Case Study on Transport 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 **Transport 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:

- Vehicle Management:
    - Adding, updating, and deleting vehicles from the system.
- Trip Management:
    - Scheduling and canceling trips for vehicles on specified routes.
- Booking Management:
    - Booking and canceling trips for passengers.
- Driver Allocation:
    - Allocating and deallocating drivers for scheduled trips.
- Booking Retrieval:
    - Retrieving bookings made by passengers or associated with specific trips.

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

Schema Design:

1. **Vehicles** table:
    - VehicleID: INT, AUTO_INCREMENT, PRIMARY KEY
    - Model: VARCHAR(255)
    - Capacity: DECIMAL(10, 2)
    - Type: VARCHAR(50) (e.g., Truck, Van, Bus)
    - Status: VARCHAR(50) (e.g., Available, On Trip, Maintenance)
2. **Routes** table:
    - RouteID: INT, AUTO_INCREMENT, PRIMARY KEY
    - StartDestination: VARCHAR(255)
    - EndDestination: VARCHAR(255)
    - Distance: DECIMAL(10, 2)
3. **Trips** table:
    - TripID: INT, AUTO_INCREMENT, PRIMARY KEY
    - VehicleID: INT, FOREIGN KEY REFERENCES Vehicles(VehicleID)
    - RouteID: INT, FOREIGN KEY REFERENCES Routes(RouteID)
    - DepartureDate: DATETIME
    - ArrivalDate: DATETIME
    - Status: VARCHAR(50) (e.g., Scheduled, In Progress, Completed, Cancelled)
    - TripType: VARCHAR(50) DEFAULT 'Freight' (e.g., Freight, Passenger)
    - MaxPassengers: INT
4. **Passengers** table:
    - PassengerID: INT, AUTO_INCREMENT, PRIMARY KEY
    - FirstName: VARCHAR(255)
    - gender: VARCHAR(255)
    - age: INT
    - Email: VARCHAR(255), UNIQUE
    - PhoneNumber: VARCHAR(50)
5. **Bookings** table:
    - BookingID: INT, AUTO_INCREMENT, PRIMARY KEY
    - TripID: INT, FOREIGN KEY REFERENCES Trips (TripID)
    - PassengerID: INT, FOREIGN KEY REFERENCES Passengers (PassengerID)
    - BookingDate: DATETIME
    - Status: VARCHAR (50) (e.g., Confirmed, Cancelled, Completed)

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 **TransportManagementService** interface/abstract class with methods for adding/removing asset and its management. The following methods will interact with database.

a. addVehicle:

Method: boolean addVehicle(Vehicle vehicle)

Description: Adds a new vehicle to the system.

b. updateVehicle:

Method: boolean updateVehicle(Vehicle vehicle)

Description: Updates information about an existing vehicle.

c. deleteVehicle:

Method: boolean deleteVehicle(int vehicleId)

Description: Deletes a vehicle from the system based on its ID.

d. scheduleTrip:

Method: boolean scheduleTrip(int vehicleId, int routeId, String departureDate, String arrivalDate)

Description: Schedules a trip for a vehicle on a specified route with departure and arrival dates.

e. cancelTrip:

Method: boolean cancelTrip(int tripId)

Description: Cancels a scheduled trip identified by its ID.

f. bookTrip:

Method: boolean bookTrip(int tripId, int passengerId, String bookingDate)

Description: Books a trip for a passenger on a specified trip with the booking date.

g. cancelBooking:

Method: boolean cancelBooking(int bookingId)

Description: Cancels a booking for a trip identified by its booking ID.

h. allocateDriver:

Method: boolean allocateDriver(int tripId, int driverId)

Description: Allocates a driver to a scheduled trip identified by its ID.

i. deallocateDriver:

Method: boolean deallocateDriver(int tripId)

Description: Deallocates a driver from a scheduled trip identified by its ID.

j. getBookingsByPassenger:

Method: List<Booking> getBookingsByPassenger(int passengerId)

Description: Retrieves a list of bookings made by a specific passenger identified by their ID.

k. getBookingsByTrip:

Method: List<Booking> getBookingsByTrip(int tripId)

Description: Retrieves a list of bookings associated with a specific trip identified by its ID.

l. getAvailableDrivers:

Method: List<Driver> getAvailableDrivers()

Description: Retrieves a list of available drivers who are not currently allocated to any trip.

- Implement the above interface in a class called **TransportManagementServiceImpl in package dao**.

Connect your application to the SQL database:

7. 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.
8. 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,
    - **VechileNotFoundException**: throw this exception when employee enters an invalid asset id which doesn't exist in db
    - **BookingNotFoundException**: throw this exception when passenger need to retrieve.
9. Create class named **TransportManagementApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.
    - addVehicle:
    - updateVehicle:
    - deleteVehicle:
    - scheduleTrip:
    - cancelTrip:
    - bookTrip:
    - cancelBooking:
    - allocateDriver:
    - deallocateDriver:
    - getBookingsByPassenger:
    - getBookingsByTrip:
    - getAvailableDrivers:

Unit Testing

10. Create Unit test cases for **Transport 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 driver mapped to vehicle successfully or not.
    - Write test case to test vehicle, driver successfully or not.
    - Write test case to test booking successfully or not.
    - write test case to test exception is thrown correctly or not when vehicle or booking not found in database.