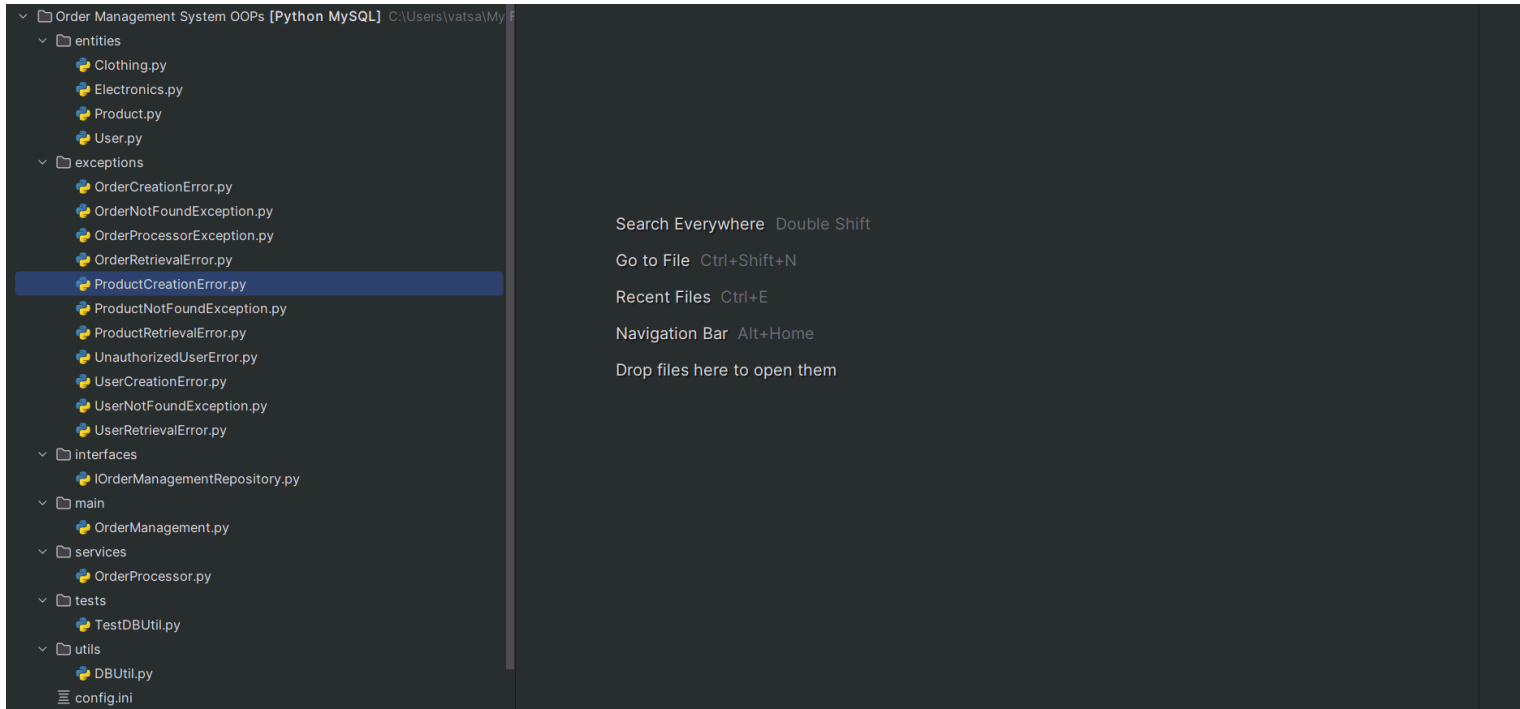


6 - Coding Challenge - Order Management System

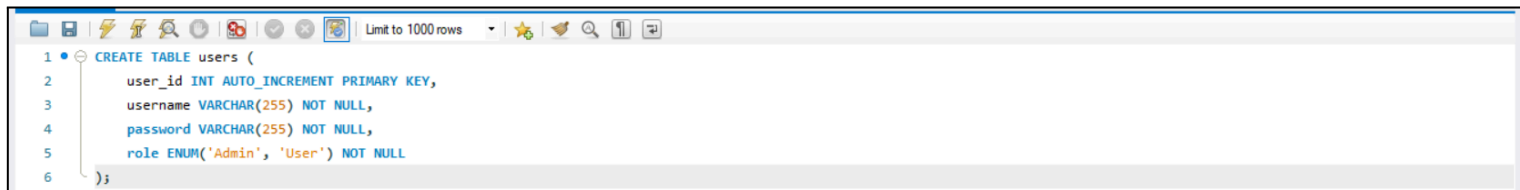
Folder Structure



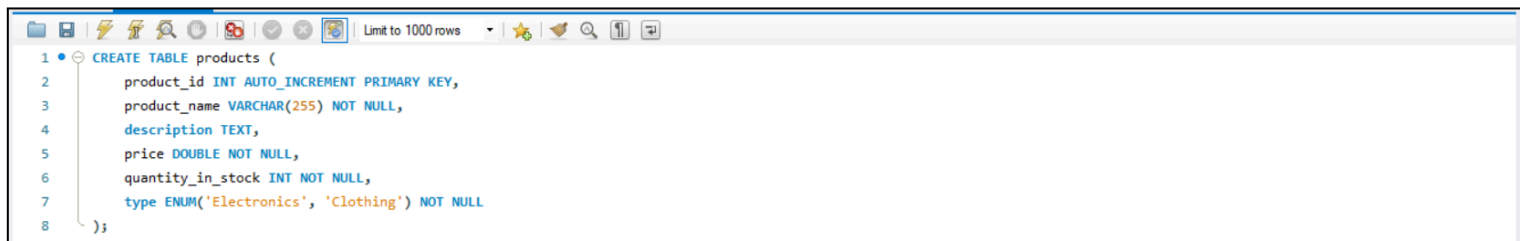
Create SQL Schema from the product and user class, and use the class attributes for table column names

SQL Tables

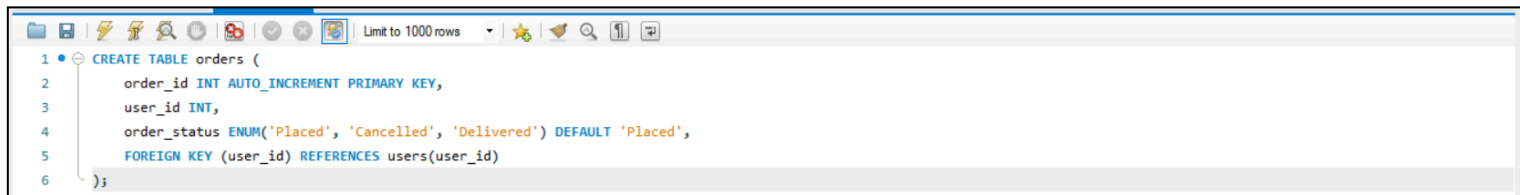
Users



Products



Orders



Order Products

```
1 CREATE TABLE order_products (  
2     order_id INT,  
3     product_id INT,  
4     PRIMARY KEY (order_id, product_id),  
5     FOREIGN KEY (order_id) REFERENCES orders(order_id),  
6     FOREIGN KEY (product_id) REFERENCES products(product_id)  
7 );
```

Classes

Product

```
1 class Product: ✓  
2     def __init__(self, productName, description, price, quantityInStock, productType):  
3         # self.__product_id = productId # Made ID auto-increment  
4         self.__product_name = productName  
5         self.__description = description  
6         self.__price = price  
7         self.__quantity_in_stock = quantityInStock  
8         self.__type = productType  
9  
10        # Made ID auto-increment  
11        # @property  
12        # def product_id(self):  
13        #     return self.__product_id  
14        #  
15        # @product_id.setter  
16        # def product_id(self, value):  
17        #     self.__product_id = value  
18  
19        3 usages (2 dynamic)  
20        @property  
21        def product_name(self):  
22            return self.__product_name  
23  
24        2 usages (2 dynamic)  
25        @product_name.setter  
26        def product_name(self, value):  
27            self.__product_name = value  
28  
29        2 usages (1 dynamic)  
30        @property  
31        def description(self):
```

Electronics

```
1 from entities.Product import Product
2
3
4 class Electronics(Product):
5     def __init__(self, productName, description, price, quantityInStock, productType, brand, warrantyPeriod):
6         super().__init__(productName, description, price, quantityInStock, productType)
7         self.__brand = brand
8         self.__warranty_period = warrantyPeriod
9
10    1 usage
11    @property
12    def brand(self):
13        return self.__brand
14
15    @brand.setter
16    def brand(self, value):
17        self.__brand = value
18
19    1 usage
20    @property
21    def warranty_period(self):
22        return self.__warranty_period
23
24    @warranty_period.setter
25    def warranty_period(self, value):
26        self.__warranty_period = value
```

Clothing

```
1 from entities.Product import Product
2
3
4 class Clothing(Product):
5     def __init__(self, productName, description, price, quantityInStock, productType, size, color):
6         super().__init__(productName, description, price, quantityInStock, productType)
7         self.__size = size
8         self.__color = color
9
10    1 usage
11    @property
12    def size(self):
13        return self.__size
14
15    @size.setter
16    def size(self, value):
17        self.__size = value
18
19    1 usage
20    @property
21    def color(self):
22        return self.__color
23
24    @color.setter
25    def color(self, value):
26        self.__color = value
```

User

```
1 class User:
2     def __init__(self, username, password, role):
3         # self.__user_id = userId # Made ID auto-increment
4         self.__username = username
5         self.__password = password
6         self.__role = role
7
8     # Made ID auto-increment
9
10    # @property
11    # def user_id(self):
12    #     return self.__user_id
13    #
14    # @user_id.setter
15    # def user_id(self, value):
16    #     self.__user_id = value
17
18    3 usages (2 dynamic)
19    @property
20    def username(self):
21        return self.__username
22
23    2 usages (2 dynamic)
24    @username.setter
25    def username(self, value):
26        self.__username = value
27
28    3 usages (2 dynamic)
29    @property
30    def password(self):
31        return self.__password
```

IOrderManagementRepository.py

```
2 usages
4 @l v class IOrderManagementRepository(ABC):
5     @abstractmethod
6 @l v def createOrder(self, user, products):
7         pass
8
9     1 usage (1 dynamic)
10    @abstractmethod
11 @l v def cancelOrder(self, userId, orderId):
12         pass
13
14    1 usage (1 dynamic)
15    @abstractmethod
16 @l v def createProduct(self, product):
17         pass
18
19    1 usage (1 dynamic)
20    @abstractmethod
21 @l v def createUser(self, user):
22         pass
23
24    2 usages (2 dynamic)
25    @abstractmethod
26 @l v def getAllProducts(self):
27         pass
28
29    1 usage (1 dynamic)
30    @abstractmethod
31 @l v def getOrderByUser(self, user):
32         pass
33
34    1 usage (1 dynamic)
```

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

```
18 v class OrderProcessor(IOrderManagementRepository):
19 v     def __init__(self, db_context):
20         self.db_context = db_context
21
22 @l v def createOrder(self, user, products):
23 v     try:
24         connection = self.db_context.getDBConn()
25         cursor = connection.cursor()
26
27         # Insert user if not exists
28         cursor.execute("INSERT IGNORE INTO users (user_id, username, password, role) VALUES (%s, %s, %s, %s)",
29             (user.user_id, user.username, user.password, user.role))
30
31         # Insert order
32         cursor.execute("INSERT INTO orders (user_id) VALUES (%s)", (user.user_id,))
33         order_id = cursor.lastrowid # Get the ID of the last inserted row
34
35         # Insert products in the order
36 v         for product in products:
37             cursor.execute("INSERT INTO order_products (order_id, product_id) VALUES (%s, %s)",
38                 (order_id, product.product_id))
39
40         connection.commit()
41         cursor.close()
42         connection.close()
43         return f"Order created successfully with ID: {order_id}"
44
```

```

48 1 usage (1 dynamic)
49 def cancelOrder(self, userId, orderId):
50     try:
51         connection = self.db_context.getDBConn()
52         cursor = connection.cursor()
53
54         # Check if the order with this user_id and order_id exist
55         cursor.execute("SELECT * FROM orders WHERE user_id = %s AND order_id = %s", (userId, orderId))
56         order = cursor.fetchone()
57
58         if order:
59             # Update the order status to "Canceled"
60             cursor.execute("UPDATE orders SET order_status = 'Cancelled' WHERE order_id = %s", (orderId,))
61
62             connection.commit()
63             cursor.close()
64             connection.close()
65             return f"Order canceled successfully"
66         else:
67             raise OrderNotFoundException("Order not found")
68
69     except mysql.connector.Error as err:
70         raise OrderProcessorException(f"Error canceling order: {err}")

```

```

71 def createProduct(self, product):
72     try:
73         connection = self.db_context.getDBConn()
74         cursor = connection.cursor()
75
76         # Insert product
77         cursor.execute("""
78             INSERT INTO products (product_name, description, price, quantity_in_stock, type)
79             VALUES (%s, %s, %s, %s, %s)
80             """, (product.product_name, product.description, product.price,
81                 product.quantity_in_stock, product.type))
82
83         connection.commit()
84         cursor.close()
85         connection.close()
86         return f"Product created successfully. Name: {product.product_name}"
87     except mysql.connector.Error as err:
88         raise ProductCreationError(f"Error creating product: {err}")
89
90 1 usage (1 dynamic)

```

```

91 def createUser(self, user):
92     try:
93         connection = self.db_context.getDBConn()
94         cursor = connection.cursor()
95
96         # Insert user
97         cursor.execute("INSERT INTO users (username, password, role) VALUES (%s, %s, %s)",
98                         (user.username, user.password, user.role))
99         connection.commit()
100         return f"User created successfully with ID: {user_id}"

```

```

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

```

```

103 2 usages (2 dynamic)
104 def getAllProducts(self):
105     try:
106         connection = self.db_context.getDBConn()
107         cursor = connection.cursor(dictionary=True)
108
109         # Retrieve all products
110         cursor.execute("SELECT * FROM products")
111         products = cursor.fetchall()
112
113         cursor.close()
114         connection.close()
115         return products
116
117     except mysql.connector.Error as err:
118         raise ProductRetrievalError(f"Error getting products: {err}")

```

```

119 1 usage (1 dynamic)
120 def getOrderByUser(self, user_id):

```

```

121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

```

Snip & Sketch



```

126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
Pull Requests
usage (1 dynamic)
def getOrderById(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()

```

```

183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
# 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

```
6 class DBUtil:
    9 usages (8 dynamic)
7     @staticmethod
8     def getDBConn(max_retries=3, retry_delay=2):
9         config = configparser.ConfigParser()
10        config.read('../config.ini')
11        database_config = config['Database']
12
13        for attempt in range(1, max_retries + 1):
14            try:
15                connection = mysql.connector.connect(
16                    host=database_config['host'],
17                    user=database_config['user'],
18                    password=database_config['password'],
19                    database=database_config['database']
20                )
21                return connection
22            except mysql.connector.Error as err:
23                print(f"Error: {err}")
24                if attempt < max_retries:
25                    print(f"Retrying connection (Attempt {attempt}/{max_retries})...")
26                    time.sleep(retry_delay)
27                else:
28                    print(f"Max retries reached. Unable to establish a connection.")
29                return None
```

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

```
config.ini x
1 [Database]
2 host = localhost
3 user = root
4 password = root
5 database = order_management_system
6
```

Main

OrderManagement

```
14 class OrderManagement:
15     def __init__(self, new_order_processor):
16         self.order_processor = new_order_processor
17
18     1 usage
19     @staticmethod
20     def display_menu():
21         print("\nOrder Management System Menu:")
22         print("1. Create User")
23         print("2. Create Product")
24         print("3. Create Order")
25         print("4. Cancel Order")
26         print("5. Get All Products")
27         print("6. Get Orders by User")
28         print("7. Exit")
```



```

29  def main(self):
30      while True:
31          self.display_menu()
32          choice = input("Enter your choice: ")
33
34          if choice == "1": # Create User
35              username = input("\nEnter Username: ")
36              password = input("Enter Password: ")
37              role = input("Enter Role (Admin/User): ")
38
39              user = User(username, password, role)
40              result = self.order_processor.createUser(user)
41              print(result)
42
43          elif choice == "2": # Create Product
44              username = input("\nEnter Username: ")
45              password = input("Enter Password: ")
46
47              try:
48                  user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
49                  if user.role == "User":
50                      raise UnauthorizedUserError("Permission denied. User must be an Admin to create a product.")
51
52                  product_name = input("\nEnter Product Name: ")
53                  description = input("Enter Product Description: ")
54                  price = float(input("Enter Product Price: "))
55                  quantity_in_stock = int(input("Enter Quantity in Stock: "))
56                  product_type = input("Enter Product Type (Electronics/Clothing): ")
57
58                  product = Product(product_name, description, price, quantity_in_stock, product_type)

```

```

54              price = float(input("Enter Product Price: "))
55              quantity_in_stock = int(input("Enter Quantity in Stock: "))
56              product_type = input("Enter Product Type (Electronics/Clothing): ")
57
58              product = Product(product_name, description, price, quantity_in_stock, product_type)
59
60              try:
61                  result = self.order_processor.createProduct(product)
62                  print(result)
63
64              except ProductCreationError as e:
65                  print(f"Error: {e}")
66
67              except UnauthorizedUserError as e:
68                  print(f"Error: {e}")
69
70              except UserNotFoundException as e:
71                  print(f"Error: {e}")
72
73          elif choice == "3": # Create Order
74              username = input("\nEnter Username: ")
75              password = input("Enter Password: ")
76

```

```

77     try:
78         user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
79
80         # Display a list of available products
81         products = self.order_processor.getAllProducts()
82         print("\nAvailable Products:")
83         for product in products:
84             print(f"{product['product_id']}. {product['product_name']} - ${product['price']}")
85
86         # Take input for multiple products to be added to the order
87         product_input = input("Enter Product IDs to add to the order (comma-separated): ")
88         product_ids = [int(product_id) for product_id in product_input.split(",")]
89
90         # Add products to the order
91         result = self.order_processor.addToOrder(user_id, product_ids)
92         print(result)
93
94     except (UserNotFoundException, UserRetrievalError, ProductRetrievalError, UnauthorizedUserError) as e:
95         print(f"Error: {e}")
96
97 elif choice == "4": # Cancel Order
98     username = input("\nEnter Username: ")
99     password = input("Enter Password: ")
100     order_id = int(input("Enter Order ID: "))
101
102     try:
103         user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)

```

```

102     try:
103         user, user_id = self.order_processor.getUserByUsernameAndPassword(username, password)
104         result = self.order_processor.cancelOrder(user_id, order_id)
105         print(result)
106     except OrderNotFoundException as e:
107         print(f"Error: {e}")
108
109 elif choice == "5": # Get All Products
110     try:
111         products = self.order_processor.getAllProducts()
112         print("\nAll Products:")
113         for product in products:
114             print(product)
115     except ProductRetrievalError as e:
116         print(f"Error: {e}")
117
118 elif choice == "6": # Get Orders by User
119     user_id = int(input("Enter User ID: "))
120
121     try:
122         orders = self.order_processor.getOrderByUser(user_id)
123         print(f"Orders for User {user_id}:")
124         for order in orders:
125             print(order)
126     except OrderRetrievalError as e:

```

Exceptions

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

OrderCreationError.py

```

1 2 usages
1 class OrderCreationError(Exception):
2     def __init__(self, message="Order Creation Error"):
3         self.message = message
4         super().__init__(self.message)

```

OrderCreationError.py OrderNotFoundException.py

```

1 4 usages
1 class OrderNotFoundException(Exception):
2     def __init__(self, message="Order Not Found Exception"):
3         self.message = message
4         super().__init__(self.message)

```

OrderCreationError.py OrderNotFoundException.py OrderProcessorException.py ×

3 usages

1 class OrderProcessorException(Exception):

2 def __init__(self, message="Order Processing Exception"):

3 self.message = message

4 super().__init__(self.message)

OrderCreationError.py × OrderNotFoundException.py OrderProcessorException.py OrderRetrievalError.py ×

5 usages

1 class OrderRetrievalError(Exception):

2 def __init__(self, message="Order Retrieval Error"):

3 self.message = message

4 super().__init__(self.message)

OrderNotFoundException.py OrderProcessorException.py OrderRetrievalError.py ProductCreationError.py ×

4 usages

1 class ProductCreationError(Exception):

2 def __init__(self, message="Product Creation Error"):

3 self.message = message

4 super().__init__(self.message)

OrderRetrievalError.py ProductCreationError.py ProductRetrievalError.py ProductNotFoundException.py ×

2 usages

1 class ProductNotFoundException(Exception):

2 def __init__(self, message="Product Not Found Exception"):

3 self.message = message

4 super().__init__(self.message)

OrderRetrievalError.py ProductCreationError.py ProductRetrievalError.py × ProductNotFoundException.py

5 usages

1 class ProductRetrievalError(Exception):

2 def __init__(self, message="Product Retrieval Error"):

3 self.message = message

4 super().__init__(self.message)

OrderRetrievalError.py ProductCreationError.py ProductRetrievalError.py × ProductNotFoundException.py

5 usages

1 class ProductRetrievalError(Exception):

2 def __init__(self, message="Product Retrieval Error"):

3 self.message = message

4 super().__init__(self.message)

OrderRetrievalError.py ProductCreationError.py ProductRetrievalError.py UnauthorizedUserError.py ×

5 usages

1 class UnauthorizedUserError(Exception):

2 def __init__(self, message="Unauthorized User Error"):

3 self.message = message

4 super().__init__(self.message)

```
ProductCreationError.py ProductRetrievalError.py UnauthorizedUserError.py UserCreationError.py x v :
1 2 usages
2 class UserCreationError(Exception):
3     def __init__(self, message="User Creation Error"):
4         self.message = message
5         super().__init__(self.message)
```

```
ProductRetrievalError.py UnauthorizedUserError.py UserCreationError.py UserNotFoundException.py x v :
1 5 usages
2 class UserNotFoundException(Exception):
3     def __init__(self, message="User Not Found Exception"):
4         self.message = message
5         super().__init__(self.message)
```

```
UnauthorizedUserError.py UserCreationError.py UserNotFoundException.py UserRetrievalError.py x v :
1 4 usages
2 class UserRetrievalError(Exception):
3     def __init__(self, message="User Retrieval Error"):
4         self.message = message
5         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': 'rtytrtyrtytr', 'price': 1111.0, 'quantity_in_stock': 111, 'type': 'Electronics'}
{'product_id': 2, 'product_name': 'Earphones', 'description': 'drfaskflnaslknal najkslnf;akjl naks;jnaskfjfnf;askfas;fkasfn asklnas', 'price': 11111.0, 'quantity_in_stock': 11111}
{'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!