

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
1  from exception.exceptions import InvalidInputException
2  from datetime import datetime
3
4  class Customer:
5      def __init__(self, customer_id=None, first_name="", last_name="", email="", phone_number="", address="", username="", password="", registration_date=None):
6          self.__customer_id = customer_id
7          self.__first_name = first_name
8          self.__last_name = last_name
9          self.__email = email
10         self.__phone_number = phone_number
11         self.__address = address
12         self.__username = username
13         self.__password = password
14         self.__registration_date = datetime.now()
15
16     @property
17     def customer_id(self):
18         return self.__customer_id
19
20     @customer_id.setter
21     def customer_id(self, value):
22         self.__customer_id = value
23
24     @property
25     def first_name(self):
26         return self.__first_name
27
28     @first_name.setter
29     def first_name(self, first_name):
30         if isinstance(first_name, str):
31             self.__first_name = first_name
32         else:
33             raise InvalidInputException("First name must be a string.")
34
35     @property
36     def last_name(self):
37         return self.__last_name
38
39     @last_name.setter
40     def last_name(self, last_name):
41         if isinstance(last_name, str):
42             self.__last_name = last_name
43         else:
44             raise InvalidInputException("Last name must be a string.")
45
46     @property
47     def email(self):
48         return self.__email
49
50     @email.setter
51     def email(self, email):
52         self.__email = self.__validate_email(email)
53
54     @property
55     def phone_number(self):
56         return self.__phone_number
57
```



```

carconnect > entity > customer.py > Customer
4  class Customer:
57
58     @phone_number.setter
59     def phone_number(self, phone):
60         if isinstance(phone, str) and phone.isdigit():
61             self.__phone_number = phone
62         else:
63             raise InvalidInputException("Phone number must be a string containing only digits.")
64
65     @property
66     def address(self):
67         return self.__address
68
69     @address.setter
70     def address(self, address):
71         if isinstance(address, str):
72             self.__address = address
73         else:
74             raise InvalidInputException("Address must be a string.")
75
76     @property
77     def username(self):
78         return self.__username
79
80     @username.setter
81     def username(self, value):
82         self.__username = value
83
84     @property
85     def password(self):
86         return self.__password
87
88     @password.setter
89     def password(self, value):
90         self.__password = value
91
92     @property
93     def registration_date(self):
94         return self.__registration_date
95
96     @registration_date.setter
97     def registration_date(self, value):
98         self.__registration_date = value
99
100     def authenticate(self, password):
101         return self.__password == password
102
103

```

Vehicle:

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

```

carconnect > entity > vehicle.py > Vehicle
1  class Vehicle:
2      def __init__(self, vehicle_id=None, model=None, make=None, year=None, color=None, registration_number=None, availability=None, daily_rate=None):
3          self.__vehicle_id = vehicle_id
4          self.__model = model
5          self.__make = make
6          self.__year = year
7          self.__color = color
8          self.__registration_number = registration_number
9          self.__availability = availability
10         self.__daily_rate = daily_rate
11
12     @property
13     def vehicle_id(self):
14         return self.__vehicle_id
15

```

```

15
16 @vehicle_id.setter
17 def vehicle_id(self, value):
18     self.__vehicle_id = value
19
20 @property
21 def model(self):
22     return self.__model
23
24 @model.setter
25 def model(self, value):
26     self.__model = value
27
28 @property
29 def make(self):
30     return self.__make
31
32 @make.setter
33 def make(self, value):
34     self.__make = value
35
36 @property
37 def year(self):
38     return self.__year
39
40 @year.setter
41 def year(self, value):
42     self.__year = value
43
44 @property
45 def color(self):
46     return self.__color
47
48 @color.setter
49 def color(self, value):
50     self.__color = value
51
52 @property
53 def registration_number(self):
54     return self.__registration_number

```

```

55
56 @registration_number.setter
57 def registration_number(self, value):
58     self.__registration_number = value
59
60 @property
61 def availability(self):
62     return self.__availability
63
64 @availability.setter
65 def availability(self, value):
66     self.__availability = value
67
68 @property
69 def daily_rate(self):
70     return self.__daily_rate
71
72 @daily_rate.setter
73 def daily_rate(self, value):
74     self.__daily_rate = value
75

```

Reservation:

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

Methods: CalculateTotalCost()

```

carconnect > entity > reservation.py > Reservation
1 class Reservation:
2     def __init__(self, reservation_id=None, customer_id=None, vehicle_id=None, start_date=None, end_date=None, total_cost=None, status=None):
3         self.__reservation_id = reservation_id
4         self.__customer_id = customer_id
5         self.__vehicle_id = vehicle_id
6         self.__start_date = start_date
7         self.__end_date = end_date
8         self.__total_cost = total_cost
9         self.__status = status
10
11 @property
12 def reservation_id(self):
13     return self.__reservation_id
14
15 @reservation_id.setter
16 def reservation_id(self, value):
17     self.__reservation_id = value
18

```

carconnect > entity > reservation.py > Reservation

```
1  class Reservation:
18
19      @property
20      def customer_id(self):
21          return self.__customer_id
22
23      @customer_id.setter
24      def customer_id(self, value):
25          self.__customer_id = value
26
27      @property
28      def vehicle_id(self):
29          return self.__vehicle_id
30
31      @vehicle_id.setter
32      def vehicle_id(self, value):
33          self.__vehicle_id = value
34
35      @property
36      def start_date(self):
37          return self.__start_date
38
39      @start_date.setter
40      def start_date(self, value):
41          self.__start_date = value
42
43      @property
44      def end_date(self):
45          return self.__end_date
46
47      @end_date.setter
48      def end_date(self, value):
49          self.__end_date = value
50
51      @property
52      def total_cost(self):
53          return self.__total_cost
54
55      @total_cost.setter
56      def total_cost(self, value):
57          self.__total_cost = value
58
59      @property
60      def status(self):
61          return self.__status
62
63      @status.setter
64      def status(self, value):
65          self.__status = value
66
67      def calculate_total_cost(self):
68          # Your calculation logic here
69          pass
70
```

Admin:

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

Methods: Authenticate(password)

```
carconnect > entity > admin.py > ...
1  import re
2  from exception.exceptions import InvalidInputException
3
4  class Admin:
5      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):
6          self.__admin_id = admin_id
7          self.__first_name = first_name
8          self.__last_name = last_name
9          self.__email = email
10         self.__phone_number = phone_number
11         self.__username = username
12         self.__password = password
13         self.__role = role
14         self.__join_date = join_date
15
16     def __validate_email(self, email):
17         if re.match(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', email):
18             return email
19         else:
20             raise InvalidInputException("Invalid email address provided.")
21
22     @property
23     def admin_id(self):
24         return self.__admin_id
25
26     @admin_id.setter
27     def admin_id(self, value):
28         self.__admin_id = value
29
30     @property
31     def first_name(self):
32         return self.__first_name
33
34     @first_name.setter
35     def first_name(self, first_name):
36         if isinstance(first_name, str):
37             self.__first_name = first_name
38         else:
39             raise InvalidInputException("First name must be a string.")
40
41     @property
42     def last_name(self):
43         return self.__last_name
44
45     @last_name.setter
46     def last_name(self, last_name):
47         if isinstance(last_name, str):
48             self.__last_name = last_name
49         else:
50             raise InvalidInputException("Last name must be a string.")
51
52     @property
53     def email(self):
54         return self.__email
55
```

```
carconnect > entity > admin.py > ...
4  class Admin:
5
56     @email.setter
57     def email(self, email):
58         self.__email = self.__validate_email(email)
59
60     @property
61     def phone_number(self):
62         return self.__phone_number
63
64     @phone_number.setter
65     def phone_number(self, phone):
66         if isinstance(phone, str) and phone.isdigit():
67             self.__phone_number = phone
68         else:
69             raise InvalidInputException("Phone number must be a string containing only digits.")
70
71     @property
72     def username(self):
73         return self.__username
74
```

```

carconnect > entity > admin.py > ...
4   class Admin:
75   @username.setter
76   def username(self, value):
77       self.__username = value
78
79   @property
80   def password(self):
81       return self.__password
82
83   @password.setter
84   def password(self, value):
85       self.__password = value
86
87   @property
88   def role(self):
89       return self.__role
90
91   @role.setter
92   def role(self, value):
93       self.__role = value
94
95   @property
96   def join_date(self):
97       return self.__join_date
98
99   @join_date.setter
100  def join_date(self, value):
101      self.__join_date = value
102
103  def authenticate(self, password):
104      return self.__password == password
105

```

CustomerService (implements ICustomerService):

Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

```

carconnect > dao > imp > customerService.py > CustomerService > get_customer_by_id
1
2   import mysql.connector
3
4   from dao.interface.ICustomerService import ICustomerService
5   from util.db_connection import DBConnection
6
7   class CustomerService(ICustomerService):
8
9       def get_customer_by_id(self, customer_id):
10          try:
11              connection = DBConnection.getConnection()
12              cursor = connection.cursor()
13              cursor.execute("SELECT * FROM customer WHERE customerID = %s", (customer_id,))
14              customer = cursor.fetchone()
15              cursor.close()
16              return customer
17          except mysql.connector.Error as e:
18              print("Error getting customer by id:", e)
19              return None
20          finally:
21              if 'connection' in locals() and connection.is_connected():
22                  connection.close()
23
24       def get_customer_by_username(self, username):
25          try:
26              connection = DBConnection.getConnection()
27              cursor = connection.cursor()
28              cursor.execute("SELECT * FROM customer WHERE username = %s", (username,))
29              customer = cursor.fetchone()
30              cursor.close()
31              return customer
32          except mysql.connector.Error as e:
33              print("Error getting customer by username:", e)
34              return None
35          finally:
36              if 'connection' in locals() and connection.is_connected():
37                  connection.close()
38

```

```

carconnect > dao > imp > customerService.py > CustomerService > delete_customer
7 class CustomerService(ICustomerService):
38
39     def register_customer(self, customer):
40         try:
41             connection = DBConnection.getConnection()
42             cursor = connection.cursor()
43             cursor.execute("INSERT INTO customer (firstname, lastname, email, phoneNumber, address, username, password, registrationDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)",
44                             (customer.first_name, customer.last_name, customer.email, customer.phone_number, customer.address, customer.username, customer.password, customer.registration_date))
45             connection.commit()
46             cursor.close()
47             return True
48         except mysql.connector.Error as e:
49             print("Error registering customer:", e)
50             return False
51         finally:
52             if 'connection' in locals() and connection.is_connected():
53                 connection.close()
54
55     def update_customer(self, customer):
56         try:
57             connection = DBConnection.getConnection()
58             cursor = connection.cursor()
59             cursor.execute("UPDATE customer SET firstname = %s, lastname = %s, email = %s, phoneNumber = %s, address = %s, username = %s, password = %s WHERE customerID = %s",
60                             (customer.first_name, customer.last_name, customer.email, customer.phone_number, customer.address, customer.username, customer.password, customer.customer_id))
61             connection.commit()
62             cursor.close()
63             return True
64         except mysql.connector.Error as e:
65             print("Error updating customer:", e)
66             return False
67         finally:
68             if 'connection' in locals() and connection.is_connected():
69                 connection.close()
70
71     def delete_customer(self, customer_id):
72         try:
73             connection = DBConnection.getConnection()
74             cursor = connection.cursor()
75             cursor.execute("DELETE FROM customer WHERE customerID = %s", (customer_id,))
76             connection.commit()
77             cursor.close()
78             return True
79         except mysql.connector.Error as e:
80             print("Error deleting customer:", e)
81             return False
82         finally:
83             if 'connection' in locals() and connection.is_connected():
84                 connection.close()
85

```

VehicleService (implements IVehicleService):

Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

```

carconnect > dao > imp > vehicleService.py > VehicleService > get_available_vehicles
1 import mysql.connector
2
3 from dao.interface.IVehicleService import IVehicleService
4 from exception.exceptions import VehicleNotFoundException
5 from util.db_connection import DBConnection
6
7 class VehicleService(IVehicleService):
8
9     def get_vehicle_by_id(self, vehicle_id):
10         try:
11             connection = DBConnection.getConnection()
12             cursor = connection.cursor()
13             cursor.execute("SELECT * FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
14             vehicle = cursor.fetchone()
15             cursor.close()
16             if not vehicle:
17                 raise VehicleNotFoundException("Vehicle not found.")
18             return vehicle
19         except mysql.connector.Error as e:
20             print("Error getting vehicle by id:", e)
21             return None
22         finally:
23             if 'connection' in locals() and connection.is_connected():
24                 connection.close()
25
26     def get_available_vehicles(self):
27         try:
28             connection = DBConnection.getConnection()
29             cursor = connection.cursor()
30             cursor.execute("SELECT * FROM vehicle WHERE availability = %s", (True,))
31             vehicles = cursor.fetchall()
32             cursor.close()
33             return vehicles
34         except mysql.connector.Error as e:
35             print("Error getting available vehicles:", e)
36             return []
37         finally:
38             if 'connection' in locals() and connection.is_connected():
39                 connection.close()
40

```

```

carconnect > dao > imp > vehicleService.py > VehicleService > get_available_vehicles
7 class VehicleService(IVehicleService):
8
9     def add_vehicle(self, vehicle):
10         try:
11             connection = DBConnection.getConnection()
12             cursor = connection.cursor()
13             cursor.execute("INSERT INTO vehicle (model, make, year, color, registrationNumber, availability, dailyRate) VALUES (%s, %s, %s, %s, %s, %s, %s)".
14                             (vehicle.model, vehicle.make, vehicle.year, vehicle.color, vehicle.registration_number, vehicle.availability, vehicle.daily_rate))
15             connection.commit()
16             cursor.close()
17             return True
18         except mysql.connector.Error as e:
19             print("Error adding vehicle:", e)
20             return False
21         finally:
22             if 'connection' in locals() and connection.is_connected():
23                 connection.close()
24
25     def update_vehicle(self, vehicle):
26         veh = self.get_vehicle_by_id(vehicle.vehicle_id)
27         if not veh:
28             return
29         try:
30             connection = DBConnection.getConnection()
31             cursor = connection.cursor()
32             cursor.execute("UPDATE vehicle SET model = %s, make = %s, year = %s, color = %s, registrationNumber = %s, availability = %s, dailyRate = %s WHERE vehicleID = %s",
33                             (vehicle.model, vehicle.make, vehicle.year, vehicle.color, vehicle.registration_number, vehicle.availability, vehicle.daily_rate, vehicle.vehicle_id))
34             connection.commit()
35             cursor.close()
36             return True
37         except mysql.connector.Error as e:
38             print("Error updating vehicle:", e)
39             return False
40         finally:
41             if 'connection' in locals() and connection.is_connected():
42                 connection.close()
43
44     def remove_vehicle(self, vehicle_id):
45         veh = self.get_vehicle_by_id(vehicle_id)
46         if not veh:
47             return
48         try:
49             connection = DBConnection.getConnection()
50             cursor = connection.cursor()
51             cursor.execute("DELETE FROM vehicle WHERE vehicleID = %s", (vehicle_id,))
52             connection.commit()
53             cursor.close()
54             return True
55         except mysql.connector.Error as e:
56             print("Error removing vehicle:", e)
57             return False
58         finally:
59             if 'connection' in locals() and connection.is_connected():
60                 connection.close()
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93

```

ReservationService (implements IReservationService):

Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation

```

carconnect > dao > imp > reservationService.py > ReservationService
1 import mysql.connector
2
3 from dao.interface.IReservationService import IReservationService
4 from exception.exceptions import ReservationException
5 from util.db_connection import DBConnection
6
7 class ReservationService(IReservationService):
8     def get_reservation_by_id(self, reservation_id):
9         try:
10             connection = DBConnection.getConnection()
11             cursor = connection.cursor()
12             cursor.execute("SELECT * FROM reservation WHERE reservationID = %s", (reservation_id,))
13             reservation = cursor.fetchone()
14             cursor.close()
15             return reservation
16         except mysql.connector.Error as e:
17             print("Error getting reservation by id:", e)
18             return None
19         finally:
20             if 'connection' in locals() and connection.is_connected():
21                 connection.close()
22
23     def get_reservations_by_customer_id(self, customer_id):
24         try:
25             connection = DBConnection.getConnection()
26             cursor = connection.cursor()
27             cursor.execute("SELECT * FROM reservation WHERE customerID = %s", (customer_id,))
28             reservations = cursor.fetchall()
29             cursor.close()
30             return reservations
31         except mysql.connector.Error as e:
32             print("Error getting reservations by customer id:", e)
33             return []
34         finally:
35             if 'connection' in locals() and connection.is_connected():
36                 connection.close()
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93

```

```

carconnect > dao > imp > reservationService.py > ReservationService
7 class ReservationService(IReservationService):
8
9     def create_reservation(self, reservation):
10         try:
11             if self.is_vehicle_booked(reservation.vehicle_id, reservation.start_date, reservation.end_date):
12                 raise ReservationException("Vehicle is already booked for the specified dates.")
13
14             connection = DBConnection.getConnection()
15             cursor = connection.cursor()
16             cursor.execute("INSERT INTO reservation (customerID, vehicleID, startDate, endDate, totalCost, status) VALUES (%s, %s, %s, %s, %s, %s)",
17                             (reservation.customer_id, reservation.vehicle_id, reservation.start_date, reservation.end_date, reservation.total_cost, reservation.status))
18             connection.commit()
19             cursor.close()
20             return True
21         except mysql.connector.Error as e:
22             print("Error creating reservation:", e)
23             return False
24         except ReservationException as e:
25             print("Reservation error:", e)
26             return False
27         finally:
28             if 'connection' in locals() and connection.is_connected():
29                 connection.close()
30
31     def is_vehicle_booked(self, vehicle_id, start_date, end_date):
32         try:
33             connection = DBConnection.getConnection()
34             cursor = connection.cursor()
35             cursor.execute("SELECT * FROM reservation WHERE vehicleID = %s AND ((startDate BETWEEN %s AND %s) OR (endDate BETWEEN %s AND %s))",
36                             (vehicle_id, start_date, end_date, start_date, end_date))
37             existing_reservation = cursor.fetchone()
38             return existing_reservation is not None
39         except mysql.connector.Error as e:
40             print("Error checking existing reservation:", e)
41             return False
42         finally:
43             if 'connection' in locals() and connection.is_connected():
44                 connection.close()
45
46     def update_reservation(self, reservation):
47         try:
48             connection = DBConnection.getConnection()
49             cursor = connection.cursor()
50             cursor.execute("UPDATE reservation SET customerID = %s, vehicleID = %s, startDate = %s, endDate = %s, totalCost = %s, status = %s WHERE reservationID = %s",
51                             (reservation.customer_id, reservation.vehicle_id, reservation.start_date, reservation.end_date, reservation.total_cost, reservation.status, reservation.reservation_id))
52             connection.commit()
53             cursor.close()
54             return True
55         except mysql.connector.Error as e:
56             print("Error updating reservation:", e)
57             return False
58         finally:
59             if 'connection' in locals() and connection.is_connected():
60                 connection.close()
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90

```

```

90
91     def cancel_reservation(self, reservation_id):
92         try:
93             connection = DBConnection.getConnection()
94             cursor = connection.cursor()
95             cursor.execute("DELETE FROM reservation WHERE reservationID = %s", (reservation_id,))
96             connection.commit()
97             cursor.close()
98             return True
99         except mysql.connector.Error as e:
100             print("Error canceling reservation:", e)
101             return False
102         finally:
103             if 'connection' in locals() and connection.is_connected():
104                 connection.close()
105

```

AdminService (implements IAdminService):

Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

```

carconnect > dao > imp > adminService.py > AdminService
1 import mysql.connector
2
3 from dao.interface.IAdminService import IAdminService
4 from exception.exceptions import AdminNotFoundException
5 from util.db_connection import DBConnection
6
7 class AdminService(IAdminService):
8     def get_admin_by_id(self, admin_id):
9         try:
10             connection = DBConnection.getConnection()
11             cursor = connection.cursor()
12             cursor.execute("SELECT * FROM admin WHERE adminID = %s", (admin_id,))
13             admin = cursor.fetchone()
14             cursor.close()
15             if not admin:
16                 raise AdminNotFoundException("Admin not found.")
17             return admin
18         except mysql.connector.Error as e:
19             print("Error getting admin by id:", e)
20             return None
21         finally:
22             if 'connection' in locals() and connection.is_connected():
23                 connection.close()
24

```



```

carconnect > dao > imp > adminService.py > $ AdminService
7 class AdminService(IAdminService):
24
25     def get_admin_by_username(self, username):
26         try:
27             connection = DBConnection.getConnection()
28             cursor = connection.cursor()
29             cursor.execute("SELECT * FROM admin WHERE username = %s", (username,))
30             admin = cursor.fetchone()
31             cursor.close()
32             return admin
33         except mysql.connector.Error as e:
34             print("Error getting admin by username:", e)
35             return None
36         finally:
37             if 'connection' in locals() and connection.is_connected():
38                 connection.close()
39
40     def register_admin(self, admin):
41         try:
42             connection = DBConnection.getConnection()
43             cursor = connection.cursor()
44             cursor.execute("INSERT INTO admin (firstName, lastname, email, phoneNumber, username, password, role, joinDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)",
45                             (admin.first_name, admin.last_name, admin.email, admin.phone_number, admin.username, admin.password, admin.role, admin.join_date))
46             connection.commit()
47             cursor.close()
48             return True
49         except mysql.connector.Error as e:
50             print("Error registering admin:", e)
51             return False
52         finally:
53             if 'connection' in locals() and connection.is_connected():
54                 connection.close()
55
56     def update_admin(self, admin):
57         adm = self.get_admin_by_id(admin.admin_id)
58         if not adm:
59             return
60         try:
61             connection = DBConnection.getConnection()
62             cursor = connection.cursor()
63             cursor.execute("UPDATE admin SET firstname = %s, lastname = %s, email = %s, phoneNumber = %s, username = %s, password = %s, role = %s WHERE adminID = %s",
64                             (admin.first_name, admin.last_name, admin.email, admin.phone_number, admin.username, admin.password, admin.role, admin.admin_id))
65             connection.commit()
66             cursor.close()
67             return True
68         except mysql.connector.Error as e:
69             print("Error updating admin:", e)
70             return False
71         finally:
72             if 'connection' in locals() and connection.is_connected():
73                 connection.close()
74

```

```

74
75     def delete_admin(self, admin_id):
76         adm = self.get_admin_by_id(admin_id)
77         if not adm:
78             return
79         try:
80             connection = DBConnection.getConnection()
81             cursor = connection.cursor()
82             cursor.execute("DELETE FROM admin WHERE adminID = %s", (admin_id,))
83             connection.commit()
84             cursor.close()
85             return True
86         except mysql.connector.Error as e:
87             print("Error deleting admin:", e)
88             return False
89         finally:
90             if 'connection' in locals() and connection.is_connected():
91                 connection.close()
92

```

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
1 from dao.imp.adminService import AdminService
2 from dao.imp.customerService import CustomerService
3
4
5 class AuthenticationService:
6     def __init__(self):
7         self.cust_obj = CustomerService()
8         self.admin_obj = AdminService()
9
10    def authenticate_customer(self, username, password):
11        customer = self.cust_obj.get_customer_by_username(username)
12        if customer and customer[7] == password:
13            return True
14        else:
15            return False
16
17    def authenticate_admin(self, username, password):
18        admin = self.admin_obj.get_admin_by_username(username)
19        if admin and admin[6] == password:
20            return True
21        else:
22            return False
23
24

```

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

```

carconnect > dao > imp > reportsGenerator.py > ReportsGenerator > generate_reservation_report > reservation_statuses
1 import mysql.connector
2 from util.db_connection import DBConnection
3
4 class ReportsGenerator:
5     def generate_vehicle_report(self):
6         try:
7             connection = DBConnection.getConnection()
8             cursor = connection.cursor()
9
10            query = "SELECT COUNT(*) FROM vehicle WHERE availability = 1"
11            cursor.execute(query)
12            available_count = cursor.fetchone()[0]
13            query = "SELECT COUNT(*) FROM vehicle"
14            cursor.execute(query)
15            total_count = cursor.fetchone()[0]
16            availability_percentage = (available_count / total_count) * 100
17            print("Vehicle Availability Analysis:")
18            print(f"Total Vehicles: {total_count}")
19            print(f"Available Vehicles: {available_count}")
20            print(f"Availability Percentage: {availability_percentage:.2f}%")
21        except mysql.connector.Error as e:
22            print("Error generating vehicle report:", e)
23        finally:
24            if 'connection' in locals() and connection.is_connected():
25                cursor.close()
26                connection.close()
27
28    def generate_reservation_report(self):
29        try:
30            connection = DBConnection.getConnection()
31            cursor = connection.cursor()
32            query = "SELECT status, COUNT(*) FROM reservation GROUP BY status"
33            cursor.execute(query)
34            reservation_statuses = cursor.fetchall()
35            print("Reservation Status Analysis:")
36            for status in reservation_statuses:
37                print(f"Status: {status[0]}, Count: {status[1]}")
38        except mysql.connector.Error as e:
39            print("Error generating reservation report:", e)
40        finally:
41            if 'connection' in locals() and connection.is_connected():
42                cursor.close()
43                connection.close()
44

```

Interfaces:

ICustomerService:

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

```

carconnect > dao > interface > ICustomerService.py > ...
1  from abc import ABC, abstractmethod
2
3  class ICustomerService(ABC):
4      @abstractmethod
5      def get_customer_by_id(self, customer_id):
6          pass
7
8      @abstractmethod
9      def get_customer_by_username(self, username):
10         pass
11
12     @abstractmethod
13     def register_customer(self, customer_data):
14         pass
15
16     @abstractmethod
17     def update_customer(self, customer_data):
18         pass
19
20     @abstractmethod
21     def delete_customer(self, customer_id):
22         pass
23
24

```

IVehicleService:

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

```

carconnect > dao > interface > IVehicleService.py > IVehicleService > remove_vehicle
1  from abc import ABC, abstractmethod
2
3  class IVehicleService(ABC):
4      @abstractmethod
5      def get_vehicle_by_id(self, vehicle_id):
6          pass
7
8      @abstractmethod
9      def get_available_vehicles(self):
10         pass
11
12     @abstractmethod
13     def add_vehicle(self, vehicle_data):
14         pass
15
16     @abstractmethod
17     def update_vehicle(self, vehicle_data):
18         pass
19
20     @abstractmethod
21     def remove_vehicle(self, vehicle_id):
22         pass
23

```

IReservationService:

- GetReservationById(reservationId) • GetReservationsByCustomerId(customerId)
- CreateReservation(reservationData) • UpdateReservation(reservationData)
- CancelReservation(reservationId)

```

carconnect > dao > interface > IReservationService.py > ...
1  from abc import ABC, abstractmethod
2
3  class IReservationService(ABC):
4      @abstractmethod
5      def get_reservation_by_id(self, reservation_id):
6          pass
7
8      @abstractmethod
9      def get_reservations_by_customer_id(self, customer_id):
10         pass
11
12     @abstractmethod
13     def create_reservation(self, reservation_data):
14         pass
15
16     @abstractmethod
17     def update_reservation(self, reservation_data):
18         pass
19
20     @abstractmethod
21     def cancel_reservation(self, reservation_id):
22         pass
23

```

IAdminService:

- GetAdminById(adminId) • GetAdminByUsername(username) • RegisterAdmin(adminData)
- UpdateAdmin(adminData) • DeleteAdmin(adminId)

```
carconnect > dao > interface > IAdminService.py > IAdminService
1  from abc import ABC, abstractmethod
2
3
4  class IAdminService(ABC):
5      @abstractmethod
6      def get_admin_by_id(self, admin_id):
7          pass
8
9      @abstractmethod
10     def get_admin_by_username(self, username):
11         pass
12
13     @abstractmethod
14     def register_admin(self, admin_data):
15         pass
16
17     @abstractmethod
18     def update_admin(self, admin_data):
19         pass
20
21     @abstractmethod
22     def delete_admin(self, admin_id):
23         pass
24
```

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
1  # util/db_connection.py
2  import mysql.connector
3  from exception.exceptions import DatabaseConnectionException
4  from util.property_util import PropertyUtil
5  import time
6
7  class DBConnection:
8      __connection = None
9
10     @staticmethod
11     def getConnection():
12         property_file = r"C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect\util\db.properties"
13         connection_string = PropertyUtil().getPropertyString(property_file)
14         try:
15             DBConnection.__connection = mysql.connector.connect(**connection_string)
16             print("Connected to MySQL database successfully.")
17             return DBConnection.__connection
18         except mysql.connector.Error as err:
19             print("Error connecting to MySQL:", err)
20             raise DatabaseConnectionException()
21         return None
22
```

```

carconnect > util > property_util.py > ...
1  import configparser
2
3  class PropertyUtil:
4      @staticmethod
5      def getPropertyString(property_file):
6          config = configparser.ConfigParser()
7          config.read(property_file)
8          connection_string = {
9              'user': config['DATABASE']['user'],
10             'host': config['DATABASE']['host'],
11             'port': config['DATABASE'].getint('port'),
12             'passwd': config['DATABASE']['passwd'],
13             'database': config['DATABASE']['database']
14         }
15         return connection_string
16

```

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 > ...
1  class AuthenticationException(Exception):
2      def __init__(self, message="Authentication error"):
3          self.message = message
4          super().__init__(self.message)
5
6
7  class ReservationException(Exception):
8      def __init__(self, message="Reservation error"):
9          self.message = message
10         super().__init__(self.message)
11
12
13 class VehicleNotFoundException(Exception):
14     def __init__(self, message="Vehicle not found"):
15         self.message = message
16         super().__init__(self.message)
17
18
19 class AdminNotFoundException(Exception):
20     def __init__(self, message="Admin not found"):
21         self.message = message
22         super().__init__(self.message)
23
24
25 class InvalidInputException(Exception):
26     def __init__(self, message="Invalid input"):
27         self.message = message
28         super().__init__(self.message)
29
30
31 class DatabaseConnectionException(Exception):
32     def __init__(self, message="Database connection error"):
33         self.message = message
34         super().__init__(self.message)
35

```

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 0x0000012D45AFEFD0>>('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.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> pytest -k test_update_customer_info
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect
collected 6 items / 5 deselected / 1 selected

testing\test_car_connect_system.py . [100%]

===== 1 passed, 5 deselected in 0.27s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```

3. Test adding a new vehicle.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> pytest -k test_add_new_vehicle
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect
collected 6 items / 5 deselected / 1 selected

testing\test_car_connect_system.py . [100%]

===== 1 passed, 5 deselected in 0.17s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

===== FAILURES =====
test_add_new_vehicle

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": 5
0}
    vehicle = Vehicle(**vehicle_data)
> assert vehicle_service.add_vehicle(vehicle)
E assert False
E + where False = <bound method VehicleService.add_vehicle of <dao.imp.vehicleService.VehicleService object at 0x000001C91C866310>>(<entity.vehicle.Vehicle obj
ect at 0x000001C91C866760>)
E + where <bound method VehicleService.add_vehicle of <dao.imp.vehicleService.VehicleService object at 0x000001C91C866310> = <dao.imp.vehicleService.Vehicle
Service object at 0x000001C91C866310>.add_vehicle

testing\test_car_connect_system.py:34: AssertionError
----- Captured stdout call -----
Connected to MySQL database successfully.
A vehicle with the same registration number already exists

===== short test summary info =====
FAILED testing\test_car_connect_system.py::test_add_new_vehicle - assert False
===== 1 failed, 5 deselected in 0.42s =====
```

4. Test updating vehicle details.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> pytest -k test_update_vehicle_details
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect
collected 6 items / 5 deselected / 1 selected

testing\test_car_connect_system.py . [100%]

===== 1 passed, 5 deselected in 0.21s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

===== FAILURES =====
test_update_vehicle_details

def test_update_vehicle_details():
    vehicle_service = VehicleService()
    vehicle_data = {"vehicle_id": 111, "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)
E
E   + where False = <bound method VehicleService.update_vehicle of <dao.imp.vehicleService.VehicleService object at 0x00000227E413EF70>>(<entity.vehicle.Vehicle object at 0x00000227E413EFD0>)
E   + where <bound method VehicleService.update_vehicle of <dao.imp.vehicleService.VehicleService object at 0x00000227E413EF70> = <dao.imp.vehicleService.VehicleService object at 0x00000227E413EF70>.update_vehicle

testing\test_car_connect_system.py:41: AssertionError
----- Captured stdout call -----
Connected to MySQL database successfully.
Vehicle not found.
vehicle with 111 not found.

===== short test summary info =====
FAILED testing/test_car_connect_system.py::test_update_vehicle_details - assert False
===== 1 failed, 5 deselected in 0.43s =====
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

E   + where False = <bound method VehicleService.update_vehicle of <dao.imp.vehicleService.VehicleService object at 0x00000241A5A4ECD0>>(<entity.vehicle.Vehicle object at 0x00000241A5A4EFD0>)
E   + where <bound method VehicleService.update_vehicle of <dao.imp.vehicleService.VehicleService object at 0x00000241A5A4ECD0> = <dao.imp.vehicleService.VehicleService object at 0x00000241A5A4ECD0>.update_vehicle

testing\test_car_connect_system.py:41: AssertionError
----- Captured stdout call -----
Connected to MySQL database successfully.
Connected to MySQL database successfully.
A vehicle with the same registration number already exists

===== short test summary info =====
FAILED testing/test_car_connect_system.py::test_update_vehicle_details - assert False
===== 1 failed, 5 deselected in 0.40s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```

5. Test getting a list of available vehicles.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ] ... ^ x

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> pytest -k test_get_available_vehicles
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect
collected 6 items / 5 deselected / 1 selected

testing\test_car_connect_system.py . [100%]

===== 1 passed, 5 deselected in 0.26s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```

6. Test getting a list of all vehicles.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: test_car_connect_system + - [ ]

PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect> pytest -k test_get_all_vehicles
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect
collected 6 items / 5 deselected / 1 selected

testing\test_car_connect_system.py .

===== 1 passed, 5 deselected in 0.24s =====
PS C:\Users\raaji\OneDrive\Documents\hexaware training\carconnect>
```



```

carconnect > testing > test_car_connect_system.py > test_update_customer_info
1 import sys
2 import os
3 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
4
5 import pytest
6 from entity.customer import Customer
7 from entity.vehicle import Vehicle
8 from dao.imp.authenticationService import AuthenticationService
9 from dao.imp.customerService import CustomerService
10 from dao.imp.vehicleService import VehicleService
11
12 def test_customer_authentication_invalid():
13     auth_service = AuthenticationService()
14     assert not auth_service.authenticate_customer("invalid_username", "invalid_password")
15
16 def test_update_customer_info():
17     customer_service = CustomerService()
18     customer_data = Customer()
19     customer_data.customer_id = 1,
20     customer_data.first_name = "New First Name",
21     customer_data.last_name = "New Last Name",
22     customer_data.email = "new_email@example.com",
23     customer_data.phone_number = "1234567890",
24     customer_data.address = "New Address",
25     customer_data.username = "new_username",
26     customer_data.password = "new_password"
27
28     assert customer_service.update_customer(customer_data)
29
30 def test_add_new_vehicle():
31     vehicle_service = VehicleService()
32     vehicle_data = {"model": "New Model", "make": "New Make", "year": 2022, "color": "Red", "registration_number": "NEW1234", "availability": True, "daily_rate": 50}
33     vehicle = Vehicle(**vehicle_data)
34     assert vehicle_service.add_vehicle(vehicle)
35
36
37 def test_update_vehicle_details():
38     vehicle_service = VehicleService()
39     vehicle_data = {"vehicle_id": 1, "model": "Updated Model", "make": "Updated Make", "year": 2023, "color": "Blue", "registration_number": "UPDATED123", "availability": True, "daily_rate": 60}
40     vehicle = Vehicle(**vehicle_data)
41     assert vehicle_service.update_vehicle(vehicle)
42
43
44 def test_get_available_vehicles():
45     vehicle_service = VehicleService()
46     available_vehicles = vehicle_service.get_available_vehicles()
47     assert available_vehicles is not None
48
49
50 def test_get_all_vehicles():
51     vehicle_service = VehicleService()
52     all_vehicles = vehicle_service.get_all_vehicles()
53     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(2024, 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:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: main +
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))
```

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:

```
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: 6
Enter username: admin
Enter password: adminpass
Connected to MySQL database successfully.
Admin Authentication Successful.
```

Generate Vehicles Reports:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

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

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: 7
Connected to MySQL database successfully.
Vehicle Availability Analysis:
Total Vehicles: 5
Available Vehicles: 5
Availability Percentage: 100.00%
```

Generate Reservations Reports:

```
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: 8
Connected to MySQL database successfully.
Reservation Status Analysis:
Status: confirmed, Count: 3
Status: pending, Count: 2
```