) VALUES (%s, %s, %s)",
                       (user.username, user.password, user.role))
        connection.commit()
        cursor.close()
        connection.close()
        return f"User created successfully with ID: {user_id}"

    except mysql.connector.Error as err:
        raise UserCreationError(f"Error creating user: {err}")

2 usages (2 dynamic)
def getAllProducts(self):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor(dictionary=True)

        # Retrieve all products
        cursor.execute("SELECT * FROM products")
        products = cursor.fetchall()

        cursor.close()
        connection.close()
        return products

    except mysql.connector.Error as err:
        raise ProductRetrievalError(f"Error getting products: {err}")

1 usage (1 dynamic)
def getOrderByUser(self, user_id):
```



Snip & Sketch

Pull Requests

```
1 usage (1 dynamic)
def getOrderByUser(self, user_id):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor(dictionary=True)

        # Retrieve orders for a user_id
        cursor.execute("""
            SELECT orders.order_id, products.product_id, products.product_name
            FROM orders
            JOIN order_products ON orders.order_id = order_products.order_id
            JOIN products ON order_products.product_id = products.product_id
            WHERE orders.user_id = %s
        """, (user_id,))
        orders = cursor.fetchall()

        cursor.close()
        connection.close()

        if not orders:
            raise OrderRetrievalError("No orders found for the user.")

        return orders

    except mysql.connector.Error as err:
        raise OrderRetrievalError(f"Error getting orders by user: {err}")

3 usages (3 dynamic)
def getUserByUsernameAndPassword(self, username, password):
    try:
```

```
def getUserByUsernameAndPassword(self, username, password):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor(dictionary=True)

        cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s", (username, password))
        user = cursor.fetchone()

        cursor.close()
        connection.close()

        if user:
            return User(user['username'], user['password'], user['role']), user['user_id']
        else:
            raise UserNotFoundException("User not found with the given credentials.")

    except mysql.connector.Error as err:
        raise UserRetrievalError(f"Error retrieving user: {err}")

1 usage (1 dynamic)
def addToOrder(self, user_id, product_ids):
    try:
        connection = self.db_context.getDBConn()
        cursor = connection.cursor(dictionary=True)

        # Check if the user has an active order, if not, create one
        cursor.execute("SELECT * FROM orders WHERE user_id = %s AND order_status = 'Placed'", (user_id,))
        active_order = cursor.fetchone()
```

```
            # Get the order_id for the user's placed order
            cursor.execute("SELECT order_id FROM orders WHERE user_id = %s AND order_status = 'Placed'",
                           (user_id,))
            order_id = cursor.fetchone()['order_id']

            # Add the products to the order_products table
            for i, product_id in enumerate(product_ids):
                # Check if the product exists
                cursor.execute("SELECT * FROM products WHERE product_id = %s", (product_id,))
                product = cursor.fetchone()

                if not product:
                    raise ProductNotFoundException(f"Product with ID {product_id} not found.")

                # Add the product to the order_products table
                cursor.execute("""
                    INSERT INTO order_products (order_id, product_id)
                    VALUES (%s, %s)
                """, (order_id, product_id))
                connection.commit()

            cursor.close()
            connection.close()
            return f"Products added to the order successfully."

        except mysql.connector.Error as err:
            raise OrderCreationError(f"Error adding products to order: {err}")
```

Database Connection

DBUtil

```python
class DBUtil:
    9 usages (8 dynamic)
    @staticmethod
    def getDBConn(max_retries=3, retry_delay=2):
        config = configparser.ConfigParser()
        config.read('../config.ini')
        database_config = config['Database']

        for attempt in range(1, max_retries + 1):
            try:
                connection = mysql.connector.connect(
                    host=database_config['host'],
                    user=database_config['user'],
                    password=database_config['password'],
                    database=database_config['database']
                )
                return connection
            except mysql.connector.Error as err:
                print(f"Error: {err}")
                if attempt < max_retries:
                    print(f"Retrying connection (Attempt {attempt}/{max_retries})...")
                    time.sleep(retry_delay)
                else:
                    print(f"Max retries reached. Unable to establish a connection.")
                    return None
```

Added a config.ini file to better manage the db credentials

config.ini

```
[Database]
host = localhost
user = root
password = root
database = order_management_system
```

Main

OrderManagement

```python
class OrderManagement:
    def __init__(self, new_order_processor):
        self.order_processor = new_order_processor

    1 usage
    @staticmethod
    def display_menu():
        print("\nOrder Management System Menu:")
        print("1. Create User")
        print("2. Create Product")
        print("3. Create Order")
        print("4. Cancel Order")
        print("5. Get All Products")
        print("6. Get Orders by User")
        print("7. Exit")
```

```
    def main(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            if choice == "1":   # Create User
                username = input("\nEnter Username: ")
                password = input("Enter Password: ")
                role = input("Enter Role (Admin/User): ")

                user = User(username, password, role)
                result = self.order_processor.createUser(user)
                print(result)

            elif choice == "2":  # Create Product
                username = input("\nEnter Username: ")
                password = input("Enter Password: ")

                try:
                    user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
                    if user.role == "User":
                        raise UnauthorizedUserError("Permission denied. User must be an Admin to create a product.")

                    product_name = input("\nEnter Product Name: ")
                    description = input("Enter Product Description: ")
                    price = float(input("Enter Product Price: "))
                    quantity_in_stock = int(input("Enter Quantity in Stock: "))
                    product_type = input("Enter Product Type (Electronics/Clothing): ")

                    product = Product(product_name, description, price, quantity_in_stock, product_type)
```

```
                    price = float(input("Enter Product Price: "))
                    quantity_in_stock = int(input("Enter Quantity in Stock: "))
                    product_type = input("Enter Product Type (Electronics/Clothing): ")

                    product = Product(product_name, description, price, quantity_in_stock, product_type)

                    try:
                        result = self.order_processor.createProduct(product)
                        print(result)

                    except ProductCreationError as e:
                        print(f"Error: {e}")

                except UnauthorizedUserError as e:
                    print(f"Error: {e}")

                except UserNotFoundException as e:
                    print(f"Error: {e}")

            elif choice == "3":  # Create Order
                username = input("\nEnter Username: ")
                password = input("Enter Password: ")
```

```
            try:
                user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)

                # Display a list of available products
                products = self.order_processor.getAllProducts()
                print("\nAvailable Products:")
                for product in products:
                    print(f"{product['product_id']}. {product['product_name']} - ${product['price']}")

                # Take input for multiple products to be added to the order
                product_input = input("Enter Product IDs to add to the order (comma-separated): ")
                product_ids = [int(product_id) for product_id in product_input.split(",")]

                # Add products to the order
                result = self.order_processor.addToOrder(user_id, product_ids)
                print(result)

            except (UserNotFoundException, UserRetrievalError, ProductRetrievalError, UnauthorizedUserError) as e:
                print(f"Error: {e}")

        elif choice == "4":  # Cancel Order
            username = input("\nEnter Username: ")
            password = input("Enter Password: ")
            order_id = int(input("Enter Order ID: "))

            try:
                user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
```

```
                user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
                result = self.order_processor.cancelOrder(user_id, order_id)
                print(result)
            except OrderNotFoundException as e:
                print(f"Error: {e}")

        elif choice == "5":  # Get All Products
            try:
                products = self.order_processor.getAllProducts()
                print("\nAll Products:")
                for product in products:
                    print(product)
            except ProductRetrievalError as e:
                print(f"Error: {e}")

        elif choice == "6":  # Get Orders by User
            user_id = int(input("Enter User ID: "))

            try:
                orders = self.order_processor.getOrderByUser(user_id)
                print(f"Orders for User {user_id}:")
                for order in orders:
                    print(order)
            except OrderRetrievalError as e:
```

Exceptions

I also created Exceptions and used them in OrderMangement.py and OrderProcessor.py

OrderCreationError.py

```
2 usages
class OrderCreationError(Exception):
    def __init__(self, message="Order Creation Error"):
        self.message = message
        super().__init__(self.message)
```

OrderCreationError.py | OrderNotFoundException.py

```
4 usages
class OrderNotFoundException(Exception):
    def __init__(self, message="Order Not Found Exception"):
        self.message = message
        super().__init__(self.message)
```

OrderCreationError.py | OrderNotFoundException.py | OrderProcessorException.py

```python
3 usages
class OrderProcessorException(Exception):
    def __init__(self, message="Order Processing Exception"):
        self.message = message
        super().__init__(self.message)
```

OrderCreationError.py | OrderNotFoundException.py | OrderProcessorException.py | OrderRetrievalError.py

```python
5 usages
class OrderRetrievalError(Exception):
    def __init__(self, message="Order Retrieval Error"):
        self.message = message
        super().__init__(self.message)
```

OrderNotFoundException.py | OrderProcessorException.py | OrderRetrievalError.py | ProductCreationError.py

```python
4 usages
class ProductCreationError(Exception):
    def __init__(self, message="Product Creation Error"):
        self.message = message
        super().__init__(self.message)
```

OrderRetrievalError.py | ProductCreationError.py | ProductRetrievalError.py | ProductNotFoundException.py

```python
2 usages
class ProductNotFoundException(Exception):
    def __init__(self, message="Product Not Found Exception"):
        self.message = message
        super().__init__(self.message)
```

OrderRetrievalError.py | ProductCreationError.py | ProductRetrievalError.py | ProductNotFoundException.py

```python
5 usages
class ProductRetrievalError(Exception):
    def __init__(self, message="Product Retrieval Error"):
        self.message = message
        super().__init__(self.message)
```

OrderRetrievalError.py | ProductCreationError.py | ProductRetrievalError.py | ProductNotFoundException.py

```python
5 usages
class ProductRetrievalError(Exception):
    def __init__(self, message="Product Retrieval Error"):
        self.message = message
        super().__init__(self.message)
```

OrderRetrievalError.py | ProductCreationError.py | ProductRetrievalError.py | UnauthorizedUserError.py

```python
5 usages
class UnauthorizedUserError(Exception):
    def __init__(self, message="Unauthorized User Error"):
        self.message = message
        super().__init__(self.message)
```

ProductCreationError.py | ProductRetrievalError.py | UnauthorizedUserError.py | UserCreationError.py

```python
2 usages
class UserCreationError(Exception):
    def __init__(self, message="User Creation Error"):
        self.message = message
        super().__init__(self.message)
```

ProductRetrievalError.py | UnauthorizedUserError.py | UserCreationError.py | UserNotFoundException.py

```python
5 usages
class UserNotFoundException(Exception):
    def __init__(self, message="User Not Found Exception"):
        self.message = message
        super().__init__(self.message)
```

UnauthorizedUserError.py | UserCreationError.py | UserNotFoundException.py | UserRetrievalError.py

```python
4 usages
class UserRetrievalError(Exception):
    def __init__(self, message="User Retrieval Error"):
        self.message = message
        super().__init__(self.message)
```

Terminal User Interface

```
Order Management System Menu:
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders by User
7. Exit
Enter your choice:
```

```
Enter your choice: 2

Enter Username: vatsal
Enter Password: root

Enter Product Name: abcd
Enter Product Description: sdasasdsada
Enter Product Price: 3000
Enter Quantity in Stock: 20
Enter Product Type (Electronics/Clothing): Clothing
Product created successfully. Name: abcd
```

```
7. Exit
Enter your choice: 5

All Products:
{'product_id': 1, 'product_name': 'qwe', 'description': 'rtytrtyrrtytr', 'price': 1111.0, 'quantity_in_stock': 111, 'type': 'Electronics'}
{'product_id': 2, 'product_name': 'Earphones', 'description': 'drfaskflnaslkfnakl najkslnf;akjl naks;jnaskfjnfn;askfas;fkasfn asklfnas', 'price': 11111.0, 'quantity_in_stock':
{'product_id': 3, 'product_name': 'abcd', 'description': 'sdasasdsada', 'price': 3000.0, 'quantity_in_stock': 20, 'type': 'Clothing'}

Order Management System Menu:
1. Create User
```

```
6. Get Orders by User
7. Exit
Enter your choice: 6
Enter User ID: 2
Orders for User 2:
{'order_id': 1, 'product_id': 1, 'product_name': 'qwe'}
{'order_id': 1, 'product_id': 2, 'product_name': 'Earphones'}
```

Conclusion

Overall it is a full-fledged backend and database connection implementation. I recommend you check the project file by file to see all the features and miscellaneous things implemented.

Thank You!