Case Study on Digital Asset Management 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 **Digital Asset Management System** 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:

- Asset management
- Asset tracking
- Asset maintenance

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

Schema Design:

1. **employees** table:
    - employee_id (Primary Key)
    - name
    - department.
    - email.

- password.

2. **assets** table:
    - asset_id (Primary Key): Unique identifier for each asset.
    - name.
    - type: Type of the asset (e.g., laptop, vehicle, equipment).
    - serial_number: Serial number or unique identifier of the asset.
    - purchase_date.
    - location: Current location of the asset.
    - status: Status of the asset (e.g., in use, decommissioned, under maintenance).
    - owner_id: (Foreign Key): References the employee who owns the asset.
3. **maintenance_records** table:
    - maintenance_id (Primary Key): Unique identifier for each maintenance record.
    - asset_id (Foreign Key): References the asset for which maintenance was performed.
    - maintenance_date.
    - description: Description of the maintenance activity.
    - cost: Cost associated with the maintenance.
4. **asset_allocations** table:
    - allocation_id (Primary Key): Unique identifier for each asset allocation.
    - asset_id (Foreign Key): References the asset that is allocated.
    - employee_id (Foreign Key): References the employee to whom the asset is allocated.
    - allocation_date: Date when the asset was allocated.
    - return_date: Date when the asset was returned (if applicable).
5. **reservations** table (to store order details):
    - reservation_id (Primary Key): Unique identifier for each reservation.
    - asset_id (Foreign Key): References the asset that is being reserved.
    - employee_id (Foreign Key): References the employee who made the reservation.
    - reservation_date: Date when the reservation was made.
    - start_date: Date when the reserved asset is needed.
    - end_date: Date when the reservation ends.
    - status: Status of the reservation (e.g., pending, approved, canceled).

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

6. **Service Provider Interface/Abstract class:**
   Keep the interfaces and implementation classes in package dao
    - Define an **AssetManagementService** interface/abstract class with methods for adding/removing asset and its management. The following methods will interact with database.

    a. **Add Asset:**
        i. **Method: boolean addAsset(Asset asset)**
        ii. Description: Adds a new asset to the system.

    b. **Update Asset:**
        i. **Method: boolean updateAsset(Asset asset)**

ii. Description: Updates information about an existing asset.

c. **Delete Asset:**
   i. **Method: boolean deleteAsset(int assetId)**
   ii. Description: Deletes an asset from the system based on its ID.

d. **Allocate Asset:**
   i. **Method: boolean allocateAsset(int assetId, int employeeId, String allocationDate)**
   ii. Description: Allocates an asset to an employee on a specified allocation date.

e. **Deallocate Asset:**
   i. **Method: boolean deallocateAsset(int assetId, int employeeId, String returnDate)**
   ii. Description: Deallocates an asset from an employee on a specified return date.

f. **Perform Maintenance:**
   i. **Method: boolean performMaintenance(int assetId, String maintenanceDate, String description, double cost)**
   ii. Description: Records maintenance activity for an asset, including the date, description, and cost.

g. **Reserve Asset:**
   i. Method: boolean reserveAsset(int assetId, int employeeId, String reservationDate, String startDate, String endDate)
   ii. Description: Reserves an asset for a specified employee for a specific period, starting from the start date to the end date. The reservation is made on the reservation date.

h. **Withdraw Reservation:**
   i. Method: boolean withdrawReservation(int reservationId)
   ii. Description: Withdraws a reservation for an asset identified by the reservation ID. The reserved asset becomes available for allocation again.

7. Implement the above interface in a class called **AssetManagementServiceImpl 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.
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,
   - **AssetNotFoundException**: throw this exception when employee enters an invalid asset id which doesn't exist in db
   - **AssetNotMaintainException**: throw this exception when employee need the asset which is not maintained for 2 years.
10. Create class named **AssetManagementApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.
    - Add Asset:

- Update Asset:
- Delete Asset:
- Allocate Asset:
- Deallocate Asset:
- Perform Maintenance:
- Reserve Asset:

Unit Testing

11. Create Unit test cases for **Digital Asset Management 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 asset created successfully or not.
    - Write test case to test asset is added to maintenance successfully or not.
    - Write test case to test asset is reserved successfully or not.
    - write test case to test exception is thrown correctly or not when employee id or asset id not found in database.