CAR CONNECT

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

Creating a database

Create database carconnect;

```
mysql> create database carconnect;
Query OK, 1 row affected (0.04 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| carconnect         |
| carrental          |
| information_schema |
| mysql              |
| performance_schema |
| practice           |
| sakila             |
| sys                |
| techshop           |
| world              |
+--------------------+
10 rows in set (0.02 sec)
```

SQL Tables:

1. Customer Table: • CustomerID (Primary Key): Unique identifier for each customer. • FirstName: First name of the customer. • LastName: Last name of the customer. • Email: Email address of the customer for communication. • PhoneNumber: Contact number of the customer. • Address: Customer's residential address. • Username: Unique username for customer login. • Password: Securely hashed password for customer authentication. • RegistrationDate: Date when the customer registered.

CREATE TABLE Customer (

CustomerID INT AUTO_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(15),

Address VARCHAR(255),

Username VARCHAR(50) UNIQUE,

Password VARCHAR(255),

RegistrationDate DATE

);

```
mysql> CREATE TABLE Customer (
    ->     CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     Email VARCHAR(100),
    ->     PhoneNumber VARCHAR(15),
    ->     Address VARCHAR(255),
    ->     Username VARCHAR(50) UNIQUE,
    ->     Password VARCHAR(255),
    ->     RegistrationDate DATE
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> desc Customer;
+------------------+--------------+------+-----+---------+----------------+
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| CustomerID       | int          | NO   | PRI | NULL    | auto_increment |
| FirstName        | varchar(50)  | YES  |     | NULL    |                |
| LastName         | varchar(50)  | YES  |     | NULL    |                |
| Email            | varchar(100) | YES  |     | NULL    |                |
| PhoneNumber      | varchar(15)  | YES  |     | NULL    |                |
| Address          | varchar(255) | YES  |     | NULL    |                |
| Username         | varchar(50)  | YES  | UNI | NULL    |                |
| Password         | varchar(255) | YES  |     | NULL    |                |
| RegistrationDate | date         | YES  |     | NULL    |                |
+------------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

2. Vehicle Table: • VehicleID (Primary Key): Unique identifier for each vehicle. • Model: Model of the vehicle. • Make: Manufacturer or brand of the vehicle. • Year: Manufacturing year of the vehicle. • Color: Color of the vehicle. • RegistrationNumber: Unique registration number for each vehicle. • Availability: Boolean indicating whether the vehicle is available for rent. • DailyRate: Daily rental rate for the vehicle.

CREATE TABLE Vehicle (

VehicleID INT AUTO_INCREMENT PRIMARY KEY,

Model VARCHAR(50),

Make VARCHAR(50),

Year INT,

Color VARCHAR(50),

RegistrationNumber VARCHAR(20) UNIQUE,

Availability BOOLEAN,

DailyRate DECIMAL(10, 2)

);

```
mysql> CREATE TABLE Vehicle (
    ->     VehicleID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Model VARCHAR(50),
    ->     Make VARCHAR(50),
    ->     Year INT,
    ->     Color VARCHAR(50),
    ->     RegistrationNumber VARCHAR(20) UNIQUE,
    ->     Availability BOOLEAN,
    ->     DailyRate DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> desc vehicle;
+--------------------+---------------+------+-----+---------+----------------+
| Field              | Type          | Null | Key | Default | Extra          |
+--------------------+---------------+------+-----+---------+----------------+
| VehicleID          | int           | NO   | PRI | NULL    | auto_increment |
| Model              | varchar(50)   | YES  |     | NULL    |                |
| Make               | varchar(50)   | YES  |     | NULL    |                |
| Year               | int           | YES  |     | NULL    |                |
| Color              | varchar(50)   | YES  |     | NULL    |                |
| RegistrationNumber | varchar(20)   | YES  | UNI | NULL    |                |
| Availability       | tinyint(1)    | YES  |     | NULL    |                |
| DailyRate          | decimal(10,2) | YES  |     | NULL    |                |
+--------------------+---------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

3. Reservation Table: • ReservationID (Primary Key): Unique identifier for each reservation. • CustomerID (Foreign Key): Foreign key referencing the Customer table. • VehicleID (Foreign Key): Foreign key referencing the Vehicle table. • StartDate: Date and time of the reservation start. • EndDate: Date and time of the reservation end. • TotalCost: Total cost of the reservation. • Status: Current status of the reservation (e.g., pending, confirmed, completed).

CREATE TABLE Reservation (

ReservationID INT AUTO_INCREMENT PRIMARY KEY,

CustomerID INT,

VehicleID INT,

StartDate DATETIME,

EndDate DATETIME,

TotalCost DECIMAL(10, 2),

Status ENUM('pending', 'confirmed', 'completed'),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)

);

```
mysql> CREATE TABLE Reservation (
    ->     ReservationID INT AUTO_INCREMENT PRIMARY KEY,
    ->     CustomerID INT,
    ->     VehicleID INT,
    ->     StartDate DATETIME,
    ->     EndDate DATETIME,
    ->     TotalCost DECIMAL(10, 2),
    ->     Status ENUM('pending', 'confirmed', 'completed'),
    ->     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    ->     FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> desc reservation;
+---------------+-----------------------------------------+------+-----+---------+----------------+
| Field         | Type                                    | Null | Key | Default | Extra          |
+---------------+-----------------------------------------+------+-----+---------+----------------+
| ReservationID | int                                     | NO   | PRI | NULL    | auto_increment |
| CustomerID    | int                                     | YES  | MUL | NULL    |                |
| VehicleID     | int                                     | YES  | MUL | NULL    |                |
| StartDate     | datetime                                | YES  |     | NULL    |                |
| EndDate       | datetime                                | YES  |     | NULL    |                |
| TotalCost     | decimal(10,2)                           | YES  |     | NULL    |                |
| Status        | enum('pending','confirmed','completed') | YES  |     | NULL    |                |
+---------------+-----------------------------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

4. Admin Table: • AdminID (Primary Key): Unique identifier for each admin. • FirstName: First name of the admin. • LastName: Last name of the admin. • Email: Email address of the admin for communication. • PhoneNumber: Contact number of the admin. • Username: Unique username for admin login. • Password: Securely hashed password for admin authentication. • Role: Role of the admin within the system (e.g., super admin, fleet manager). • JoinDate: Date when the admin joined the system.

CREATE TABLE Admin (

AdminID INT AUTO_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(15),

Username VARCHAR(50) UNIQUE,

Password VARCHAR(255),

Role VARCHAR(20),

JoinDate DATE

);

```
mysql> CREATE TABLE Admin (
    ->     AdminID INT AUTO_INCREMENT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     Email VARCHAR(100),
    ->     PhoneNumber VARCHAR(15),
    ->     Username VARCHAR(50) UNIQUE,
    ->     Password VARCHAR(255),
    ->     Role VARCHAR(20),
    ->     JoinDate DATE
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> desc admin;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| AdminID     | int          | NO   | PRI | NULL    | auto_increment |
| FirstName   | varchar(50)  | YES  |     | NULL    |                |
| LastName    | varchar(50)  | YES  |     | NULL    |                |
| Email       | varchar(100) | YES  |     | NULL    |                |
| PhoneNumber | varchar(15)  | YES  |     | NULL    |                |
| Username    | varchar(50)  | YES  | UNI | NULL    |                |
| Password    | varchar(255) | YES  |     | NULL    |                |
| Role        | varchar(20)  | YES  |     | NULL    |                |
| JoinDate    | date         | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

Inserting Data into tables:

Customer Table:

INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate) VALUES

('John', 'Doe', 'john.doe@example.com', '1234567890', '123 Main Street, City, Country', 'johndoe', 'password123', '2024-01-01'),

('Jane', 'Smith', 'jane.smith@example.com', '0987654321', '456 Elm Street, City, Country', 'janesmith', 'letmein', '2024-01-15'),

('Michael', 'Johnson', 'michael.johnson@example.com', '1112223333', '789 Oak Street, City, Country', 'michaelj', 'securepass', '2024-02-10'),

('Emily', 'Brown', 'emily.brown@example.com', '4445556666', '101 Pine Street, City, Country', 'emilyb', 'p@ssw0rd', '2024-03-05'),

('David', 'Wilson', 'david.wilson@example.com', '7778889999', '246 Maple Street, City, Country', 'davidw', 'welcome123', '2024-04-20');

```
mysql> INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate) VALUES
    -> ('John', 'Doe', 'john.doe@example.com', '1234567890', '123 Main Street, City, Country', 'johndoe', 'password123', '2024-01-01'),
    -> ('Jane', 'Smith', 'jane.smith@example.com', '0987654321', '456 Elm Street, City, Country', 'janesmith', 'letmein', '2024-01-15'),
    -> ('Michael', 'Johnson', 'michael.johnson@example.com', '1112223333', '789 Oak Street, City, Country', 'michaelj', 'securepass', '2024-02-10'),
    -> ('Emily', 'Brown', 'emily.brown@example.com', '4445556666', '101 Pine Street, City, Country', 'emilyb', 'p@ssw0rd', '2024-03-05'),
    -> ('David', 'Wilson', 'david.wilson@example.com', '7778889999', '246 Maple Street, City, Country', 'davidw', 'welcome123', '2024-04-20');
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from customer;
+------------+-----------+----------+--------------------------+-------------+-----------------------------+----------+-------------+------------------+
| CustomerID | FirstName | LastName | Email                    | PhoneNumber | Address                     | Username | Password    | RegistrationDate |
+------------+-----------+----------+--------------------------+-------------+-----------------------------+----------+-------------+------------------+
|          1 | John      | Doe      | john.doe@example.com     | 1234567890  | 123 Main Street, City, Country | johndoe  | password123 | 2024-01-01       |
|          2 | Jane      | Smith    | jane.smith@example.com   | 0987654321  | 456 Elm Street, City, Country  | janesmith | letmein     | 2024-01-15       |
|          3 | Michael   | Johnson  | michael.johnson@example.com | 1112223333 | 789 Oak Street, City, Country  | michaelj | securepass  | 2024-02-10       |
|          4 | Emily     | Brown    | emily.brown@example.com  | 4445556666  | 101 Pine Street, City, Country | emilyb   | p@ssw0rd    | 2024-03-05       |
|          5 | David     | Wilson   | david.wilson@example.com | 7778889999  | 246 Maple Street, City, Country | davidw   | welcome123  | 2024-04-20       |
+------------+-----------+----------+--------------------------+-------------+-----------------------------+----------+-------------+------------------+
5 rows in set (0.00 sec)
```

Vehicle Table:

INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)

VALUES

('Toyota Camry', 'Toyota', 2020, 'Silver', 'ABC123', TRUE, 50.00),

('Honda Civic', 'Honda', 2019, 'Black', 'XYZ456', TRUE, 45.00),

('Ford Mustang', 'Ford', 2018, 'Red', 'DEF789', TRUE, 60.00),

('Chevrolet Cruze', 'Chevrolet', 2017, 'Blue', 'GHI012', TRUE, 55.00),

('BMW X5', 'BMW', 2021, 'White', 'JKL345', TRUE, 70.00);

```
mysql> INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
    -> VALUES
    -> ('Toyota Camry', 'Toyota', 2020, 'Silver', 'ABC123', TRUE, 50.00),
    -> ('Honda Civic', 'Honda', 2019, 'Black', 'XYZ456', TRUE, 45.00),
    -> ('Ford Mustang', 'Ford', 2018, 'Red', 'DEF789', TRUE, 60.00),
    -> ('Chevrolet Cruze', 'Chevrolet', 2017, 'Blue', 'GHI012', TRUE, 55.00),
    -> ('BMW X5', 'BMW', 2021, 'White', 'JKL345', TRUE, 70.00);
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from vehicle;
+-----------+-----------------+-----------+------+--------+--------------------+--------------+-----------+
| VehicleID | Model           | Make      | Year | Color  | RegistrationNumber | Availability | DailyRate |
+-----------+-----------------+-----------+------+--------+--------------------+--------------+-----------+
|         1 | Toyota Camry    | Toyota    | 2020 | Silver | ABC123             |            1 |     50.00 |
|         2 | Honda Civic     | Honda     | 2019 | Black  | XYZ456             |            1 |     45.00 |
|         3 | Ford Mustang    | Ford      | 2018 | Red    | DEF789             |            1 |     60.00 |
|         4 | Chevrolet Cruze | Chevrolet | 2017 | Blue   | GHI012             |            1 |     55.00 |
|         5 | BMW X5          | BMW       | 2021 | White  | JKL345             |            1 |     70.00 |
+-----------+-----------------+-----------+------+--------+--------------------+--------------+-----------+
5 rows in set (0.00 sec)
```

Reservation Table:

INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)

VALUES

(1, 1, '2024-05-05 10:00:00', '2024-05-07 18:00:00', 100.00, 'confirmed'),

(2, 2, '2024-05-10 09:00:00', '2024-05-12 15:00:00', 90.00, 'pending'),

(3, 3, '2024-05-15 14:00:00', '2024-05-18 12:00:00', 180.00, 'confirmed'),

(4, 4, '2024-05-20 08:00:00', '2024-05-22 20:00:00', 110.00, 'pending'),

(5, 5, '2024-05-25 11:00:00', '2024-05-28 10:00:00', 210.00, 'confirmed');

```
mysql> INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
    -> VALUES
    -> (1, 1, '2024-05-05 10:00:00', '2024-05-07 18:00:00', 100.00, 'confirmed'),
    -> (2, 2, '2024-05-10 09:00:00', '2024-05-12 15:00:00', 90.00, 'pending'),
    -> (3, 3, '2024-05-15 14:00:00', '2024-05-18 12:00:00', 180.00, 'confirmed'),
    -> (4, 4, '2024-05-20 08:00:00', '2024-05-22 20:00:00', 110.00, 'pending'),
    -> (5, 5, '2024-05-25 11:00:00', '2024-05-28 10:00:00', 210.00, 'confirmed');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from reservation;
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
| ReservationID | CustomerID | VehicleID | StartDate           | EndDate             | TotalCost | Status    |
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
|             2 |          1 |         1 | 2024-05-05 10:00:00 | 2024-05-07 18:00:00 |    100.00 | confirmed |
|             3 |          2 |         2 | 2024-05-10 09:00:00 | 2024-05-12 15:00:00 |     90.00 | pending   |
|             4 |          3 |         3 | 2024-05-15 14:00:00 | 2024-05-18 12:00:00 |    180.00 | confirmed |
|             5 |          4 |         4 | 2024-05-20 08:00:00 | 2024-05-22 20:00:00 |    110.00 | pending   |
|             6 |          5 |         5 | 2024-05-25 11:00:00 | 2024-05-28 10:00:00 |    210.00 | confirmed |
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
5 rows in set (0.00 sec)
```

Admin Table:

INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate) VALUES

('Admin', 'Smith', 'admin@example.com', '1112223333', 'admin', 'adminpass', 'SuperAdmin', '2024-01-01'),

('Alice', 'Johnson', 'alice.johnson@example.com', '9998887777', 'alicej', 'admin123', 'Admin', '2024-02-15'),

('Bob', 'Davis', 'bob.davis@example.com', '3334445555', 'bobd', 'password123', 'Admin', '2024-03-20'),

('Ella', 'Martinez', 'ella.martinez@example.com', '6665554444', 'ellam', 'securepass', 'Admin', '2024-04-10'),

('Chris', 'Wilson', 'chris.wilson@example.com', '2223334444', 'chrisw', 'letmein', 'Admin', '2024-05-05');

```
mysql> INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate) VALUES
    -> ('Admin', 'Smith', 'admin@example.com', '1112223333', 'admin', 'adminpass', 'SuperAdmin', '2024-01-01'),
    -> ('Alice', 'Johnson', 'alice.johnson@example.com', '9998887777', 'alicej', 'admin123', 'Admin', '2024-02-15'),
    -> ('Bob', 'Davis', 'bob.davis@example.com', '3334445555', 'bobd', 'password123', 'Admin', '2024-03-20'),
    -> ('Ella', 'Martinez', 'ella.martinez@example.com', '6665554444', 'ellam', 'securepass', 'Admin', '2024-04-10'),
    -> ('Chris', 'Wilson', 'chris.wilson@example.com', '2223334444', 'chrisw', 'letmein', 'Admin', '2024-05-05');
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from admin;
+---------+-----------+----------+--------------------------+-------------+----------+-------------+------------+------------+
| AdminID | FirstName | LastName | Email                    | PhoneNumber | Username | Password    | Role       | JoinDate   |
+---------+-----------+----------+--------------------------+-------------+----------+-------------+------------+------------+
|       1 | Admin     | Smith    | admin@example.com        | 1112223333  | admin    | adminpass   | SuperAdmin | 2024-01-01 |
|       2 | Alice     | Johnson  | alice.johnson@example.com | 9998887777  | alicej   | admin123    | Admin      | 2024-02-15 |
|       3 | Bob       | Davis    | bob.davis@example.com    | 3334445555  | bobd     | password123 | Admin      | 2024-03-20 |
|       4 | Ella      | Martinez | ella.martinez@example.com | 6665554444  | ellam    | securepass  | Admin      | 2024-04-10 |
|       5 | Chris     | Wilson   | chris.wilson@example.com | 2223334444  | chrisw   | letmein     | Admin      | 2024-05-05 |
+---------+-----------+----------+--------------------------+-------------+----------+-------------+------------+------------+
5 rows in set (0.00 sec)
```

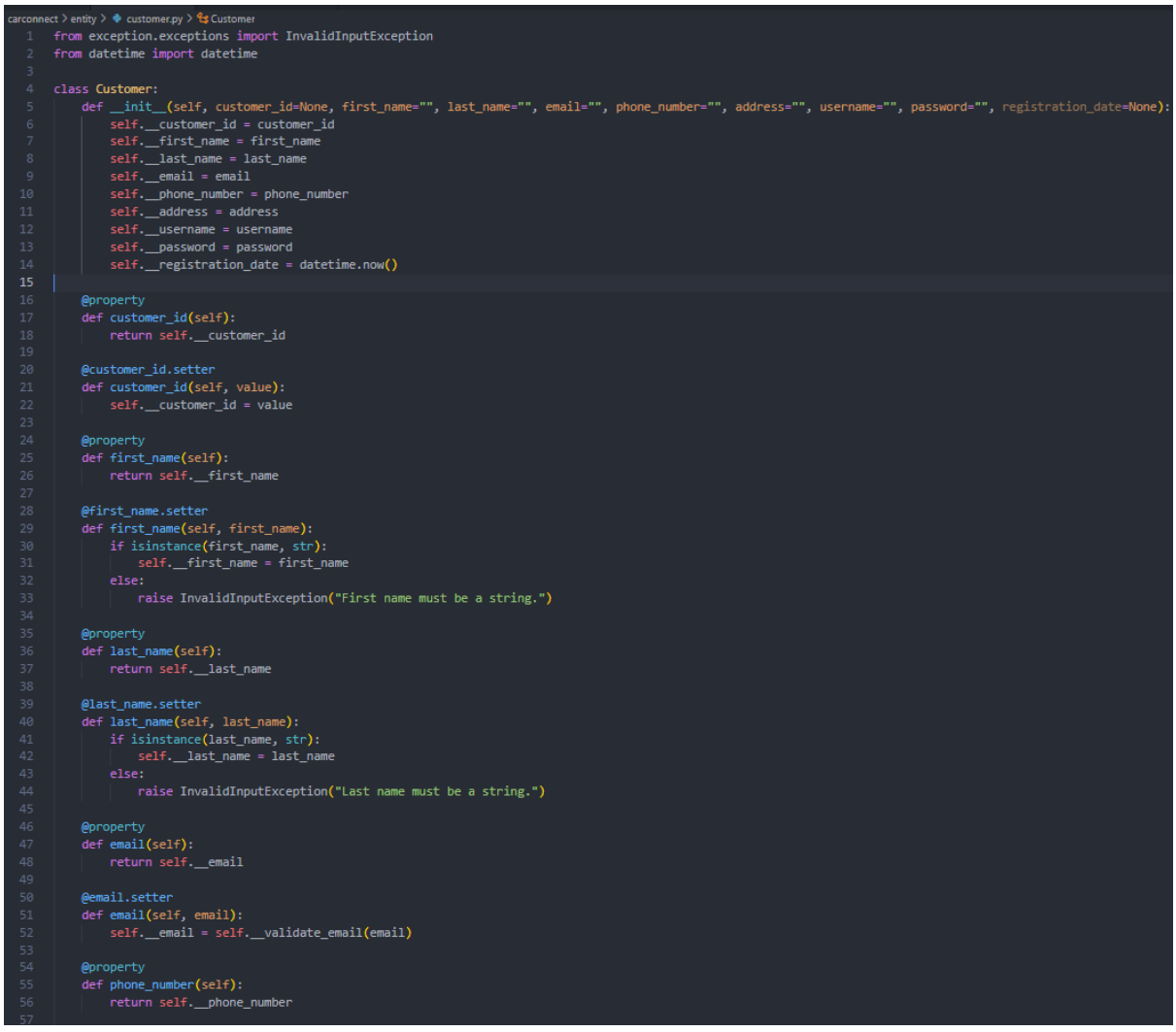
Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Classes:

Customer:

Properties: CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate

Methods: Authenticate(password)



```
carconnect > entity > customer.py > Customer
class Customer:

    @phone_number.setter
    def phone_number(self, phone):
        if isinstance(phone, str) and phone.isdigit():
            self.__phone_number = phone
        else:
            raise InvalidInputException("Phone number must be a string containing only digits.")

    @property
    def address(self):
        return self.__address

    @address.setter
    def address(self, address):
        if isinstance(address, str):
            self.__address = address
        else:
            raise InvalidInputException("Address must be a string.")

    @property
    def username(self):
        return self.__username

    @username.setter
    def username(self, value):
        self.__username = value

    @property
    def password(self):
        return self.__password

    @password.setter
    def password(self, value):
        self.__password = value

    @property
    def registration_date(self):
        return self.__registration_date

    @registration_date.setter
    def registration_date(self, value):
        self.__registration_date = value

    def authenticate(self, password):
        return self.__password == password
```

Vehicle:

Properties: VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate

```
carconnect > entity > vehicle.py > Vehicle
class Vehicle:
    def __init__(self, vehicle_id=None, model=None, make=None, year=None, color=None, registration_number=None, availability=None, daily_rate=None):
        self.__vehicle_id = vehicle_id
        self.__model = model
        self.__make = make
        self.__year = year
        self.__color = color
        self.__registration_number = registration_number
        self.__availability = availability
        self.__daily_rate = daily_rate

    @property
    def vehicle_id(self):
        return self.__vehicle_id
```

```python
    @vehicle_id.setter
    def vehicle_id(self, value):
        self.__vehicle_id = value

    @property
    def model(self):
        return self.__model

    @model.setter
    def model(self, value):
        self.__model = value

    @property
    def make(self):
        return self.__make

    @make.setter
    def make(self, value):
        self.__make = value

    @property
    def year(self):
        return self.__year

    @year.setter
    def year(self, value):
        self.__year = value

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, value):
        self.__color = value

    @property
    def registration_number(self):
        return self.__registration_number
```

```python
    @registration_number.setter
    def registration_number(self, value):
        self.__registration_number = value

    @property
    def availability(self):
        return self.__availability

    @availability.setter
    def availability(self, value):
        self.__availability = value

    @property
    def daily_rate(self):
        return self.__daily_rate

    @daily_rate.setter
    def daily_rate(self, value):
        self.__daily_rate = value
```

Reservation:

Properties: ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status

Methods: CalculateTotalCost()

carconnect > entity > reservation.py > Reservation

```python
class Reservation:
    def __init__(self, reservation_id=None, customer_id=None, vehicle_id=None, start_date=None, end_date=None, total_cost=None, status=None):
        self.__reservation_id = reservation_id
        self.__customer_id = customer_id
        self.__vehicle_id = vehicle_id
        self.__start_date = start_date
        self.__end_date = end_date
        self.__total_cost = total_cost
        self.__status = status

    @property
    def reservation_id(self):
        return self.__reservation_id

    @reservation_id.setter
    def reservation_id(self, value):
        self.__reservation_id = value
```

carconnect > entity > reservation.py > Reservation

```python
class Reservation:

    @property
    def customer_id(self):
        return self.__customer_id

    @customer_id.setter
    def customer_id(self, value):
        self.__customer_id = value

    @property
    def vehicle_id(self):
        return self.__vehicle_id

    @vehicle_id.setter
    def vehicle_id(self, value):
        self.__vehicle_id = value

    @property
    def start_date(self):
        return self.__start_date

    @start_date.setter
    def start_date(self, value):
        self.__start_date = value

    @property
    def end_date(self):
        return self.__end_date

    @end_date.setter
    def end_date(self, value):
        self.__end_date = value

    @property
    def total_cost(self):
        return self.__total_cost

    @total_cost.setter
    def total_cost(self, value):
        self.__total_cost = value

    @property
    def status(self):
        return self.__status

    @status.setter
    def status(self, value):
        self.__status = value

    def calculate_total_cost(self):
        # Your calculation logic here
        pass

```

Admin:

Properties: AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate

Methods: Authenticate(password)

carconnect > entity > admin.py > ...

```python
import re
from exception.exceptions import InvalidInputException

class Admin:
    def __init__(self, admin_id=None, first_name=None, last_name=None, email=None, phone_number=None, username=None, password=None, role=None, join_date=None):
        self.__admin_id = admin_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
        self.__phone_number = phone_number
        self.__username = username
        self.__password = password
        self.__role = role
        self.__join_date = join_date

    def __validate_email(self, email):
        if re.match(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', email):
            return email
        else:
            raise InvalidInputException("Invalid email address provided.")

    @property
    def admin_id(self):
        return self.__admin_id

    @admin_id.setter
    def admin_id(self, value):
        self.__admin_id = value

    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, first_name):
        if isinstance(first_name, str):
            self.__first_name = first_name
        else:
            raise InvalidInputException("First name must be a string.")

    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
    def last_name(self, last_name):
        if isinstance(last_name, str):
            self.__last_name = last_name
        else:
            raise InvalidInputException("Last name must be a string.")

    @property
    def email(self):
        return self.__email
```

carconnect > entity > admin.py > ...

```python
class Admin:

    @email.setter
    def email(self, email):
        self.__email = self.__validate_email(email)

    @property
    def phone_number(self):
        return self.__phone_number

    @phone_number.setter
    def phone_number(self, phone):
        if isinstance(phone, str) and phone.isdigit():
            self.__phone_number = phone
        else:
            raise InvalidInputException("Phone number must be a string containing only digits.")

    @property
    def username(self):
        return self.__username
```

carconnect > entity > admin.py > ...

```python
class Admin:

    @username.setter
    def username(self, value):
        self.__username = value

    @property
    def password(self):
        return self.__password

    @password.setter
    def password(self, value):
        self.__password = value

    @property
    def role(self):
        return self.__role

    @role.setter
    def role(self, value):
        self.__role = value

    @property
    def join_date(self):
        return self.__join_date

    @join_date.setter
    def join_date(self, value):
        self.__join_date = value

    def authenticate(self, password):
        return self.__password == password

```

CustomerService (implements ICustomerService):

Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

carconnect > dao > imp > customerService.py > CustomerService > get_customer_by_id

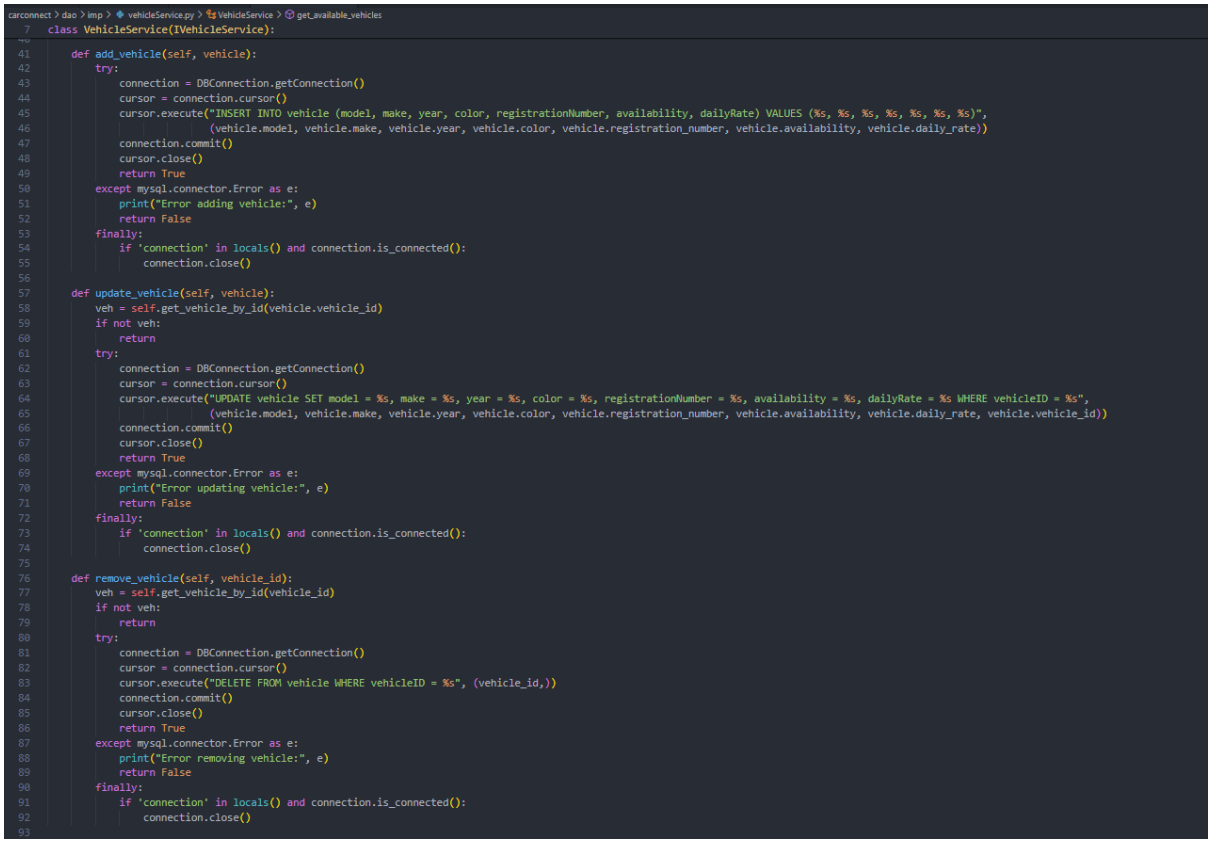
```python
import mysql.connector

from dao.interface.ICustomerService import ICustomerService
from util.db_connection import DBConnection

class CustomerService(ICustomerService):

    def get_customer_by_id(self, customer_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM customer WHERE customerID = %s", (customer_id,))
            customer = cursor.fetchone()
            cursor.close()
            return customer
        except mysql.connector.Error as e:
            print("Error getting customer by id:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def get_customer_by_username(self, username):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM customer WHERE username = %s", (username,))
            customer = cursor.fetchone()
            cursor.close()
            return customer
        except mysql.connector.Error as e:
            print("Error getting customer by username:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

```python
class CustomerService(ICustomerService):

    def register_customer(self, customer):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("INSERT INTO customer (firstname, lastname, email, phoneNumber, address, username, password, registrationDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)",
                           (customer.first_name, customer.last_name, customer.email, customer.phone_number, customer.address, customer.username, customer.password, customer.registration_date))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error registering customer:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def update_customer(self, customer):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("UPDATE customer SET firstname = %s, lastname = %s, email = %s, phoneNumber = %s, address = %s, username = %s, password = %s WHERE customerID = %s",
                           (customer.first_name, customer.last_name, customer.email, customer.phone_number, customer.address, customer.username, customer.password, customer.customer_id))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error updating customer:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def delete_customer(self, customer_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("DELETE FROM customer WHERE customerID = %s", (customer_id,))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error deleting customer:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

VehicleService (implements IVehicleService):

Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

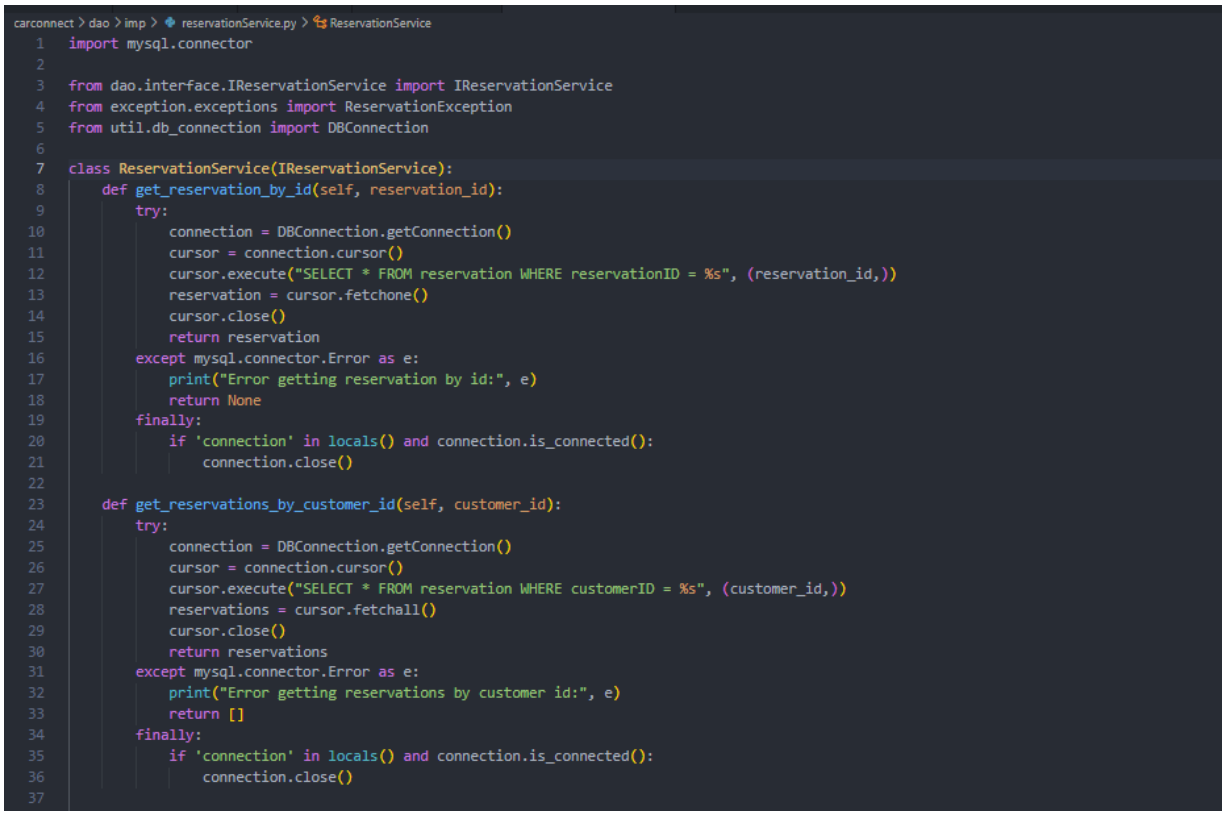
```python
import mysql.connector

from dao.interface.IVehicleService import IVehicleService
from exception.exceptions import VehicleNotFoundException
from util.db_connection import DBConnection

class VehicleService(IVehicleService):

    def get_vehicle_by_id(self, vehicle_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
            vehicle = cursor.fetchone()
            cursor.close()
            if not vehicle:
                raise VehicleNotFoundException("Vehicle not found.")
            return vehicle
        except mysql.connector.Error as e:
            print("Error getting vehicle by id:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def get_available_vehicles(self):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM vehicle WHERE availability = %s", (True,))
            vehicles = cursor.fetchall()
            cursor.close()
            return vehicles
        except mysql.connector.Error as e:
            print("Error getting available vehicles:", e)
            return []
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```



```
carconnect > dao > imp > vehicleService.py > VehicleService > get_available_vehicles
class VehicleService(IVehicleService):

    def add_vehicle(self, vehicle):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("INSERT INTO vehicle (model, make, year, color, registrationNumber, availability, dailyRate) VALUES (%s, %s, %s, %s, %s, %s, %s)",
                           (vehicle.model, vehicle.make, vehicle.year, vehicle.color, vehicle.registration_number, vehicle.availability, vehicle.daily_rate))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error adding vehicle:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def update_vehicle(self, vehicle):
        veh = self.get_vehicle_by_id(vehicle.vehicle_id)
        if not veh:
            return
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("UPDATE vehicle SET model = %s, make = %s, year = %s, color = %s, registrationNumber = %s, availability = %s, dailyRate = %s WHERE vehicleID = %s",
                           (vehicle.model, vehicle.make, vehicle.year, vehicle.color, vehicle.registration_number, vehicle.availability, vehicle.daily_rate, vehicle.vehicle_id))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error updating vehicle:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def remove_vehicle(self, vehicle_id):
        veh = self.get_vehicle_by_id(vehicle_id)
        if not veh:
            return
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("DELETE FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error removing vehicle:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

ReservationService (implements IReservationService):

Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation



```
carconnect > dao > imp > reservationService.py > ReservationService
import mysql.connector

from dao.interface.IReservationService import IReservationService
from exception.exceptions import ReservationException
from util.db_connection import DBConnection

class ReservationService(IReservationService):
    def get_reservation_by_id(self, reservation_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM reservation WHERE reservationID = %s", (reservation_id,))
            reservation = cursor.fetchone()
            cursor.close()
            return reservation
        except mysql.connector.Error as e:
            print("Error getting reservation by id:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def get_reservations_by_customer_id(self, customer_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM reservation WHERE customerID = %s", (customer_id,))
            reservations = cursor.fetchall()
            cursor.close()
            return reservations
        except mysql.connector.Error as e:
            print("Error getting reservations by customer id:", e)
            return []
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

```python
class ReservationService(IReservationService):
    def create_reservation(self, reservation):
        try:
            if self.is_vehicle_booked(reservation.vehicle_id, reservation.start_date, reservation.end_date):
                raise ReservationException("Vehicle is already booked for the specified dates.")

            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("INSERT INTO reservation (customerID, vehicleID, startDate, endDate, totalCost, status) VALUES (%s, %s, %s, %s, %s, %s)",
                           (reservation.customer_id, reservation.vehicle_id, reservation.start_date, reservation.end_date, reservation.total_cost, reservation.status))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error creating reservation:", e)
            return False
        except ReservationException as e:
            print("Reservation error:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def is_vehicle_booked(self, vehicle_id, start_date, end_date):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM reservation WHERE vehicleID = %s AND ((startDate BETWEEN %s AND %s) OR (endDate BETWEEN %s AND %s))",
                           (vehicle_id, start_date, end_date, start_date, end_date))
            existing_reservation = cursor.fetchone()
            return existing_reservation is not None
        except mysql.connector.Error as e:
            print("Error checking existing reservation:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def update_reservation(self, reservation):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("UPDATE reservation SET customerID = %s, vehicleID = %s, startDate = %s, endDate = %s, totalCost = %s, status = %s WHERE reservationID = %s",
                           (reservation.customer_id, reservation.vehicle_id, reservation.start_date, reservation.end_date, reservation.total_cost, reservation.status, reservation.reservation_id))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error updating reservation:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

```python
    def cancel_reservation(self, reservation_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("DELETE FROM reservation WHERE reservationID = %s", (reservation_id,))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error canceling reservation:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

AdminService (implements IAdminService):

Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

```python
import mysql.connector

from dao.interface.IAdminService import IAdminService
from exception.exceptions import AdminNotFoundException
from util.db_connection import DBConnection

class AdminService(IAdminService):
    def get_admin_by_id(self, admin_id):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM admin WHERE adminID = %s", (admin_id,))
            admin = cursor.fetchone()
            cursor.close()
            if not admin:
                raise AdminNotFoundException("Admin not found.")
            return admin
        except mysql.connector.Error as e:
            print("Error getting admin by id:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

carconnect > dao > imp > adminService.py > AdminService

```python
class AdminService(IAdminService):

    def get_admin_by_username(self, username):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM admin WHERE username = %s", (username,))
            admin = cursor.fetchone()
            cursor.close()
            return admin
        except mysql.connector.Error as e:
            print("Error getting admin by username:", e)
            return None
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def register_admin(self, admin):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("INSERT INTO admin (firstName, lastname, email, phoneNumber, username, password, role, joinDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)",
                           (admin.first_name, admin.last_name, admin.email, admin.phone_number, admin.username, admin.password, admin.role, admin.join_date))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error registering admin:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

    def update_admin(self, admin):
        adm = self.get_admin_by_id(admin.admin_id)
        if not adm:
            return
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("UPDATE admin SET firstname = %s, lastname = %s, email = %s, phoneNumber = %s, username = %s, password = %s, role = %s WHERE adminID = %s",
                           (admin.first_name, admin.last_name, admin.email, admin.phone_number, admin.username, admin.password, admin.role, admin.admin_id))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error updating admin:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()
```

```python

    def delete_admin(self, admin_id):
        adm = self.get_admin_by_id(admin_id)
        if not adm:
            return
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            cursor.execute("DELETE FROM admin WHERE adminID = %s", (admin_id,))
            connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print("Error deleting admin:", e)
            return False
        finally:
            if 'connection' in locals() and connection.is_connected():
                connection.close()

```

DatabaseContext: A class responsible for handling database connections and interactions.

AuthenticationService: A class responsible for handling user authentication.

carconnect > dao > imp > authenticationService.py > AuthenticationService > authenticate_admin

```python
from dao.imp.adminService import AdminService
from dao.imp.customerService import CustomerService


class AuthenticationService:
    def __init__(self):
        self.cust_obj = CustomerService()
        self.admin_obj = AdminService()

    def authenticate_customer(self, username, password):
        customer = self.cust_obj.get_customer_by_username(username)
        if customer and customer[7] == password:
            return True
        else:
            return False

    def authenticate_admin(self, username, password):
        admin = self.admin_obj.get_admin_by_username(username)
        if admin and admin[6] == password:
            return True
        else:
            return False


```

ReportGenerator: A class for generating reports based on reservation and vehicle data.

carconnect > dao > imp > reportsGenerator.py > ReportsGenerator > generate_reservation_report > reservation_statuses

```python
import mysql.connector
from util.db_connection import DBConnection

class ReportsGenerator:
    def generate_vehicle_report(self):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()

            query = "SELECT COUNT(*) FROM vehicle WHERE availability = 1"
            cursor.execute(query)
            available_count = cursor.fetchone()[0]
            query = "SELECT COUNT(*) FROM vehicle"
            cursor.execute(query)
            total_count = cursor.fetchone()[0]
            availability_percentage = (available_count / total_count) * 100
            print("Vehicle Availability Analysis:")
            print(f"Total Vehicles: {total_count}")
            print(f"Available Vehicles: {available_count}")
            print(f"Availability Percentage: {availability_percentage:.2f}%")
        except mysql.connector.Error as e:
            print("Error generating vehicle report:", e)
        finally:
            if 'connection' in locals() and connection.is_connected():
                cursor.close()
                connection.close()

    def generate_reservation_report(self):
        try:
            connection = DBConnection.getConnection()
            cursor = connection.cursor()
            query = "SELECT status, COUNT(*) FROM reservation GROUP BY status"
            cursor.execute(query)
            reservation_statuses = cursor.fetchall()
            print("Reservation Status Analysis:")
            for status in reservation_statuses:
                print(f"Status: {status[0]}, Count: {status[1]}")
        except mysql.connector.Error as e:
            print("Error generating reservation report:", e)
        finally:
            if 'connection' in locals() and connection.is_connected():
                cursor.close()
                connection.close()

```

Interfaces:

ICustomerService:

• GetCustomerById(customerId) • GetCustomerByUsername(username) • RegisterCustomer(customerData) • UpdateCustomer(customerData) • DeleteCustomer(customerId)

```python
from abc import ABC, abstractmethod

class ICustomerService(ABC):
    @abstractmethod
    def get_customer_by_id(self, customer_id):
        pass

    @abstractmethod
    def get_customer_by_username(self, username):
        pass

    @abstractmethod
    def register_customer(self, customer_data):
        pass

    @abstractmethod
    def update_customer(self, customer_data):
        pass

    @abstractmethod
    def delete_customer(self, customer_id):
        pass
```

IVehicleService:

• GetVehicleById(vehicleId) • GetAvailableVehicles() • AddVehicle(vehicleData) • UpdateVehicle(vehicleData) • RemoveVehicle(vehicleId)

```python
from abc import ABC, abstractmethod

class IVehicleService(ABC):
    @abstractmethod
    def get_vehicle_by_id(self, vehicle_id):
        pass

    @abstractmethod
    def get_available_vehicles(self):
        pass

    @abstractmethod
    def add_vehicle(self, vehicle_data):
        pass

    @abstractmethod
    def update_vehicle(self, vehicle_data):
        pass

    @abstractmethod
    def remove_vehicle(self, vehicle_id):
        pass
```

IReservationService:

• GetReservationById(reservationId) • GetReservationsByCustomerId(customerId)

• CreateReservation(reservationData) • UpdateReservation(reservationData)

• CancelReservation(reservationId)

```python
from abc import ABC, abstractmethod

class IReservationService(ABC):
    @abstractmethod
    def get_reservation_by_id(self, reservation_id):
        pass

    @abstractmethod
    def get_reservations_by_customer_id(self, customer_id):
        pass

    @abstractmethod
    def create_reservation(self, reservation_data):
        pass

    @abstractmethod
    def update_reservation(self, reservation_data):
        pass

    @abstractmethod
    def cancel_reservation(self, reservation_id):
        pass
```

IAdminService:

• GetAdminById(adminId) • GetAdminByUsername(username) • RegisterAdmin(adminData)

• UpdateAdmin(adminData) • DeleteAdmin(adminId)

carconnect > dao > interface > IAdminService.py > IAdminService

```python
from abc import ABC, abstractmethod


class IAdminService(ABC):
    @abstractmethod
    def get_admin_by_id(self, admin_id):
        pass

    @abstractmethod
    def get_admin_by_username(self, username):
        pass

    @abstractmethod
    def register_admin(self, admin_data):
        pass

    @abstractmethod
    def update_admin(self, admin_data):
        pass

    @abstractmethod
    def delete_admin(self, admin_id):
        pass

```

Connect your application to the SQL database:

• Create a connection string that includes the necessary information to connect to your SQL Server database. This includes the server name, database name, authentication credentials, and any other relevant settings.

• Use the SqlConnection class to establish a connection to the SQL Server database.

• Once the connection is open, you can use the SqlCommand class to execute SQL queries.

carconnect > util > db_connection.py > DBConnection > getConnection

```python
# util/db_connection.py
import mysql.connector
from exception.exceptions import DatabaseConnectionException
from util.property_util import PropertyUtil
import time

class DBConnection:
    __connection = None

    @staticmethod
    def getConnection():
        property_file = r"C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect\util\db.properties"
        connection_string = PropertyUtil().getPropertyString(property_file)
        try:
            DBConnection.__connection = mysql.connector.connect(**connection_string)
            print("Connected to MySQL database successfully.")
            return DBConnection.__connection
        except mysql.connector.Error as err:
            print("Error connecting to MySQL:", err)
            raise DatabaseConnectionException()
            return None

```

Custom Exceptions:

AuthenticationException:

• Thrown when there is an issue with user authentication.

• Example Usage: Incorrect username or password during customer or admin login.

ReservationException:

• Thrown when there is an issue with reservations.

• Example Usage: Attempting to make a reservation for a vehicle that is already reserved.

VehicleNotFoundException:

• Thrown when a requested vehicle is not found.

• Example Usage: Trying to get details of a vehicle that does not exist.

AdminNotFoundException:

• Thrown when an admin user is not found.

• Example Usage: Attempting to access details of an admin that does not exist.

InvalidInputException:

• Thrown when there is invalid input data.

• Example Usage: When a required field is missing or has an incorrect format.

DatabaseConnectionException:

• Thrown when there is an issue with the database connection.

• Example Usage: Unable to establish a connection to the database.

carconnect > exception > exceptions.py > ...

```python
class AuthenticationException(Exception):
    def __init__(self, message="Authentication error"):
        self.message = message
        super().__init__(self.message)


class ReservationException(Exception):
    def __init__(self, message="Reservation error"):
        self.message = message
        super().__init__(self.message)


class VehicleNotFoundException(Exception):
    def __init__(self, message="Vehicle not found"):
        self.message = message
        super().__init__(self.message)


class AdminNotFoundException(Exception):
    def __init__(self, message="Admin not found"):
        self.message = message
        super().__init__(self.message)


class InvalidInputException(Exception):
    def __init__(self, message="Invalid input"):
        self.message = message
        super().__init__(self.message)


class DatabaseConnectionException(Exception):
    def __init__(self, message="Database connection error"):
        self.message = message
        super().__init__(self.message)
```

Unit Testing:

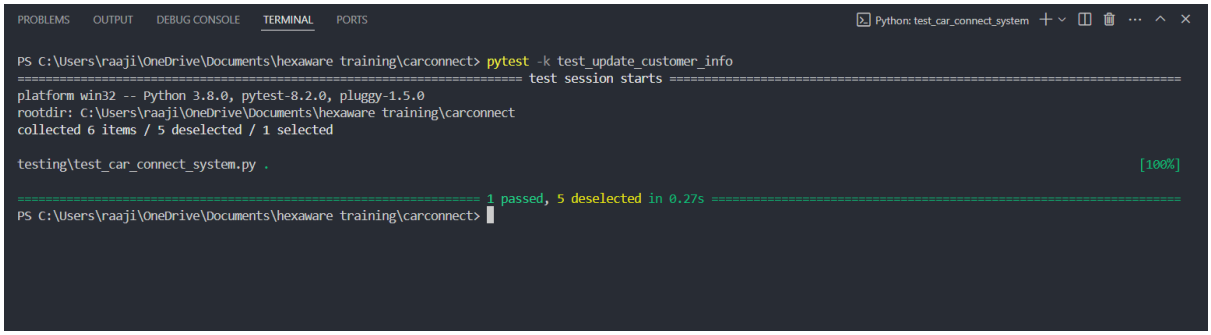
Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:

1. Test customer authentication with invalid credentials.

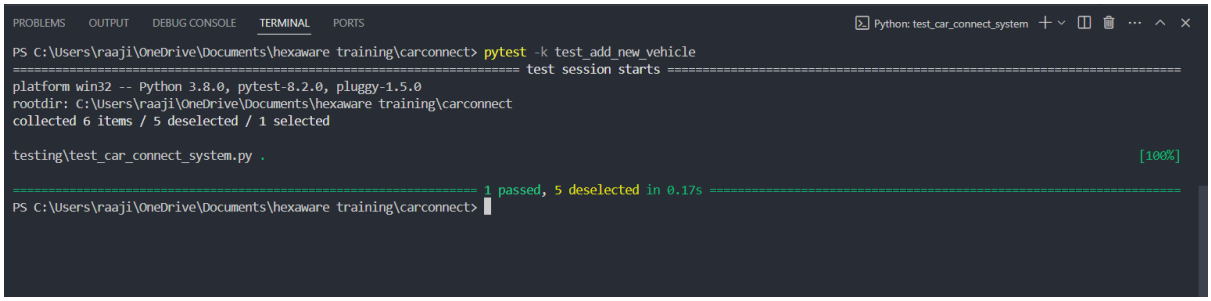
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system

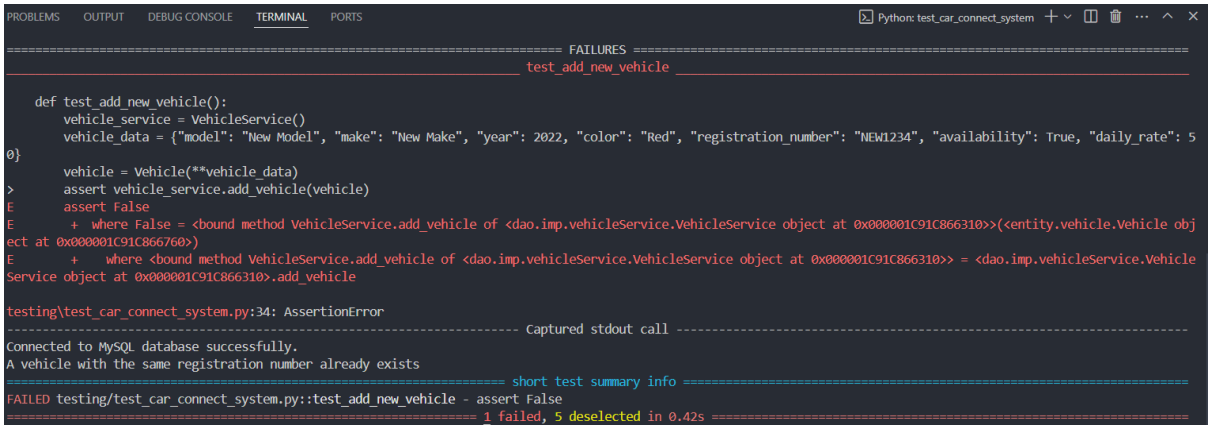
```
_____ test_customer_authentication_invalid _____

    def test_customer_authentication_invalid():
        auth_service = AuthenticationService()
>       assert auth_service.authenticate_customer("invalid_username", "invalid_password")
E       AssertionError: assert False
E        +  where False = <bound method AuthenticationService.authenticate_customer of <dao.imp.authenticationService.AuthenticationService object at 0x0000012D45AFEFD
0>>('invalid_username', 'invalid_password')
E        +    where <bound method AuthenticationService.authenticate_customer of <dao.imp.authenticationService.AuthenticationService object at 0x0000012D45AFEFD0>> =
<dao.imp.authenticationService.AuthenticationService object at 0x0000012D45AFEFD0>.authenticate_customer

testing\test_car_connect_system.py:14: AssertionError
---------- Captured stdout call ----------
Connected to MySQL database successfully.
Customer not found with username invalid_username
Customer Authentication Unsuccessful.
 Incorrect username or password
========== short test summary info ==========
FAILED testing/test_car_connect_system.py::test_customer_authentication_invalid - AssertionError: assert False
========== 1 failed, 5 deselected in 0.32s ==========
```

2. Test updating customer information.

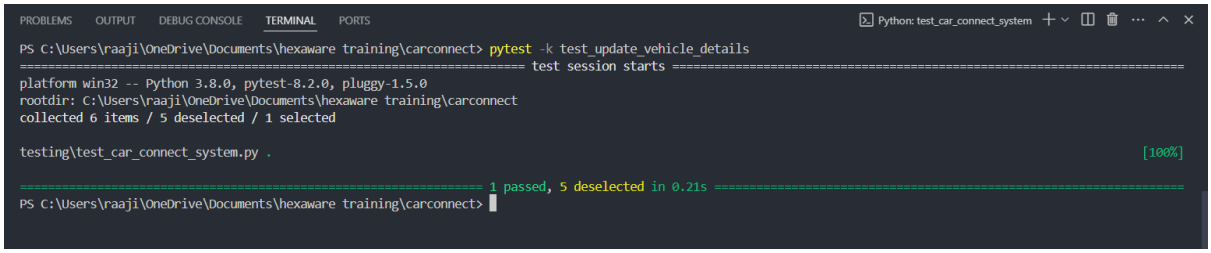


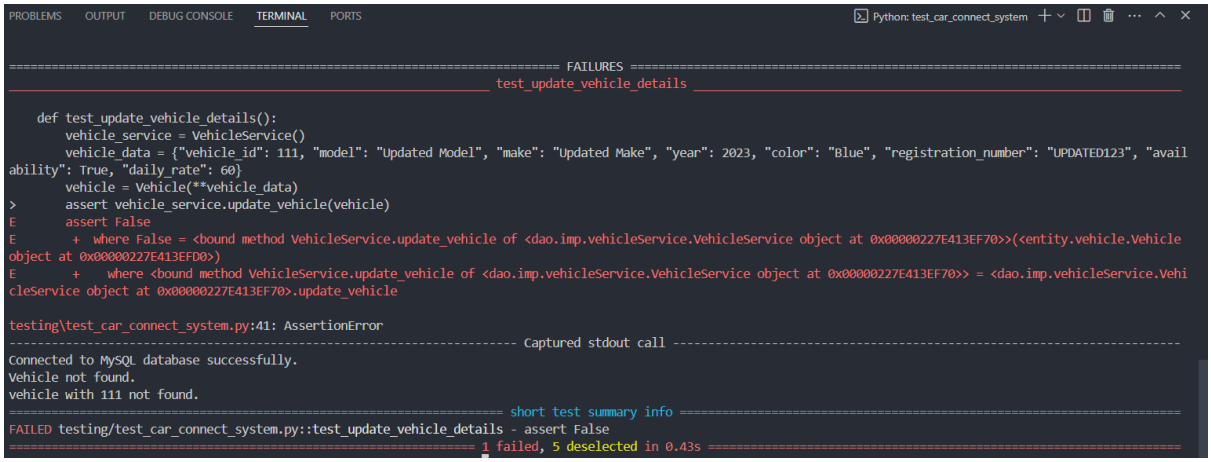
3. Test adding a new vehicle.

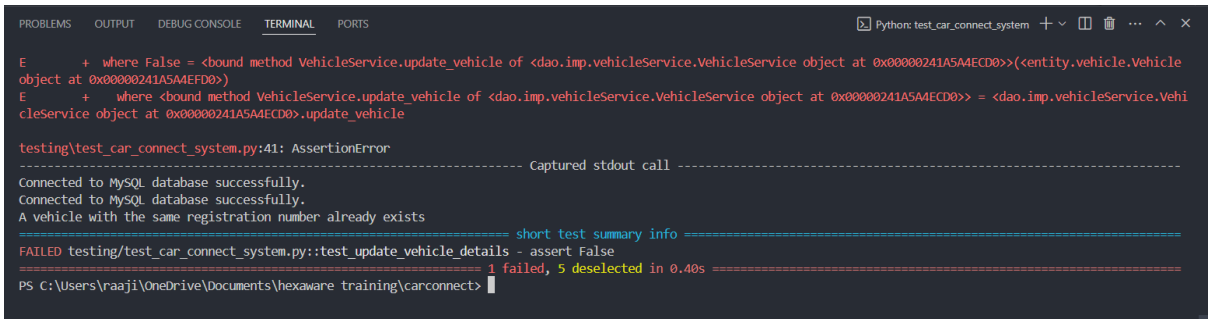




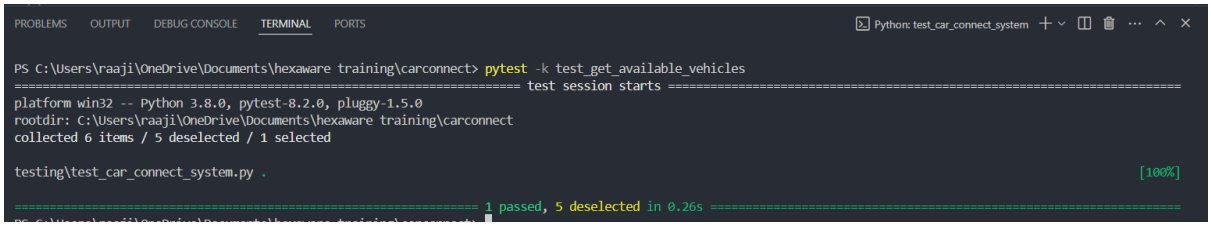
4. Test updating vehicle details.



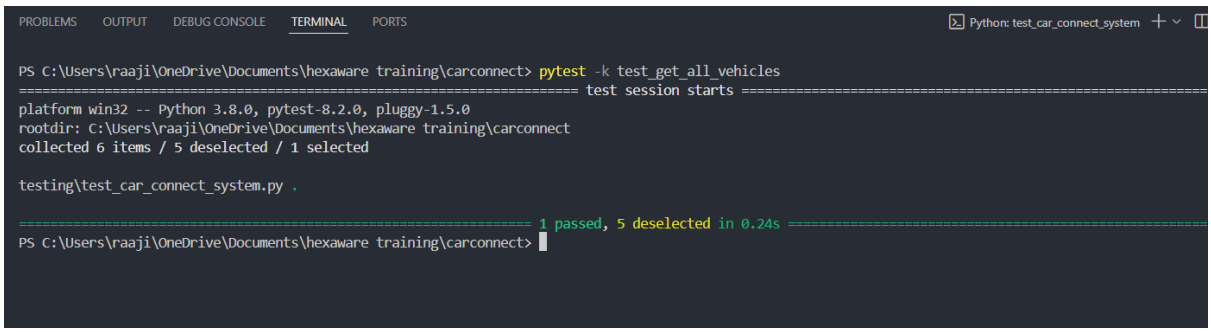




5. Test getting a list of available vehicles.



6. Test getting a list of all vehicles.



carconnect > testing > test_car_connect_system.py > test_update_customer_info

```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))

import pytest
from entity.customer import Customer
from entity.vehicle import Vehicle
from dao.imp.authenticationService import AuthenticationService
from dao.imp.customerService import CustomerService
from dao.imp.vehicleService import VehicleService

def test_customer_authentication_invalid():
    auth_service = AuthenticationService()
    assert not auth_service.authenticate_customer("invalid_username", "invalid_password")

def test_update_customer_info():
    customer_service = CustomerService()
    customer_data = Customer(
        customer_id= 1,
        first_name= "New First Name",
        last_name= "New Last Name",
        email= "new_email@example.com",
        phone_number= "1234567890",
        address= "New Address",
        username= "new_username",
        password= "new_password"
    )
    assert customer_service.update_customer(customer_data)

def test_add_new_vehicle():
    vehicle_service = VehicleService()
    vehicle_data = {"model": "New Model", "make": "New Make", "year": 2022, "color": "Red", "registration_number": "NEW1234", "availability": True, "daily_rate": 50}
    vehicle = Vehicle(**vehicle_data)
    assert vehicle_service.add_vehicle(vehicle)


def test_update_vehicle_details():
    vehicle_service = VehicleService()
    vehicle_data = {"vehicle_id": 1, "model": "Updated Model", "make": "Updated Make", "year": 2023, "color": "Blue", "registration_number": "UPDATED123", "availability": True, "daily_rate": 60}
    vehicle = Vehicle(**vehicle_data)
    assert vehicle_service.update_vehicle(vehicle)


def test_get_available_vehicles():
    vehicle_service = VehicleService()
    available_vehicles = vehicle_service.get_available_vehicles()
    assert available_vehicles is not None


def test_get_all_vehicles():
    vehicle_service = VehicleService()
    all_vehicles = vehicle_service.get_all_vehicles()
    assert all_vehicles is not None
```

Customer Operations:

Get customer by Id:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> & C:/Users/raaji/AppData/Local/Programs/Python/Python38/python.exe "c:/Users/raaji/
OneDrive/Documents/hexaware training/carconnect/main/main.py"

Menu:
1. Customer Operations
2. Vehicle Operations
3. Reservation Operations
4. Admin Operations
5. Exit
Enter your choice: 1

Customer Operations:
1. Get customer by ID
2. Get customer by Username
3. Register customer
4. Update customer
5. Delete customer
Enter your choice: 1
Enter customer ID: 2
Connected to MySQL database successfully.
Customer details: (2, 'Jane', 'Smith', 'jane.smith@example.com', '0987654321', '456 Elm Street, City, Country', 'janesmith', 'letmein', datetime.date(
2024, 1, 15))
```

Get customer by username:

```
Menu:
1. Customer Operations
2. Vehicle Operations
3. Reservation Operations
4. Admin Operations
5. Exit
Enter your choice: 1

Customer Operations:
1. Get customer by ID
2. Get customer by Username
3. Register customer
4. Update customer
5. Delete customer
Enter your choice: 2
Enter customer username: johndoe
Connected to MySQL database successfully.
Customer details: (1, 'John', 'Doe', 'john.doe@example.com', '1234567890', '123 Main Street, City, Country', 'johndoe', 'password123', datetime.date(2
024, 1, 1))
```

Register customer:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter your choice: 1

Customer Operations:
1. Get customer by ID
2. Get customer by Username
3. Register customer
4. Update customer
5. Delete customer
Enter your choice: 3
Enter first name: max
Enter last name: ver
Enter email: ver@gmail.com
Enter phone number: 2345678
Enter address: 123 monaco
Enter username: max
Enter password: 123max
Enter registration date (YYYY-MM-DD): 2024-05-05
Connected to MySQL database successfully.
Customer registered successfully.
```

```
Enter your choice: 1

Customer Operations:
1. Get customer by ID
2. Get customer by Username
3. Register customer
4. Update customer
5. Delete customer
Enter your choice: 3
Enter first name: demo
Enter last name: demo
Enter email: demo@mail.com
Enter phone number: 1234
Enter address: demo
Enter username: max
Enter password: 123r
Enter registration date (YYYY-MM-DD): 2024-05-06
Connected to MySQL database successfully.
A customer with that username already exists
Failed to register customer.
```

Update Customer:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter your choice: 1

Customer Operations:
1. Get customer by ID
2. Get customer by Username
3. Register customer
4. Update customer
5. Delete customer
Enter your choice: 4
Enter customer ID: 2
Enter new first name: charles
Enter new last name: leclerc
Enter new email: leclerc@mail.com
Enter new phone number: 5678903234
Enter new address: 234 monaco
Enter new username: charles
Enter new password: charles12
Connected to MySQL database successfully.
Customer updated successfully.
```

Update Vehicle:

```
Enter your choice: 2

Vehicle Operations:
1. Get vehicle by ID
2. Get available vehicles
3. Add vehicle
4. Update vehicle
5. Remove vehicle
Enter your choice: 4
Enter vehicle ID: 14
Enter new model: new model
Enter new make: new make
Enter new year: 2222
Enter new color: red
Enter new registration number: new45
Is vehicle available? (True/False): 1
Enter new daily rate: 12
Connected to MySQL database successfully.
Connected to MySQL database successfully.
Vehicle updated successfully.
```

Get Reservation by ID:

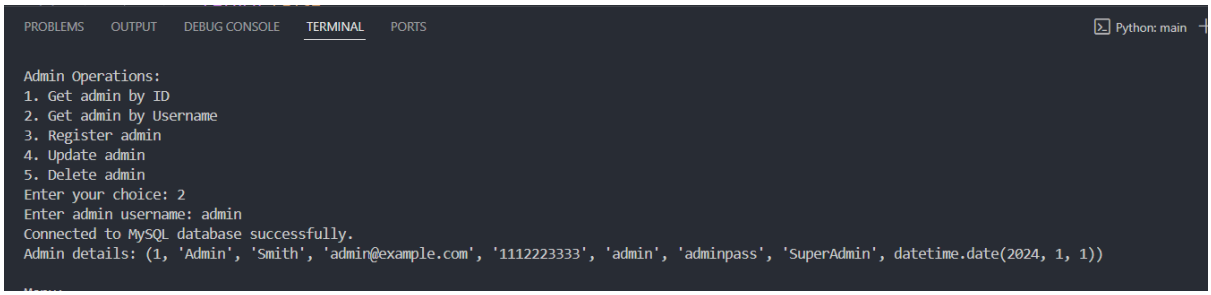
```
Menu:
1. Customer Operations
2. Vehicle Operations
3. Reservation Operations
4. Admin Operations
5. Authenticate Customer
6. Authenticate Admin
7. Generate Vehicles Reports
8. Generate Reservations Reports
9. Exit
Enter your choice: 3

Reservation Operations:
1. Get reservation by ID
2. Get reservations by Customer ID
3. Create reservation
4. Update reservation
5. Cancel reservation
Enter your choice: 1
Enter reservation ID: 3
Connected to MySQL database successfully.
Reservation details: (3, None, None, datetime.datetime(2024, 12, 12, 0, 0), datetime.datetime(2026, 12, 12, 0, 0), Decimal('23456.00'), 'confirmed')
```

Get reservation by customer ID:

```
Reservation Operations:
1. Get reservation by ID
2. Get reservations by Customer ID
3. Create reservation
4. Update reservation
5. Cancel reservation
Enter your choice: 2
Enter customer ID: 4
Connected to MySQL database successfully.
Reservations for customer ID 4
(5, 4, 4, datetime.datetime(2026, 12, 12, 0, 0), datetime.datetime(2027, 12, 12, 0, 0), Decimal('12343.00'), 'pending')
```

Admin Operations:

Get admin by id:



Get admin by username:

```
Admin Operations:
1. Get admin by ID
2. Get admin by Username
3. Register admin
4. Update admin
5. Delete admin
Enter your choice: 2
Enter admin username: admin
Connected to MySQL database successfully.
Admin details: (1, 'Admin', 'Smith', 'admin@example.com', '1112223333', 'admin', 'adminpass', 'SuperAdmin', datetime.date(2024, 1, 1))
```

Register admin:

```
Admin Operations:
1. Get admin by ID
2. Get admin by Username
3. Register admin
4. Update admin
5. Delete admin
Enter your choice: 3
Enter first name: demo
Enter last name: demo
Enter email: demo@mail.com
Enter phone number: 111111
Enter username: demo
Enter password: demo
Enter role: admin
Enter join date (YYYY-MM-DD): 2024-01-10
Connected to MySQL database successfully.
Admin registered successfully.
```

Update admin:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

4. Admin Operations
5. Authenticate Customer
6. Authenticate Admin
7. Generate Vehicles Reports
8. Generate Reservations Reports
9. Exit
Enter your choice: 4

Admin Operations:
1. Get admin by ID
2. Get admin by Username
3. Register admin
4. Update admin
5. Delete admin
Enter your choice: 4
Enter admin ID: 3
Enter new first name: bob
Enter new last name: davis
Enter new email: bob.davis@example.com
Enter new phone number: 3334445555
Enter new username: bobd
Enter new password: password123
Enter new role: Admin
Enter new join date (YYYY-MM-DD): 2024-03-20
Connected to MySQL database successfully.
Connected to MySQL database successfully.
Admin updated successfully.
```

Delete admin:

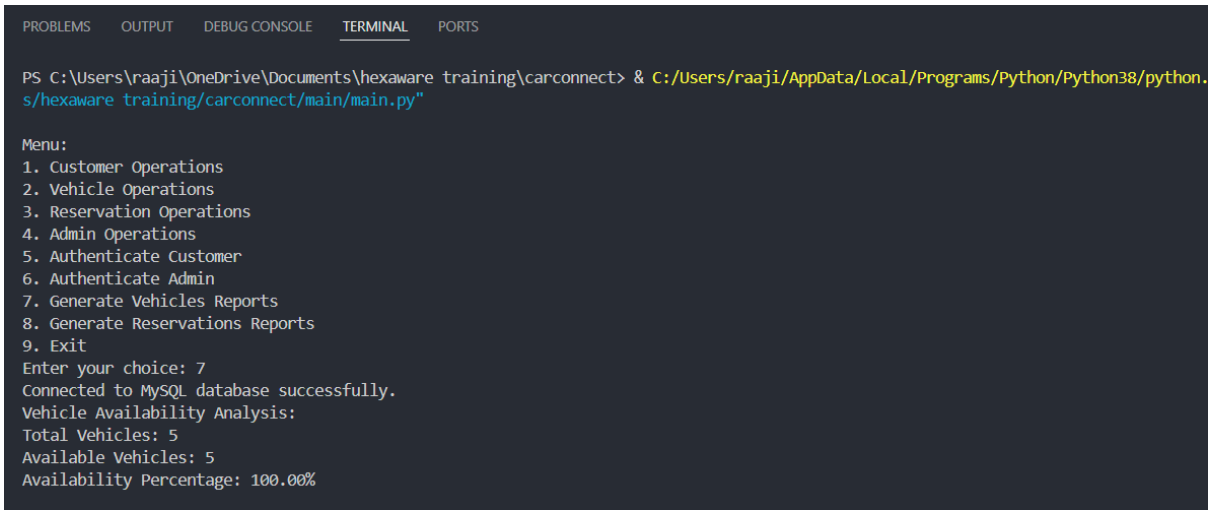
```
Menu:
1. Customer Operations
2. Vehicle Operations
3. Reservation Operations
4. Admin Operations
5. Authenticate Customer
6. Authenticate Admin
7. Generate Vehicles Reports
8. Generate Reservations Reports
9. Exit
Enter your choice: 4

Admin Operations:
1. Get admin by ID
2. Get admin by Username
3. Register admin
4. Update admin
5. Delete admin
Enter your choice: 5
Enter admin ID: 9
Connected to MySQL database successfully.
Connected to MySQL database successfully.
Admin deleted successfully.
```

Authenticate Customer:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Menu:
1. Customer Operations
2. Vehicle Operations
3. Reservation Operations
4. Admin Operations
5. Authenticate Customer
6. Authenticate Admin
7. Generate Vehicles Reports
8. Generate Reservations Reports
9. Exit
Enter your choice: 5
Enter username: johndoe
Enter password:password123
Connected to MySQL database successfully.
Customer Authentication Successful.
```

Authenticate Admin:

Generate Vehicles Reports:



Generate Reservations Reports: