Case Study on Finance Management System

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 **Finace management** 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. **User Authentication:** Users can log in to their accounts securely.
2. **Expense Management:** Users can add, view, update, and delete their expenses.
3. **Expense Categorization:** Expenses can be categorized into different types (e.g., food, transportation, utilities).
4. **Reports Generation:** Users can generate reports for their expenses over specific time periods.
5. **Database Connectivity:** Data will be stored in a relational database to ensure persistence.

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

Schema Design:

Users:

- user_id (Primary Key)
- username
- password
- email

Expenses:

- expense_id (Primary Key)
- user_id (Foreign Key referencing Users table)
- amount
- category_id (Foreign Key referencing ExpenseCategories table)
- date
- description

ExpenseCategories:

- category_id (Primary Key)
- category_name

Explanation:

- The **Users** table stores information about registered users of the system.
- The **Expenses** table contains details of expenses, including the user who incurred the expense, the amount, category, date, and optional description.
- The **ExpenseCategories** table stores different categories that can be assigned to expenses.

Foreign Key Constraints:

- The **user_id** column in the **Expenses** table is a foreign key referencing the **user_id** column in the **Users** table, establishing a relationship between expenses and users.
- The **category_id** column in the **Expenses** table is a foreign key referencing the **category_id** column in the **ExpenseCategories** table, linking expenses to their respective categories.

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

2. **Service Provider Interface/Abstract class:**
   Keep the interfaces and implementation classes in package dao
   - Define **IFinaceRepository** interface/abstract class with methods for adding/removing products to/from the cart and placing orders. The following methods will interact with database.
     1. **createUser()**
        parameter: User user
        return type: boolean
     2. **createExpense()**
        parameter: Expense expense
        return type: boolean
     3. **deleteUser()**
        parameter: userId
        return type: boolean
     4. **deleteExpense(expenseId)**

parameter: expenseId
return type: boolean

5. **getAllExpenses(userId):** list all expnses a user.
   parameter: userId
   return type: list of expenes
6. **updateexpense(userId, Expense):** should update order table and orderItems table.
   1. parameter: userId, expense object
   2. return type: boolean

3. Implement the above interface in a class called **FinanceRepositoryImpl in package dao**.

Connect your application to the SQL database:

4. 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.
5. 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,
   - **UserNotFoundException**: throw this exception when user enters an invalid user id which doesn't exist in db
   - **ExpenseNotFoundException**: throw this exception when user enters an invalid product id which doesn't exist in db
6. Create class named **FinanceApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.
   1. Add User.
   2. Add expense.
   3. Delete User.
   4. Delete expense.
   5. Update expense
   6.

Unit Testing

7. Create Unit test cases for **Finance 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 if user created successfully
   - Write test case to test if expene is created successfully.
   - Write test case to test search of expense
   - write test case to test if the exceptions are thrown correctly based on scenario