Case Study on Ecommerce Application

Instructions

- Project submissions should be done through the partcipants' Github repository, and the link should be shared with trainers and Hexavarsity.
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Ecommerce** implemented with a strong focus on **SQL**, **control flow statements, loops, arrays, collections, exception handling**, **database interaction** and **Unit Testing**.
- Follow **object-oriented principles** throughout the project. Use classes and objects to model real-world entities, **encapsulate data and behavior**, and **ensure code reusability**.
- Throw **user defined exceptions** from corresponding methods and handled**.**
- The following **Directory structure** is to be followed in the application.
    - **entity/model**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a **DBPropertyUtil** class with a static function which takes property file name as parameter and returns connection string.
        - Create a **DBConnUtil** class which holds **static method** which takes connection string as parameter file and returns **connection object(Use method defined in DBPropertyUtil class to get the connection String)**.
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.

Key Functionalities:

1. **Customer Management**
    - Add new customers, Update, and retrieve customer information and order details,
2. **Product Management:**
    - Users can view a list of available products, add, and delete products.
3. **Cart Management:**
    - Users can add and remove products to their shopping cart.
4. **Order Management:**
    - Users can place orders, which include product details, quantities, and shipping information.
    - The order total is calculated based on the cart contents.

Create following tables in SQL Schema with appropriate class and write the unit test case for the Ecommerce application.

Schema Design:

1. **customers** table:
    - customer_id (Primary Key)
    - name
    - email
    - password
2. **products** table:
    - product_id (Primary Key)
    - name
    - price
    - description
    - stockQuantity
3. **cart** table:
    - cart_id (Primary Key)
    - customer_id (Foreign Key)
    - product_id (Foreign Key)
    - quantity
4. **orders** table:
    - order_id (Primary Key)
    - customer_id (Foreign Key)
    - order_date
    - total_price
    - shipping_address
5. **order_items** table (to store order details):
    - order_item_id (Primary Key)
    - order_id (Foreign Key)
    - product_id (Foreign Key)
    - quantity

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

6. **Service Provider Interface/Abstract class:**
   Keep the interfaces and implementation classes in package dao
    - Define an **OrderProcessorRepository** interface/abstract class with methods for adding/removing products to/from the cart and placing orders. The following methods will interact with database.
        1. **createProduct()**
           parameter: Product product
           return type: boolean

2. **createCustomer()**
   parameter: Customer customer
   return type: boolean
3. **deleteProduct()**
   parameter: productId
   return type: boolean
4. **deleteCustomer(customerId)**
   parameter: customerId
   return type: boolean
5. **addToCart():** insert the product in cart.
   parameter: Customer customer, Product product, int quantity
   return type: boolean
6. **removeFromCart():** delete the product in cart.
   parameter: Customer customer, Product product
   return type: boolean
7. **getAllFromCart(Customer customer):** list the product in cart for a customer.
   parameter: Customer customer
   return type: list of product
8. **placeOrder(Customer customer, List<Map<Product,quantity>>, string shippingAddress):** should update order table and orderItems table.
   1. parameter: Customer customer, list of product and quantity
   2. return type: boolean
9. **getOrdersByCustomer()**
   1. parameter: customerid
   2. return type: list of product and quantity

7. Implement the above interface in a class called **OrderProcessorRepositoryImpl in package dao**.

Connect your application to the SQL database:

8. Write code to establish a connection to your SQL database.
   - Create a utility class **DBConnection** in a package **util** with a static variable **connection** of Type **Connection** and a static method **getConnection()** which returns connection.
   - Connection properties supplied in the connection string should be read from a property file.
   - Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.
9. Create the exceptions in package **myexceptions** and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,
   - **CustomerNotFoundException**: throw this exception when user enters an invalid customer id which doesn't exist in db
   - **ProductNotFoundException**: throw this exception when user enters an invalid product id which doesn't exist in db
   - **OrderNotFoundException**: throw this exception when user enters an invalid order id which doesn't exist in db

10. Create class named **EcomApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.
    1. Register Customer.
    2. Create Product.
    3. Delete Product.
    4. Add to cart.
    5. View cart.
    6. Place order.
    7. View Customer Order

Unit Testing

11. Create Unit test cases for **Ecommerce System** are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:
    - Write test case to test Product created successfully or not.
    - Write test case to test product is added to cart successfully or not.
    - Write test case to test product is ordered successfully or not.
    - write test case to test exception is thrown correctly or not when customer id or product id not found in database.