Zuhar Regal 4227160
Exercise 3
Due date : 8 May 2024 - 13:00

Introduction

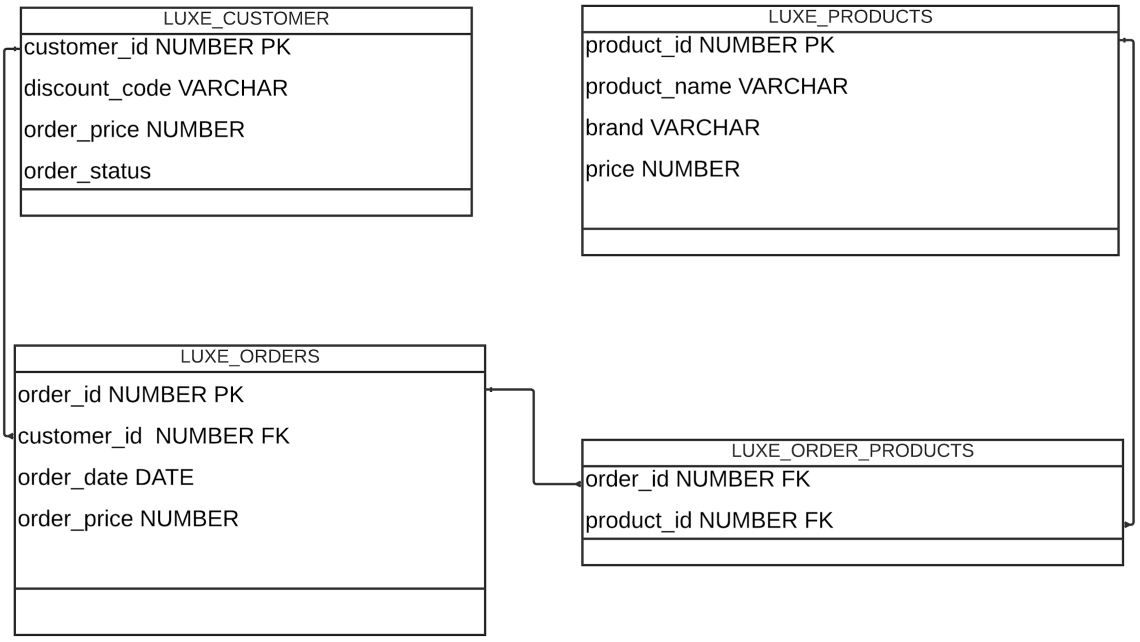
Based on my performance as the Junior Python developer at the online retail store, LUXE that specializes in selling apparel and accessories. I have been given an opportunity to produce more complex algorithms which the company hopes to deploy in its production online store. My first task is to enhance the existing order management system by implementing new features and improving its functionality.

Background

The current system is largely based on Python and requires an algorithm that calculates the total cost of a customer's shopping cart, applies discounts if a valid discount code is provided, and displays the final order total to the customer. However, the company wants to keep track of all orders placed and gain insights into sales data for future analysis.

1. Order Tracking: Implement a feature that allows the system to store order the details in a persistent manner, such as a CSV file or a database. Each order should have a unique identifier (order ID) assigned to it, as well as the relevant order details associated with a given order.

Using Oracle Apex i have created LUXE DATABASE



2. Discount Validation: Enhance the system so that users can supply a valid discount code which should apply a discount as a percentage of the total order. Order Summary: Provide a summary of the order details to the customer after each successful transaction.

Customer_functionality

1. Importing Libraries: The Customer functionality code begins by importing necessary libraries: `http.client` for making HTTP requests and `json` for handling JSON data.

2. Fetching Data Function: The `fetch_data` function is defined to fetch data from an API. It takes an API URL as input, sends a GET request to the API, and returns the data if the response status is 200 (OK).

3. Customer Functionality: The `customer_functionality` function handles the customer's interaction:

- Input: It prompts the user to enter their discount code and customer ID.

- Fetching Customer Data: It fetches customer data from an API endpoint based on the provided customer ID.

- Validation: If customer data is successfully fetched and the customer exists, it proceeds to check if a discount code is required and validates it.

- Discount Code Validation: If a discount code is required and provided, it checks if the provided code matches the code associated with the customer. If the code matches or no discount code is required, it proceeds with the checkout process.

- Order Price Calculation: It calculates the order price before any discount is applied and then calculates the shipping cost based on the total order price. If the order qualifies for free shipping (over R500), the shipping cost is set to R0; otherwise, it's set to R60.

- Discount Application: If a discount code is provided and applicable, it applies a 15% discount to the order price.

- Display: It displays the order price before and after any discounts, shipping cost, and order status.

4. Function Call: Finally, the `customer_functionality` function is called to execute the customer interaction process.

Overall, this code allows a customer to input their discount code and customer ID, validates the discount code, calculates the order price, shipping cost, and displays the order details including the order status. It also handles scenarios where a discount code is not applicable.

IF DISCOUNT CODE NONE

```
Enter your discount code (or 'NONE' if not applicable): NONE
Enter your customer ID: 28
Connection successful!
Discount code is correct.
Order Price (before discount): R319
Shipping Cost: R60
```

```
Order Price (before discount): R319
Shipping Cost: R60
Order Price (after shipping, no discount applied): R379
Order Status: DELIVERED

Process finished with exit code 0
```

IF DISCOUNT CODE APPLICABLE

```
Enter your discount code (or 'NONE' if not applicable): DEFRG
Enter your customer ID: 24
Connection successful!
Discount code is correct.
Order Price (before discount): R319
Shipping Cost: R60
```

```
Order Price (before discount): R319
Shipping Cost: R60
Order Price (after 15% discount and shipping): R331.15
Order Status: DELIVERED

Process finished with exit code 0
```

IF DISCOUNT CODE INCORRECT

```
Enter your discount code (or 'NONE' if not applicable): FERGN
Enter your customer ID: 24
Connection successful!
Invalid discount code. Please try again.

Process finished with exit code 0
```

Staff Functionality

1. Importing Libraries: The code starts by importing necessary libraries: `http.client` for making HTTP requests and `json` for handling JSON data.

2. Fetching Data Function: The `fetch_data` function is defined to fetch data from an API. It takes an API URL as input, sends a GET request to the API using HTTPS connection, and returns the fetched data if the response status is 200 (OK). Otherwise, it prints a message indicating the failure to fetch data.

3. Staff Functionality: The `staff_functionality` function handles the functionality for staff members.

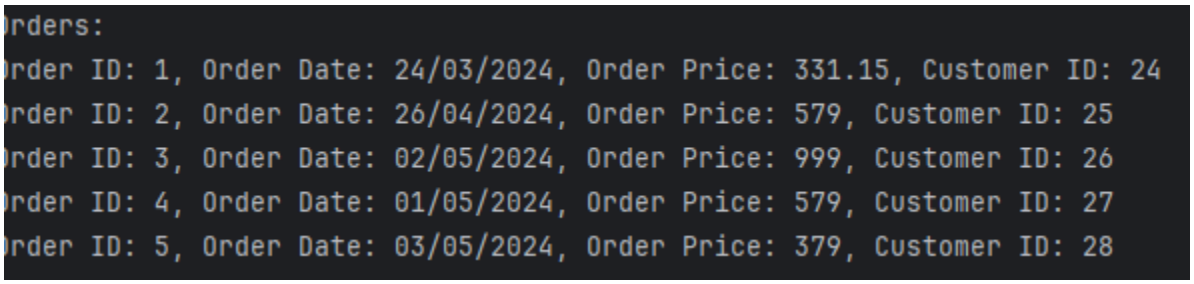
- Fetching All Orders Data: It fetches data for all orders from an API endpoint.

- Validation: If data for all orders is successfully fetched and there are items in the response, it proceeds to process and display the order details.

- Displaying Order Details: For each order in the fetched data, it extracts the order ID, order date, order price, and customer ID, and prints these details in a formatted manner.

4. Function Call: Finally, the `staff_functionality` function is called to execute the staff functionality.

Overall, this code allows staff members to fetch and view all order details, including the order ID, order date, order price, and customer ID. It provides a simple and straightforward way for staff members to access this information and view their sales with each date for analysis.



Conclusion

Both the 'customer_functionality' and 'staff_functionality' functions are intended to be modular and expandable, with functionality customized to certain user roles. Here's how they accomplish extensibility and usefulness.

1. Modularity: Every functionality is contained within its own function. This modular design enables simple maintenance, testing, and updates. If a certain function has to be modified or enhanced, developers can do it without affecting other portions of the software.

2. Parameterization: These functions allow input parameters as needed, such as customer ID and discount code for 'customer_functionality', but no input parameters for'staff_functionality'. This provides flexibility in usage and customisation based on various scenarios or requirements.

3. Code Reuse: Both functions utilize a common `fetch_data` function to fetch data from an API. This promotes code reuse and avoids duplication, enhancing maintainability and reducing the chance of errors.

4. Error Handling: The code incorporates error handling features, such as checking for successful responses (status code 200) in the 'fetch_data' function and validating fetched data prior to processing it. This ensures robustness and resilience when dealing with unforeseen scenarios, such as API failures or erroneous data.

5. Scalability: The methods can handle larger datasets or additional functionality by using pagination or adding new logic to the existing structure. For example, if more complicated operations are required, such as data manipulation or analysis, these functions can be increased without causing substantial modifications to the overall design.

6. User Interaction: Both functions feature prompts for user input, such as the customer ID and discount code for 'customer_functionality', resulting in a user-friendly interface for interaction. This improves usability and makes functionalities available to non-technical people.

7. Clear comments: The code contains comments and descriptive variable names to increase readability and maintainability. This guarantees that other developers can readily grasp the code's purpose and functionality, allowing for collaboration and future updates.

Overall, by adhering to the principles of modularity, parameterization, code reuse, error handling, scalability, user interaction, and clear documentation, the 'customer_functionality' and'staff_functionality' functions are extensible and functional, capable of meeting various requirements and adapting to changes in the application's scope.

End