HEXAWARE CASE STUDY - 1

TEAM MEMBERS :
AISHWARYA B
MOHAMMED IBRAHIM SHERIFF
CASE STUDY NAME: 1- CAR CONNECT

ABSTRACT:

In an era of rapidly evolving mobility solutions, **CarConnect** presents a robust, scalable, and database-driven car rental management system developed using **Python** and **MySQL**, aligned with industry practices for real-time, service-based applications. This project is designed to simulate and streamline the operations of a car rental service, offering both **admin** and **customer-facing functionalities** through a clean command-line interface.

The system architecture is layered and modular, comprising:

- **Entity Layer**: Represents real-world objects like Customer, Vehicle, Reservation, and Admin.
- **DAO and Interface Layer**: Ensures a clean separation of database access logic via well-defined interfaces and implementation classes using parameterised SQL queries (mysql-connector).
- **Service Layer**: Encapsulates business rules such as booking constraints, vehicle availability, and discount logic.
- **Utility Layer**: Handles authentication, exceptions, and database connection pooling, ensuring fault-tolerant operations.
- **CLI-based UI**: Provides role-based access and guided workflows for customers and administrators.

Key business features include:

- Real-time vehicle reservation with overlap prevention logic.
- Birthday month loyalty discount (15%) and age restrictions (minimum 15 years) for safe and ethical booking.
- Admin capabilities to manage vehicles, approve/cancel reservations, and generate SQL-driven reports on fleet utilisation and revenue trends.

Unit testing is done using **Pytest**, covering authentication, customer updates, vehicle management, and availability logic. All components are tested against a live database to ensure production-grade reliability. **CarConnect** is a proof-of-concept that showcases backend software development best practices, demonstrating both technical proficiency and the ability to solve real-world problems efficiently.

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

1. Customer Table:
Query:

-> create table Customer(CustomerID int primary key auto_increment,FirstName varchar(60),LastName varchar(60),Email varchar(60),PhoneNumber varchar(11),Address text,Username varchar(60) unique,Password varchar(200),RegistrationDate date);

```
mysql> create table Customer(CustomerID int primary key, FirstName varchar(6
0), LastName varchar(60),Email varchar(55), PhoneNumber varchar(11),Address
text, Username varchar(60) unique,Password varchar(60),RegistrationDate date
);
Query OK, 0 rows affected (0.22 sec)

mysql> desc customer;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| CustomerID       | int         | NO   | PRI | NULL    |       |
| FirstName        | varchar(60) | YES  |     | NULL    |       |
| LastName         | varchar(60) | YES  |     | NULL    |       |
| Email            | varchar(55) | YES  |     | NULL    |       |
| PhoneNumber      | varchar(11) | YES  |     | NULL    |       |
| Address          | text        | YES  |     | NULL    |       |
| Username         | varchar(60) | YES  | UNI | NULL    |       |
| Password         | varchar(60) | YES  |     | NULL    |       |
| RegistrationDate | date        | YES  |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
9 rows in set (0.03 sec)
```

2. Vehicle Table:
Query:
-> create table vehicle(VehicleID int primary key auto_increment,Model varchar(60),Make varchar(60),Year int,Color varchar(60),RegistrationNumber varchar(60) unique,Availability bit,DailyRate decimal(10,2));

```
mysql> create table vehicle(VehicleID int primary key auto_increment,Model varchar(60),Make varchar(60),Year int,Color varchar(60),RegistrationNumber varcha
r(60) unique,Availability bit,DailyRate decimal(10,2));
Query OK, 0 rows affected (0.24 sec)

mysql> desc vehicle;
+------------------+--------------+------+-----+---------+----------------+
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| VehicleID        | int          | NO   | PRI | NULL    | auto_increment |
| Model            | varchar(60)  | YES  |     | NULL    |                |
| Make             | varchar(60)  | YES  |     | NULL    |                |
| Year             | int          | YES  |     | NULL    |                |
| Color            | varchar(60)  | YES  |     | NULL    |                |
| RegistrationNumber | varchar(60)  | YES  | UNI | NULL    |                |
| Availability     | bit(1)       | YES  |     | NULL    |                |
| DailyRate        | decimal(10,2) | YES  |     | NULL    |                |
+------------------+--------------+------+-----+---------+----------------+
8 rows in set (0.05 sec)
```

3. Reservation Table:

Query:
-> create table reservation(ReservationID int primary key auto_increment,CustomerID int,VehicleID int,StartDate datetime,EndDate datetime,TotalCost decimal(10,2),Status varchar(60), foreign key (CustomerID) references Customer(CustomerID),foreign key (VehicleID) references Vehicle(VehicleID));
Query OK, 0 rows affected (0.26 sec)

```
mysql> create table reservation(ReservationID int primary key auto_increment,CustomerID int,VehicleID int,StartDate datetime,EndDate
datetime,TotalCost decimal(10,2),Status varchar(60), foreign key (CustomerID) references Customer(CustomerID),foreign key (VehicleID)
 references Vehicle(VehicleID));
Query OK, 0 rows affected (0.26 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        | varchar(60)   | YES  |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
7 rows in set (0.01 sec)
```

4. Admin Table:

Query:
-> create table Admin(AdminID int primary key auto_increment,FirstName varchar(60),LastName varchar(60), Email varchar(60),PhoneNumber varchar(11),Username varchar(100) unique,Password varchar(200),Role varchar(60),JoinDate date);
Query OK, 0 rows affected (0.26 sec)

```
mysql> create table Admin(AdminID int primary key auto_increment,FirstName v
archar(60),LastName varchar(60), Email varchar(60),PhoneNumber varchar(11),U
sername varchar(100) unique,Password varchar(200),Role varchar(60),JoinDate
date);
Query OK, 0 rows affected (0.26 sec)

mysql> desc Admin;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| AdminID     | int          | NO   | PRI | NULL    | auto_increment |
| FirstName   | varchar(60)  | YES  |     | NULL    |                |
| LastName    | varchar(60)  | YES  |     | NULL    |                |
| Email       | varchar(60)  | YES  |     | NULL    |                |
| PhoneNumber | varchar(11)  | YES  |     | NULL    |                |
| Username    | varchar(100) | YES  | UNI | NULL    |                |
| Password    | varchar(200) | YES  |     | NULL    |                |
| Role        | varchar(60)  | YES  |     | NULL    |                |
| JoinDate    | date         | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
9 rows in set (0.04 sec)
```

NOTE: *Inserting sample data into SQL tables ensures the database schema functions correctly before application integration, providing 15 unique records per table for key entities such as Customer, Vehicle, Admin, and Reservation. This enables thorough testing, reliable query validation, and a smoother development and debugging experience.*

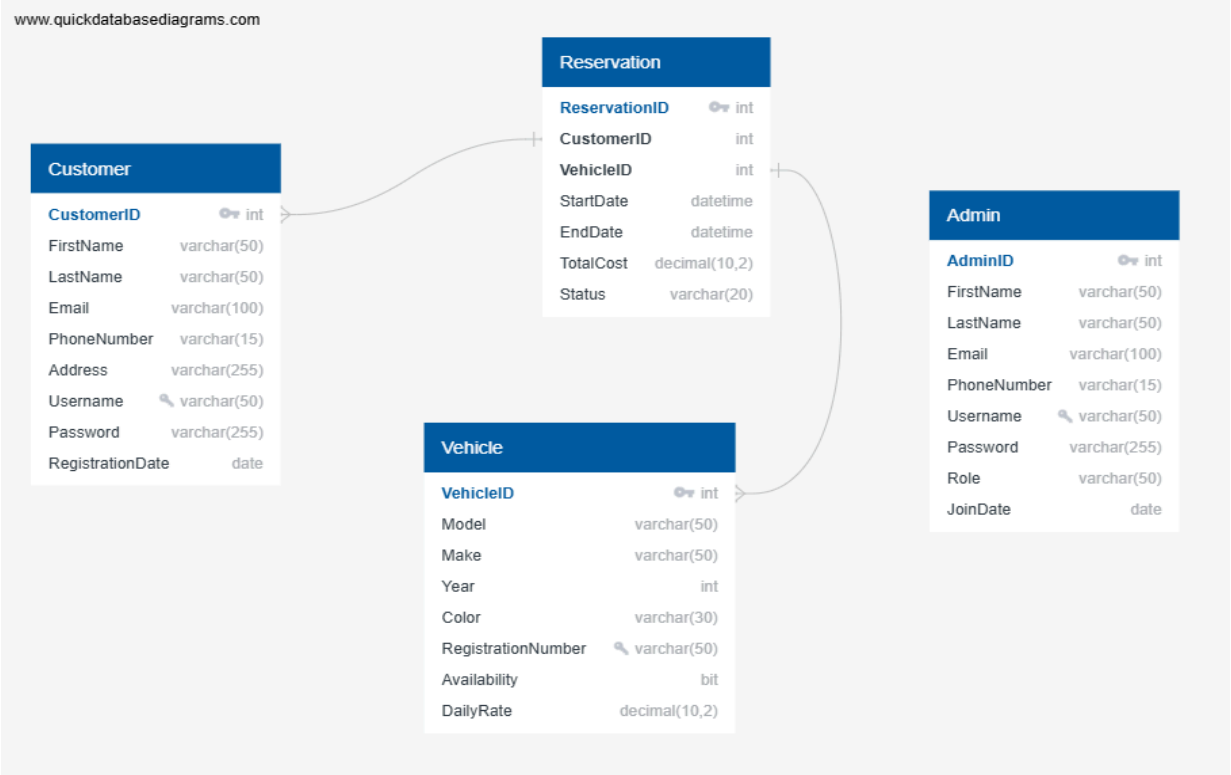
```
mysql> select * from customer;
+------------+-----------+----------+-------------------------+-------------+-------------------------+-----------+-----------+------------------+
| CustomerID | FirstName | LastName | Email                   | PhoneNumber | Address                 | Username  | Password  | RegistrationDate |
+------------+-----------+----------+-------------------------+-------------+-------------------------+-----------+-----------+------------------+
|          1 | Ravi      | Sharma   | ravi.sharma@gmail.com   | 9876543210  | 11 MG Road, Pune        | ravi01    | ravi@123  | 2024-06-01       |
|          2 | Anjali    | Mehra    | anjali.mehra@yahoo.com  | 9823456789  | 22 LBS Marg, Mumbai     | anjalim   | anjali@123| 2024-06-02       |
|          3 | Karthik   | Rao      | karthik.rao@gmail.com   | 9845612345  | 3 Gandhi Street, Hyderabad | karthikrao | kr@2024 | 2024-06-03       |
|          4 | Sneha     | Pillai   | sneha.pillai@gmail.com  | 9812345678  | 45 Brigade Rd, Bengaluru | snehap    | sneha#pass| 2024-06-04       |
|          5 | Amit      | Verma    | amit.verma@gmail.com    | 9934567890  | 67 Nehru Nagar, Delhi   | amitv     | amit1234  | 2024-06-05       |
|          6 | Nisha     | Patil    | nisha.patil@gmail.com   | 9988776655  | 29 Boat Club Rd, Pune   | nishapatil| nish@987  | 2024-06-06       |
|          7 | Rahul     | Desai    | rahul.desai@yahoo.com   | 9867543210  | 88 Juhu Beach Rd, Mumbai | rahuldesai| rahul#456 | 2024-06-07       |
|          8 | Pooja     | Yadav    | pooja.yadav@gmail.com   | 9898989898  | 15 Kalindi Kunj, Lucknow | poojay    | poo@pass  | 2024-06-08       |
|          9 | Rakesh    | Nair     | rakesh.nair@gmail.com   | 9743210123  | 91 Kochi Bypass         | rakeshn   | rakesh12  | 2024-06-09       |
|         10 | Divya     | Reddy    | divya.reddy@gmail.com   | 9767891234  | 5 Jubilee Hills, Hyderabad | divyar    | divya@pass| 2024-06-10       |
|         11 | Arjun     | Bhatt    | arjun.bhatt@gmail.com   | 9834567891  | 10 Shastri Nagar, Jaipur | arjunbhatt| ajb456    | 2024-06-11       |
|         12 | Meena     | Kapoor   | meena.kapoor@gmail.com  | 9807654321  | 73 Rajouri Garden, Delhi | meenak    | meena@321 | 2024-06-12       |
|         13 | Santosh   | Joshi    | santosh.joshi@gmail.com | 9798765432  | 19 Camp Area, Nagpur    | santoshj  | sj@pass   | 2024-06-13       |
|         14 | Isha      | Chatterjee | isha.cj@gmail.com     | 9754321876  | 88 Bhowanipur, Kolkata  | ishacj    | ic@2024   | 2024-06-14       |
|         15 | Vikram    | Bansal   | vikram.bansal@gmail.com | 9789012345  | 11 Vikas Puri, Delhi    | vikramb   | vbansal   | 2024-06-15       |
+------------+-----------+----------+-------------------------+-------------+-------------------------+-----------+-----------+------------------+
15 rows in set (0.00 sec)
```

```
mysql> select * from vehicle;
+-----------+---------------+----------+------+--------+--------------------+--------------------------+-----------+
| VehicleID | Model         | Make     | Year | Color  | RegistrationNumber | Availability             | DailyRate |
+-----------+---------------+----------+------+--------+--------------------+--------------------------+-----------+
|         1 | Innova Crysta | Toyota   | 2022 | White  | MH01AB1234         | 0x01                     |  12500.00 |
|         2 | XUV700        | Mahindra | 2023 | Black  | DL05CD2345         | 0x01                     |  13999.00 |
|         3 | City          | Honda    | 2021 | Silver | KA03EF3456         | 0x01                     |  10500.00 |
|         4 | Verna         | Hyundai  | 2022 | Red    | TN07GH4567         | 0x01                     |  11000.00 |
|         5 | Kushaq        | Skoda    | 2023 | Blue   | GJ09IJ5678         | 0x01                     |   9800.00 |
|         6 | Creta         | Hyundai  | 2021 | Grey   | MH12KL6789         | 0x00                     |  11500.00 |
|         7 | Fortuner      | Toyota   | 2023 | White  | DL10MN7890         | 0x01                     |  15500.00 |
|         8 | Thar          | Mahindra | 2022 | Black  | KA05OP8901         | 0x01                     |  13000.00 |
|         9 | Compass       | Jeep     | 2021 | Red    | KL11QR9012         | 0x01                     |  14200.00 |
|        10 | Seltos        | Kia      | 2022 | Blue   | TN22ST0123         | 0x01                     |  10800.00 |
|        11 | i20           | Hyundai  | 2020 | Silver | UP32UV1234         | 0x01                     |   9500.00 |
|        12 | Altroz        | Tata     | 2021 | Gold   | WB19WX2345         | 0x01                     |   9200.00 |
|        13 | Brezza        | Maruti   | 2022 | White  | RJ14YZ3456         | 0x01                     |   9900.00 |
|        14 | Scorpio-N     | Mahindra | 2023 | Black  | AP10ZA4567         | 0x01                     |  14900.00 |
|        15 | Venue         | Hyundai  | 2022 | Grey   | CH01XY5678         | 0x01                     |  10200.00 |
+-----------+---------------+----------+------+--------+--------------------+--------------------------+-----------+
15 rows in set (0.00 sec)
```

```
mysql> select * from reservation;
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
| ReservationID | CustomerID | VehicleID | StartDate           | EndDate             | TotalCost | Status    |
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
|             1 |          1 |         1 | 2025-06-20 09:00:00 | 2025-06-22 09:00:00 |  25000.00 | confirmed |
|             2 |          2 |         3 | 2025-06-21 10:00:00 | 2025-06-23 10:00:00 |  21000.00 | confirmed |
|             3 |          3 |         5 | 2025-06-19 14:00:00 | 2025-06-20 14:00:00 |   9800.00 | pending   |
|             4 |          4 |         2 | 2025-06-17 08:00:00 | 2025-06-18 08:00:00 |  13999.00 | confirmed |
|             5 |          5 |         4 | 2025-06-18 16:00:00 | 2025-06-20 16:00:00 |  22000.00 | completed |
|             6 |          6 |         6 | 2025-06-21 10:00:00 | 2025-06-23 10:00:00 |  23000.00 | confirmed |
|             7 |          7 |         7 | 2025-06-24 09:00:00 | 2025-06-25 09:00:00 |  15500.00 | pending   |
|             8 |          8 |         8 | 2025-06-19 12:00:00 | 2025-06-21 12:00:00 |  26000.00 | confirmed |
|             9 |          9 |        10 | 2025-06-20 07:00:00 | 2025-06-22 07:00:00 |  21600.00 | confirmed |
|            10 |         10 |         9 | 2025-06-22 11:00:00 | 2025-06-24 11:00:00 |  28400.00 | confirmed |
|            11 |         11 |        11 | 2025-06-21 10:00:00 | 2025-06-22 10:00:00 |   9500.00 | cancelled |
|            12 |         12 |        13 | 2025-06-23 15:00:00 | 2025-06-25 15:00:00 |  29800.00 | confirmed |
|            13 |         13 |        14 | 2025-06-24 13:00:00 | 2025-06-26 13:00:00 |  19800.00 | pending   |
|            14 |         14 |        12 | 2025-06-22 08:00:00 | 2025-06-23 08:00:00 |   9200.00 | completed |
|            15 |         15 |        15 | 2025-06-25 09:00:00 | 2025-06-27 09:00:00 |  20400.00 | confirmed |
+---------------+------------+-----------+---------------------+---------------------+-----------+-----------+
15 rows in set (0.00 sec)
```

```
mysql> select * from admin;
+---------+-----------+----------+-------------------------+-------------+---------------+-------------+---------------+------------+
| AdminID | FirstName | LastName | Email                   | PhoneNumber | Username      | Password    | Role          | JoinDate   |
+---------+-----------+----------+-------------------------+-------------+---------------+-------------+---------------+------------+
|       1 | Sunita    | Nair     | sunita.nair@carconnect.com | 9123456700 | adminuser0001 | admin@123   | super admin   | 2023-01-01 |
|       2 | Alok      | Mishra   | alok.mishra@carconnect.com | 9123456701 | adminuser0002 | admin@456   | fleet manager | 2023-01-10 |
|       3 | Farah     | Ahmed    | farah.ahmed@carconnect.com | 9123456702 | adminuser0003 | admin@farah | admin         | 2023-01-15 |
|       4 | Nitin     | Kumar    | nitin.kumar@carconnect.com | 9123456703 | adminuser0004 | nitinpass   | super admin   | 2023-01-20 |
|       5 | Vidya     | Iyer     | vidya.iyer@carconnect.com | 9123456704 | adminuser0005 | iyer@admin  | admin         | 2023-02-01 |
|       6 | Sameer    | Patel    | sameer.patel@carconnect.com | 9123456705 | adminuser0006 | sameer2024 | fleet manager | 2023-02-05 |
|       7 | Neha      | Rastogi  | neha.rastogi@carconnect.com | 9123456706 | adminuser0007 | neha@987    | support admin | 2023-02-10 |
|       8 | Arvind    | Chauhan  | arvind.ch@carconnect.com | 9123456707 | adminuser0008 | adminarvind | fleet manager | 2023-02-15 |
|       9 | Lata      | Kapoor   | lata.kapoor@carconnect.com | 9123456708 | adminuser0009 | latakap123  | admin         | 2023-03-01 |
|      10 | Rajan     | Joshi    | rajan.joshi@carconnect.com | 9123456709 | adminuser0010 | joshi@321   | support admin | 2023-03-10 |
|      11 | Mehul     | Seth     | mehul.seth@carconnect.com | 9123456710 | adminuser0011 | mehul2023   | admin         | 2023-03-15 |
|      12 | Priya     | Garg     | priya.garg@carconnect.com | 9123456711 | adminuser0012 | pgadmin@1   | super admin   | 2023-03-20 |
|      13 | Rishi     | Malik    | rishi.malik@carconnect.com | 9123456712 | adminuser0013 | rishi!admin | fleet manager | 2023-04-01 |
|      14 | Geeta     | Das      | geeta.das@carconnect.com | 9123456713 | adminuser0014 | geetadas123 | admin         | 2023-04-05 |
|      15 | Sandeep   | Singh    | sandeep.singh@carconnect.com | 9123456714 | adminuser0015 | ssingh@99   | support admin | 2023-04-10 |
+---------+-----------+----------+-------------------------+-------------+---------------+-------------+---------------+------------+
15 rows in set (0.00 sec)
```

ENTITY RELATIONSHIP DIAGRAM:
`



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

Classes:

- Customer:
  - Properties: CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate
  - Methods: Authenticate(password)

customer.py:

```
from datetime import datetime

class Customer:
    def __init__(self, customer_id=None, first_name="", last_name="",
email="", phone_number="", address="", username="", password="",
registration_date=None):
        self.__customer_id = customer_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
```

```
        self.__phone_number = phone_number
        self.__address = address
        self.__username = username
        self.__password = password
        self.__registration_date = registration_date if
registration_date else datetime.now()

    # Getters
    def get_customer_id(self): return self.__customer_id
    def get_first_name(self): return self.__first_name
    def get_last_name(self): return self.__last_name
    def get_email(self): return self.__email
    def get_phone_number(self): return self.__phone_number
    def get_address(self): return self.__address
    def get_username(self): return self.__username
    def get_password(self): return self.__password
    def get_registration_date(self): return self.__registration_date

    # Setters



    def set_customer_id(self, customer_id): self.__customer_id =
customer_id
    def set_first_name(self, first_name): self.__first_name =
first_name
    def set_last_name(self, last_name): self.__last_name = last_name
    def set_email(self, email): self.__email = email
    def set_phone_number(self, phone_number): self.__phone_number =
phone_number
    def set_address(self, address): self.__address = address
    def set_username(self, username): self.__username = username
    def set_password(self, password): self.__password = password
    def set_registration_date(self, registration_date):
self.__registration_date = registration_date

    # Method
    def authenticate(self, input_password):
        return self.__password == input_password
```

EXPLANATION:

The Customer class stores user details like name, email, and username. It includes private fields with getters/setters and a method authenticate() to verify login. It's used across the project for customer profile, login, and update features.

• Vehicle:

  • Properties: VehicleID, Model, Make, Year, Colour, RegistrationNumber, Availability, DailyRate

```
class Vehicle:
    def __init__(self, vehicle_id=None, model="", make="", year=0,
color="", registration_number="", availability=True, daily_rate=0.0):
        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

    # Getters
    def get_vehicle_id(self): return self.__vehicle_id
    def get_model(self): return self.__model
    def get_make(self): return self.__make
    def get_year(self): return self.__year
    def get_color(self): return self.__color
    def get_registration_number(self): return
self.__registration_number
    def is_available(self): return self.__availability
    def get_daily_rate(self): return self.__daily_rate

    # Setters
    def set_vehicle_id(self, vehicle_id): self.__vehicle_id =
vehicle_id
    def set_model(self, model): self.__model = model
    def set_make(self, make): self.__make = make
    def set_year(self, year): self.__year = year
    def set_color(self, color): self.__color = color
    def set_registration_number(self, registration_number):
self.__registration_number = registration_number
    def set_availability(self, availability): self.__availability =
availability
```

```
    def set_daily_rate(self, daily_rate): self.__daily_rate =
daily_rate
```

EXPLANATION:

The Vehicle class stores car details like model, make, year, and availability status. It helps track which cars are available, rented, or removed. It includes private fields with full getter/setter methods and supports adding, updating, and deleting vehicle records.

• Reservation:

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

• Methods: CalculateTotalCost()

```
from datetime import datetime

class Reservation:
    def __init__(self, reservation_id=None, customer_id=None,
vehicle_id=None, start_date=None, end_date=None, total_cost=0.0,
status="Booked"):
        self.__reservation_id = reservation_id
        self.__customer_id = customer_id
        self.__vehicle_id = vehicle_id
        self.__start_date = start_date if start_date else
datetime.now()
        self.__end_date = end_date if end_date else datetime.now()
        self.__total_cost = total_cost
        self.__status = status

    # Getters
    def get_reservation_id(self): return self.__reservation_id
    def get_customer_id(self): return self.__customer_id
    def get_vehicle_id(self): return self.__vehicle_id
    def get_start_date(self): return self.__start_date
    def get_end_date(self): return self.__end_date
    def get_total_cost(self): return self.__total_cost
    def get_status(self): return self.__status

    # Setters
    def set_reservation_id(self, reservation_id): self.__reservation_id
= reservation_id
    def set_customer_id(self, customer_id): self.__customer_id =
customer_id
```

```
    def set_vehicle_id(self, vehicle_id): self.__vehicle_id =
vehicle_id
    def set_start_date(self, start_date): self.__start_date =
start_date
    def set_end_date(self, end_date): self.__end_date = end_date
    def set_total_cost(self, total_cost): self.__total_cost =
total_cost
    def set_status(self, status): self.__status = status
    def calculate_total_cost(self, daily_rate: float):
        days = (self.__end_date - self.__start_date).days
        days = max(1, days)  # minimum 1 day
        self.__total_cost = daily_rate * days
```

EXPLANATION:

The Reservation class stores booking details between customers and vehicles. It includes dates, status, and a method to calculate total cost using calculate_total_cost(). Used in vehicle booking, viewing, and cancellation features.

- Admin:
- Properties: AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate
- Methods: Authenticate(password)

```
from datetime import datetime

class Admin:
    def __init__(self, admin_id=None, first_name="", last_name="",
email="", phone_number="", username="", password="", role="",
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 if join_date else datetime.now()

    # Getters
    def get_admin_id(self): return self.__admin_id
    def get_first_name(self): return self.__first_name
    def get_last_name(self): return self.__last_name
    def get_email(self): return self.__email
```

```
    def get_phone_number(self): return self.__phone_number
    def get_username(self): return self.__username
    def get_password(self): return self.__password
    def get_role(self): return self.__role
    def get_join_date(self): return self.__join_date

    # Setters
    def set_admin_id(self, admin_id): self.__admin_id = admin_id
    def set_first_name(self, first_name): self.__first_name =
first_name
    def set_last_name(self, last_name): self.__last_name = last_name
    def set_email(self, email): self.__email = email
    def set_phone_number(self, phone_number): self.__phone_number =
phone_number
    def set_username(self, username): self.__username = username
    def set_password(self, password): self.__password = password
    def set_role(self, role): self.__role = role
    def set_join_date(self, join_date): self.__join_date = join_date

    # Method: Authenticate password
    def authenticate(self, input_password):
        return self.__password == input_password
```

EXPLANATION:

The Admin class manages system administrators. It stores login, contact, and role info, with a method to authenticate passwords. It supports login, viewing reports, adding vehicles, and managing reservations from the admin panel.

• CustomerService (implements ICustomerService):
    • Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

```
from dao.interfaces.icustomer_service import ICustomerService
from entity.customer import Customer
from util.db_conn_util import DBConnUtil

class CustomerService(ICustomerService):

    def get_customer_by_id(self, customer_id: int) -> Customer | None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Customer WHERE CustomerID = %s",
      (customer_id,))
```

```
        row = cursor.fetchone()
        conn.close()

        if row:
            return Customer(
                customer_id=row[0],
                first_name=row[1],
                last_name=row[2],
                email=row[3],
                phone_number=row[4],
                address=row[5],
                username=row[6],
                password=row[7],
                registration_date=row[8]
            )
        return None

    def get_customer_by_username(self, username: str) -> Customer |
      None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Customer WHERE Username = %s",
      (username,))
        row = cursor.fetchone()
        conn.close()

        if row:
            return Customer(
                customer_id=row[0],
                first_name=row[1],
                last_name=row[2],
                email=row[3],
                phone_number=row[4],
                address=row[5],
                username=row[6],
                password=row[7],
                registration_date=row[8]
            )
        return None

    def register_customer(self, customer: Customer) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
```

```
        cursor.execute("""
            INSERT INTO Customer (FirstName, LastName, Email,
      PhoneNumber, Address, Username, Password, RegistrationDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            customer.get_first_name(),
            customer.get_last_name(),
            customer.get_email(),
            customer.get_phone_number(),
            customer.get_address(),
            customer.get_username(),
            customer.get_password(),
            customer.get_registration_date()
        ))
        conn.commit()
        conn.close()

    def update_customer(self, customer: Customer) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE Customer
            SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber
      = %s, Address = %s,
                Username = %s, Password = %s, RegistrationDate = %s
            WHERE CustomerID = %s
        """, (
            customer.get_first_name(),
            customer.get_last_name(),
            customer.get_email(),
            customer.get_phone_number(),
            customer.get_address(),
            customer.get_username(),
            customer.get_password(),
            customer.get_registration_date(),
            customer.get_customer_id()
        ))
        conn.commit()
        conn.close()

    def delete_customer(self, customer_id: int) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
```

```
        cursor.execute("DELETE FROM Customer WHERE CustomerID = %s",
      (customer_id,))
        conn.commit()
        conn.close()
```

EXPLANATION:

CustomerService handles all customer-related database operations — registration, login, update, and deletion. It implements the ICustomerService interface and connects to the DB using DBConnUtil. All logic is isolated here so the UI or main module remains clean and focused.

• VehicleService (implements IVehicleService):

  • Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

```
from dao.interfaces.ivehicle_service import IVehicleService
from entity.vehicle import Vehicle
from util.db_conn_util import DBConnUtil
from exception.vehicle_not_found_exception import
VehicleNotFoundException

class VehicleService(IVehicleService):

    def get_vehicle_by_id(self, vehicle_id: int) -> Vehicle:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Vehicle WHERE VehicleID = %s",
(vehicle_id,))
        row = cursor.fetchone()
        conn.close()

        if row:
            return Vehicle(*row)
        else:
            raise VehicleNotFoundException(f" Vehicle with ID
{vehicle_id} does not exist.")

    def get_available_vehicles(self) -> list[Vehicle]:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Vehicle WHERE Availability =
TRUE")
```

```
        rows = cursor.fetchall()
        conn.close()
        return [Vehicle(*row) for row in rows]

    def add_vehicle(self, vehicle: Vehicle) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO Vehicle (Model, Make, Year, Color,
RegistrationNumber, Availability, DailyRate)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (
            vehicle.get_model(),
            vehicle.get_make(),
            vehicle.get_year(),
            vehicle.get_color(),
            vehicle.get_registration_number(),
            vehicle.is_available(),
            vehicle.get_daily_rate()
        ))
        conn.commit()
        conn.close()

    def update_vehicle(self, vehicle: Vehicle) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE Vehicle
            SET Model = %s, Make = %s, Year = %s, Color = %s,
RegistrationNumber = %s, Availability = %s, DailyRate = %s
            WHERE VehicleID = %s
        """, (
            vehicle.get_model(),
            vehicle.get_make(),
            vehicle.get_year(),
            vehicle.get_color(),
            vehicle.get_registration_number(),
            vehicle.is_available(),
            vehicle.get_daily_rate(),
            vehicle.get_vehicle_id()
        ))
        conn.commit()
        conn.close()
```

```
    def delete_vehicle(self, vehicle_id: int) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("DELETE FROM Vehicle WHERE VehicleID = %s",
(vehicle_id,))
        conn.commit()
        conn.close()
```

EXPLANATION:

The VehicleService manages vehicle records in the database — from adding a new car to updating its details or removing it. It also fetches available cars or gets a vehicle by ID. It implements IVehicleService and is essential for booking and admin management.

• ReservationService (implements IReservationService):
    • Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation

```
from dao.interfaces.ireservation_service import IReservationService
from entity.reservation import Reservation
from util.db_conn_util import DBConnUtil
from exception.reservation_exception import ReservationException

class ReservationService(IReservationService):

    def get_reservation_by_id(self, reservation_id: int) -> Reservation
| None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Reservation WHERE ReservationID =
%s", (reservation_id,))
        row = cursor.fetchone()
        conn.close()

        if row:
            return Reservation(*row)
        return None

    def get_reservations_by_customer_id(self, customer_id: int) ->
list[Reservation]:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
```

```
        cursor.execute("SELECT * FROM Reservation WHERE CustomerID =
%s", (customer_id,))
        rows = cursor.fetchall()
        conn.close()

        return [Reservation(*row) for row in rows]

    def create_reservation(self, reservation: Reservation) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()

        # Check if the vehicle is already reserved during that time
period
        cursor.execute("""
            SELECT * FROM Reservation
            WHERE VehicleID = %s AND Status = 'Active' AND (
                (StartDate <= %s AND EndDate >= %s) OR
                (StartDate <= %s AND EndDate >= %s) OR
                (%s <= StartDate AND %s >= EndDate)
            )
        """, (
            reservation.get_vehicle_id(),
            reservation.get_start_date(), reservation.get_start_date(),
            reservation.get_end_date(), reservation.get_end_date(),
            reservation.get_start_date(), reservation.get_end_date()
        ))

        overlap = cursor.fetchone()
        if overlap:
            conn.close()
            raise ReservationException(" Cannot reserve: This vehicle
is already booked for the selected period.")

        # If no overlap, insert reservation
        cursor.execute("""
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate,
EndDate, TotalCost, Status)
            VALUES (%s, %s, %s, %s, %s, %s)
        """, (
            reservation.get_customer_id(),
            reservation.get_vehicle_id(),
            reservation.get_start_date(),
            reservation.get_end_date(),
```

```
            reservation.get_total_cost(),
            reservation.get_status()
        ))

        conn.commit()
        conn.close()

    def update_reservation(self, reservation: Reservation) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE Reservation
            SET StartDate = %s, EndDate = %s, TotalCost = %s, Status =
%s
            WHERE ReservationID = %s
        """, (
            reservation.get_start_date(),
            reservation.get_end_date(),
            reservation.get_total_cost(),
            reservation.get_status(),
            reservation.get_reservation_id()
        ))
        conn.commit()
        conn.close()

    def cancel_reservation(self, reservation_id: int) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("DELETE FROM Reservation WHERE ReservationID =
%s", (reservation_id,))
        conn.commit()
        conn.close()
```

EXPLANATION:

The ReservationService handles all bookings: creating, updating, and cancelling. It fetches reservations by ID or customer ID. Total cost is calculated using duration and daily rate. Errors like invalid dates throw ReservationException. It connects to the DB using DBConnUtil.

• AdminService (implements IAdminService):

• Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

```
from dao.interfaces.iadmin_service import IAdminService
from entity.admin import Admin
from util.db_conn_util import DBConnUtil
from exception.admin_not_found_exception import AdminNotFoundException

class AdminService(IAdminService):

    def get_admin_by_id(self, admin_id: int) -> Admin | None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Admin WHERE AdminID = %s",
(admin_id,))
        row = cursor.fetchone()
        conn.close()

        if row:
            return Admin(
                admin_id=row[0],
                first_name=row[1],
                last_name=row[2],
                email=row[3],
                phone_number=row[4],
                username=row[5],
                password=row[6],
                role=row[7],
                join_date=row[8]
            )
        return None

    def get_admin_by_username(self, username: str) -> Admin:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Admin WHERE Username = %s",
(username,))
        row = cursor.fetchone()
        conn.close()

        if row:
            return Admin(
```

```
                admin_id=row[0],
                first_name=row[1],
                last_name=row[2],
                email=row[3],
                phone_number=row[4],
                username=row[5],
                password=row[6],
                role=row[7],
                join_date=row[8]
            )
        else:
            raise AdminNotFoundException(f" Admin with username
'{username}' does not exist.")

    def register_admin(self, admin: Admin) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber,
Username, Password, Role, JoinDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            admin.get_first_name(),
            admin.get_last_name(),
            admin.get_email(),
            admin.get_phone_number(),
            admin.get_username(),
            admin.get_password(),
            admin.get_role(),
            admin.get_join_date()
        ))
        conn.commit()
        conn.close()

    def update_admin(self, admin: Admin) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE Admin
            SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber
= %s,
                Username = %s, Password = %s, Role = %s, JoinDate = %s
            WHERE AdminID = %s
```

```
        """, (
            admin.get_first_name(),
            admin.get_last_name(),
            admin.get_email(),
            admin.get_phone_number(),
            admin.get_username(),
            admin.get_password(),
            admin.get_role(),
            admin.get_join_date(),
            admin.get_admin_id()
        ))
        conn.commit()
        conn.close()

    def delete_admin(self, admin_id: int) -> None:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("DELETE FROM Admin WHERE AdminID = %s",
(admin_id,))
        conn.commit()
        conn.close()
```

EXPLANATION:

The AdminService manages admin users. It fetches by ID or username, registers new admins, updates details, or deletes the account. It's mostly used during admin login and inside the admin dashboard. All errors are cleanly handled using AdminNotFoundException.

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

```
import mysql.connector
from util.db_property_util import DBPropertyUtil
from exception.database_connection_exception import
DatabaseConnectionException

class DBConnUtil:
    @staticmethod
    def get_connection():
        try:
            props =
DBPropertyUtil.get_connection_properties("config/db.properties")
```

```
            return mysql.connector.connect(
                host=props['server'],
                database=props['database'],
                user=props['username'],
                password=props['password']
            )
        except Exception as e:
            raise DatabaseConnectionException(f" Database connection
failed: {str(e)}")
```

```
[database]
driver={MySQL ODBC 8.0 ANSI Driver}
server=localhost
database=carconnect
username=root
password=rootadmin
```

EXPLANATION:

The DBConnUtil acts as DatabaseContext. It centralizes and secures connection logic using credentials from db.properties. If the connection fails, it raises a DatabaseConnectionException. This keeps all DB interactions stable and reusable across services like Customer, Vehicle, and Reservation.

• AuthenticationService:
• A class responsible for handling user authentication.

```
from dao.implementations.customer_service import CustomerService
from dao.implementations.admin_service import AdminService

from exception.authentication_exception import AuthenticationException
from exception.admin_not_found_exception import AdminNotFoundException

class AuthenticationService:

    @staticmethod
    def authenticate_customer(username, password):
        customer_service = CustomerService()
        customer = customer_service.get_customer_by_username(username)

        if customer and customer.authenticate(password):
            return customer
```

```
        else:
            raise AuthenticationException(" Invalid customer username
or password.")

    @staticmethod
    def authenticate_admin(username, password):
        admin_service = AdminService()

        try:
            admin = admin_service.get_admin_by_username(username)
        except AdminNotFoundException as e:
            raise AuthenticationException(str(e))  # Re-raise with
clear message

        if admin.authenticate(password):
            return admin
        else:
            raise AuthenticationException(" Invalid admin password.")
```

EXPLANATION:

The AuthenticationService verifies login for both customers and admins. It calls respective services, checks passwords using the authenticate() method, and throws AuthenticationException if credentials are wrong. This makes login logic reusable and clean across the whole app.

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

```
from util.db_conn_util import DBConnUtil

class ReportGenerator:

    @staticmethod
    def total_reservations():
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM Reservation")
        count = cursor.fetchone()[0]
        conn.close()
        return count

    @staticmethod
```

```
    def total_revenue():
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT SUM(TotalCost) FROM Reservation")
        revenue = cursor.fetchone()[0] or 0
        conn.close()
        return revenue

    @staticmethod
    def most_booked_vehicle():
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT VehicleID, COUNT(*) AS cnt
            FROM Reservation
            GROUP BY VehicleID
            ORDER BY cnt DESC
            LIMIT 1
        """)
        result = cursor.fetchone()
        conn.close()
        return result  # (VehicleID, Count)
```

EXPLANATION:

The ReportGenerator is used by admins to see key stats like total bookings, revenue, and the most rented vehicle. It directly queries the database and returns results using static methods. Helps make smart business decisions with simple analytics.

Interfaces:

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

```
from abc import ABC, abstractmethod
from entity.customer import Customer

class ICustomerService(ABC):
```

```
    @abstractmethod
    def get_customer_by_id(self, customer_id: int) -> Customer:
        pass

    @abstractmethod
    def get_customer_by_username(self, username: str) -> Customer:
        pass

    @abstractmethod
    def register_customer(self, customer: Customer) -> None:
        pass

    @abstractmethod
    def update_customer(self, customer: Customer) -> None:
        pass

    @abstractmethod
    def delete_customer(self, customer_id: int) -> None:
        pass
```

EXPLANATION:

The ICustomerService defines the contract for all customer-related operations. It includes methods for fetching, registering, updating, and deleting customers. It helps maintain structure and ensures any class like CustomerService follows this rule.

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

```
from abc import ABC, abstractmethod
from entity.vehicle import Vehicle

class IVehicleService(ABC):

    @abstractmethod
    def get_vehicle_by_id(self, vehicle_id: int) -> Vehicle:
        pass

    @abstractmethod
    def get_available_vehicles(self) -> list[Vehicle]:
```

```
        pass

    @abstractmethod
    def add_vehicle(self, vehicle: Vehicle) -> None:
        pass

    @abstractmethod
    def update_vehicle(self, vehicle: Vehicle) -> None:
        pass

    @abstractmethod
    def delete_vehicle(self, vehicle_id: int) -> None:
        pass
```

EXPLANATION:

The IVehicleService is an abstract interface that defines what any vehicle service class must do. It ensures every service includes logic to add, fetch, update, or remove vehicles. This keeps the structure neat and lets VehicleService follow a clean blueprint.

• IReservationService:
• GetReservationById(reservationId)
• GetReservationsByCustomerId(customerId)
• CreateReservation(reservationData)
• UpdateReservation(reservationData)
• CancelReservation(reservationId)

```
from abc import ABC, abstractmethod
from entity.reservation import Reservation

class IReservationService(ABC):

    @abstractmethod
    def get_reservation_by_id(self, reservation_id: int) ->
Reservation:
        pass

    @abstractmethod
    def get_reservations_by_customer_id(self, customer_id: int) ->
list[Reservation]:
        pass

    @abstractmethod
    def create_reservation(self, reservation: Reservation) -> None:
        pass
```

```
    @abstractmethod
    def update_reservation(self, reservation: Reservation) -> None:
        pass

    @abstractmethod
    def cancel_reservation(self, reservation_id: int) -> None:
        pass
```

EXPLANATION:

The IReservationService defines the structure for all reservation tasks like booking, updating, viewing, or canceling. It's implemented by ReservationService, keeping the logic modular, testable, and reusable. Makes reservation handling clean and predictable.

• IAdminService:
• GetAdminById(adminId)
• GetAdminByUsername(username)
• RegisterAdmin(adminData)
• UpdateAdmin(adminData)
• DeleteAdmin(adminId)

```
from abc import ABC, abstractmethod
from entity.admin import Admin

class IAdminService(ABC):

    @abstractmethod
    def get_admin_by_id(self, admin_id: int) -> Admin:
        pass

    @abstractmethod
    def get_admin_by_username(self, username: str) -> Admin:
        pass

    @abstractmethod
    def register_admin(self, admin: Admin) -> None:
        pass

    @abstractmethod
    def update_admin(self, admin: Admin) -> None:
        pass

    @abstractmethod
```

```
    def delete_admin(self, admin_id: int) -> None:
        pass
```

EXPLANATION:
The IAdminService outlines all admin operations: fetch, register, update, or delete an admin user. Any implementing class, like AdminService, must define these methods. It helps enforce structure, separation of concerns, and keeps admin logic clean and testable.

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.

1. **Connection String**

```
[database]
driver={MySQL ODBC 8.0 ANSI Driver}
server=localhost
database=carconnect
username=root
password=rootadmin
```

This holds all the details needed to connect to the MySQL DB, just like a connection string.

2. **Use DBConnUtil as a SqlConnection Equivalent**

```
import mysql.connector
from util.db_property_util import DBPropertyUtil
from exception.database_connection_exception import
DatabaseConnectionException

class DBConnUtil:
    @staticmethod
    def get_connection():
        try:
            props =
DBPropertyUtil.get_connection_properties("config/db.properties")
            return mysql.connector.connect(
                host=props['server'],
                database=props['database'],
                user=props['username'],
                password=props['password']
```

```
            )
        except Exception as e:
            raise DatabaseConnectionException(f" Database connection
failed: {str(e)}")
```

We use DBConnUtil in place of SqlConnection, loading credentials from db.properties. Every service connects to the database using this utility and executes queries via cursor.execute(). The Python equivalent of SqlCommand.

Custom Exceptions:

AuthenticationException:

• Thrown when there is an issue with user authentication.

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

1\CarConnect\exception\authentication_exception.py

```
class AuthenticationException(Exception):

    def __init__(self, message="Authentication failed: Invalid username
or password."):

        super().__init__(message)
```

1\CarConnect\util\authentication_service.py

```
class AuthenticationService:

    @staticmethod

    def authenticate_customer(username, password):

        customer_service = CustomerService()

        customer = customer_service.get_customer_by_username(username)

        if customer and customer.authenticate(password):

            return customer

        else:

            raise AuthenticationException(" Invalid customer username
or password.")
```

1\CarConnect\main\main_module.py

```
 try:

        customer =
AuthenticationService.authenticate_customer(username, password)

    except AuthenticationException as e:

        print(e)

        return
```

EXPLANATION:

AuthenticationException is raised when login fails due to incorrect credentials. It provides a clear and reusable error message for both customer and admin login flows. This helps keep the authentication code clean and meaningful for users.

OUTPUT TERMINAL:

```
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> python -m main.main_module
>>
=== Welcome to CarConnect ===
Are you a:
1. Customer
2. Admin
Enter choice (1 or 2): 1
=== Customer Login ===
Username: meerapatel
Password: meeraaaaaaa
   Invalid customer username or password.
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect>
```

ReservationException:

• Thrown when there is an issue with reservations.

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

1\CarConnect\exception\reservation_exception.py

```
class ReservationException(Exception):
```

```
    def __init__(self, message="Reservation error occurred."):

        super().__init__(message)
```

1\CarConnect\dao\implementations\reservation_service.py

```
overlap = cursor.fetchone()

        if overlap:

            conn.close()

            raise ReservationException(" Cannot reserve: This vehicle
is already booked for the selected period.")
```

EXPLANATION:

ReservationException is raised when something goes wrong while booking a car. It helps catch cases like unavailable vehicles or invalid reservation dates, and shows a clear message to the user. Keeps the reservation logic smooth and error-free.

OUTPUT TERMINAL:

```
==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. View Available Vehicles by Date Range
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1-10): 7

   Top Featured Vehicle
Vehicle ID : 1
Model      : Civic
Booked     : 4 times

Start Date (YYYY-MM-DD): 2025-07-01
End Date (YYYY-MM-DD): 2025-07-05

   Available Vehicles:
3 | Jeep | Matte Black | ₹2100.00

Enter Vehicle ID to book: 1
   Cannot reserve: Vehicle is already booked in that time range.
```

VehicleNotFoundException:

• Thrown when a requested vehicle is not found.

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

1\CarConnect\exception\vehicle_not_found_exception.py

```
class VehicleNotFoundException(Exception):

    def __init__(self, message="Vehicle not found."):

        super().__init__(message)
```

1\CarConnect\dao\implementations\vehicle_service.py

```
def get_vehicle_by_id(self, vehicle_id: int) -> Vehicle:

        conn = DBConnUtil.get_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT * FROM Vehicle WHERE VehicleID = %s",
(vehicle_id,))

        row = cursor.fetchone()

        conn.close()

        if row:

            return Vehicle(*row)

        else:

            raise VehicleNotFoundException(f" Vehicle with ID
{vehicle_id} does not exist.")
```

1\CarConnect\main\main_module.py

```
elif choice == "5":

            try:

                vid = int(input("Enter Vehicle ID: "))

                v = vehicle_service.get_vehicle_by_id(vid)
```

```
                print("Vehicle Found:")
                print("Model:", v.get_model())
                print("Make:", v.get_make())
                print("Year:", v.get_year())
                print("Color:", v.get_color())
                print("Reg No:", v.get_registration_number())
                print("Available:", "Yes" if v.is_available() else
"No")
                print("Rate/day: ₹", v.get_daily_rate())

            except VehicleNotFoundException as ve:
                print(ve)
            except ValueError:
                print(" Invalid input. Please enter a valid vehicle
ID.")
```

EXPLANATION:

VehicleNotFoundException is raised when a user tries to access a vehicle that doesn't exist in the database. This ensures clean user feedback and prevents operations on missing or deleted vehicles.

TERMINAL OUTPUT:

```
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> python -m main.main_module
>>
=== Welcome to CarConnect ===
Are you a:
1. Customer
2. Admin
Enter choice (1 or 2): 1
=== Customer Login ===
Username: MEERAPATEL
Password: meera@123

   Welcome, Meera!


==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. Get Available Vehicles
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1-10): 5
Enter Vehicle ID: 9845
   Vehicle with ID 9845 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.

OUTPUT TERMINAL:

```
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> python -m main.main_module
>>
=== Welcome to CarConnect ===
Are you a:
1. Customer
2. Admin
Enter choice (1 or 2): 2
=== Admin Login ===
Admin Username: adminuser0003
Password: 68461654
   Invalid admin password.
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect>
```

Unit Testing:

Create NUnit test cases for the 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:

```
import pytest
```

```
from dao.implementations.customer_service import CustomerService

from dao.implementations.vehicle_service import VehicleService

from entity.customer import Customer

from entity.vehicle import Vehicle

from util.authentication_service import AuthenticationService

from exception.authentication_exception import AuthenticationException

from util.db_conn_util import DBConnUtil

def test_invalid_login():

    with pytest.raises(AuthenticationException):

        AuthenticationService.authenticate_customer("wronguser",
"wrongpass")

def test_update_customer_info():

    service = CustomerService()

    updated_customer = Customer(

        customer_id=1,

        first_name="UpdatedFirst",

        last_name="UpdatedLast",

        email="updated_email@example.com",

        phone_number="9876543210",

        address="Updated Address",

        username="updateduser",

        password="securepass123",

        registration_date="2024-01-01"

    )

    service.update_customer(updated_customer)
```

```
    fetched = service.get_customer_by_id(1)
    assert fetched.get_first_name() == "UpdatedFirst"
    assert fetched.get_email() == "updated_email@example.com"


def test_add_vehicle():
    service = VehicleService()
    conn = DBConnUtil.get_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Vehicle WHERE RegistrationNumber =
'TEST-1234'")
    conn.commit()
    conn.close()


    vehicle = Vehicle(
        make="TestBrand",
        model="ModelX",
        year=2022,
        color="Red",
        registration_number="TEST-1234",
        availability=True,
        daily_rate=2000.0
    )
    service.add_vehicle(vehicle)
    vehicles = service.get_all_vehicles()
    assert any(v.get_registration_number() == "TEST-1234" for v in
vehicles)
```

```python
def test_update_vehicle():
    service = VehicleService()
    vehicle = Vehicle(
        vehicle_id=1,  # check ID 1 exists
        make="UpdatedBrand",
        model="UpdatedModel",
        year=2023,
        color="Blue",
        registration_number="TEST-UPDATED-5678",  #unique,
        availability=True,
        daily_rate=2500.0
    )
    service.update_vehicle(vehicle)
    updated = service.get_vehicle_by_id(1)
    assert updated.get_make() == "UpdatedBrand"
    assert updated.get_daily_rate() == 2500.0
def test_get_available_vehicles():
    service = VehicleService()
    vehicles = service.get_available_vehicles()
    assert isinstance(vehicles, list)
    assert all(v.is_available() for v in vehicles)
def test_get_all_vehicles():
    service = VehicleService()
    vehicles = service.get_all_vehicles()
    assert isinstance(vehicles, list)
```

```
    assert len(vehicles) >= 1
```

$env:PYTHONPATH="."

pytest test/test_carconnect.py

```
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> pytest test/test_carconnect.py
>>
============================================================ test session starts ============================================================
platform win32 -- Python 3.11.2, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect
collected 6 items

test\test_carconnect.py ......                                                                                                      [100%]

============================================================ 6 passed in 0.34s =============================================================
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect>
```

-> VS CODE TERMINAL

```
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> $env:PYTHONPATH="."
PS C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect> pytest test/test_carconnect.py
============================================== test session starts ==============================================
platform win32 -- Python 3.11.2, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect
collected 6 items

test\test_carconnect.py ......                                                                          [100%]

============================================== 6 passed in 0.24s ==============================================
```

-> WINDOWS POWERSHELL

1. Test customer authentication with invalid credentials.

```
main_module.py: error: the following arguments are required: filenames   python -m main.main_module
>> C:\Users\Aishwarya\OneDrive\Desktop\hexaware\ASSIGNMENT 1\CarConnect>
=== Welcome to CarConnect ===
Are you a:
1. Customer
2. Admin
Enter choice (1 or 2): 1
=== Customer Login ===
Username: meerapatel
Password: kr@2024
   Invalid customer username or password.
```

2. Test updating customer information.

```
   Welcome, Karthik!


==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. Get Available Vehicles
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1-10): 2
Update Details:
First name: Karthik
Last name: Rao
Email: karthikrao007@gmail.com
Phone: 8903019088
Address: 11 MG Road, Bombay
Username: karthikrao
Password: kr@2024
   Customer details updated successfully.
```

BEFORE:

```
mysql> select * from customer;
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
| CustomerID | FirstName | LastName   | Email                   | PhoneNumber | Address                   | Username   | Password   | RegistrationDate |
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
|          1 | Ravi      | Sharma     | ravi.sharma@gmail.com   | 9876543210  | 11 MG Road, Pune          | ravi01     | ravi@123   | 2024-06-01       |
|          2 | Anjali    | Mehra      | anjali.mehra@yahoo.com  | 9823456789  | 22 LBS Marg, Mumbai       | anjalim    | anjali@123 | 2024-06-02       |
|          3 | Karthik   | Rao        | karthik.rao@gmail.com   | 9845612345  | 3 Gandhi Street, Hyderabad | karthikrao | kr@2024    | 2024-06-03       |
|          4 | Sneha     | Pillai     | sneha.pillai@gmail.com  | 9812345678  | 45 Brigade Rd, Bengaluru  | snehap     | sneha#pass | 2024-06-04       |
|          5 | Amit      | Verma      | amit.verma@gmail.com    | 9934567890  | 67 Nehru Nagar, Delhi     | amitv      | amit1234   | 2024-06-05       |
|          6 | Nisha     | Patil      | nisha.patil@gmail.com   | 9988776655  | 29 Boat Club Rd, Pune     | nishapatil | nish@987   | 2024-06-06       |
|          7 | Rahul     | Desai      | rahul.desai@yahoo.com   | 9867543210  | 88 Juhu Beach Rd, Mumbai  | rahuldesai | rahul#456  | 2024-06-07       |
|          8 | Pooja     | Yadav      | pooja.yadav@gmail.com   | 9898989898  | 15 Kalindi Kunj, Lucknow  | poojay     | poo@pass   | 2024-06-08       |
|          9 | Rakesh    | Nair       | rakesh.nair@gmail.com   | 9743210123  | 91 Kochi Bypass           | rakeshn    | rakesh12   | 2024-06-09       |
|         10 | Divya     | Reddy      | divya.reddy@gmail.com   | 9767891234  | 5 Jubilee Hills, Hyderabad | divyar     | divya@pass | 2024-06-10       |
|         11 | Arjun     | Bhatt      | arjun.bhatt@gmail.com   | 9834567891  | 10 Shastri Nagar, Jaipur  | arjunbhatt | ajb456     | 2024-06-11       |
|         12 | Meena     | Kapoor     | meena.kapoor@gmail.com  | 9807654321  | 73 Rajouri Garden, Delhi  | meenak     | meena@321  | 2024-06-12       |
|         13 | Santosh   | Joshi      | santosh.joshi@gmail.com | 9798765432  | 19 Camp Area, Nagpur      | santoshj   | sj@pass    | 2024-06-13       |
|         14 | Isha      | Chatterjee | isha.cj@gmail.com       | 9754321876  | 88 Bhowanipur, Kolkata    | ishacj     | ic@2024    | 2024-06-14       |
|         15 | Vikram    | Bansal     | vikram.bansal@gmail.com | 9789012345  | 11 Vikas Puri, Delhi      | vikramb    | vbansal    | 2024-06-15       |
|         17 | Meera     | Patel      | meera.patel@example.com | 9876543210  | Mumbai                    | meerapatel | meera@123  | 2025-06-20       |
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
16 rows in set (0.00 sec)
```

AFTER:

```
mysql> select * from customer;
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
| CustomerID | FirstName | LastName   | Email                   | PhoneNumber | Address                   | Username   | Password   | RegistrationDate |
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
|          1 | Ravi      | Sharma     | ravi.sharma@gmail.com   | 9876543210  | 11 MG Road, Pune          | ravi01     | ravi@123   | 2024-06-01       |
|          2 | Anjali    | Mehra      | anjali.mehra@yahoo.com  | 9823456789  | 22 LBS Marg, Mumbai       | anjalim    | anjali@123 | 2024-06-02       |
|          3 | Karthik   | Rao        | karthikrao007@gmail.com | 8903019088  | 11 MG Road, Bombay        | karthikrao | kr@2024    | 2025-06-21       |
|          4 | Sneha     | Pillai     | sneha.pillai@gmail.com  | 9812345678  | 45 Brigade Rd, Bengaluru  | snehap     | sneha#pass | 2024-06-04       |
|          5 | Amit      | Verma      | amit.verma@gmail.com    | 9934567890  | 67 Nehru Nagar, Delhi     | amitv      | amit1234   | 2024-06-05       |
|          6 | Nisha     | Patil      | nisha.patil@gmail.com   | 9988776655  | 29 Boat Club Rd, Pune     | nishapatil | nish@987   | 2024-06-06       |
|          7 | Rahul     | Desai      | rahul.desai@yahoo.com   | 9867543210  | 88 Juhu Beach Rd, Mumbai  | rahuldesai | rahul#456  | 2024-06-07       |
|          8 | Pooja     | Yadav      | pooja.yadav@gmail.com   | 9898989898  | 15 Kalindi Kunj, Lucknow  | poojay     | poo@pass   | 2024-06-08       |
|          9 | Rakesh    | Nair       | rakesh.nair@gmail.com   | 9743210123  | 91 Kochi Bypass           | rakeshn    | rakesh12   | 2024-06-09       |
|         10 | Divya     | Reddy      | divya.reddy@gmail.com   | 9767891234  | 5 Jubilee Hills, Hyderabad | divyar     | divya@pass | 2024-06-10       |
|         11 | Arjun     | Bhatt      | arjun.bhatt@gmail.com   | 9834567891  | 10 Shastri Nagar, Jaipur  | arjunbhatt | ajb456     | 2024-06-11       |
|         12 | Meena     | Kapoor     | meena.kapoor@gmail.com  | 9807654321  | 73 Rajouri Garden, Delhi  | meenak     | meena@321  | 2024-06-12       |
|         13 | Santosh   | Joshi      | santosh.joshi@gmail.com | 9798765432  | 19 Camp Area, Nagpur      | santoshj   | sj@pass    | 2024-06-13       |
|         14 | Isha      | Chatterjee | isha.cj@gmail.com       | 9754321876  | 88 Bhowanipur, Kolkata    | ishacj     | ic@2024    | 2024-06-14       |
|         15 | Vikram    | Bansal     | vikram.bansal@gmail.com | 9789012345  | 11 Vikas Puri, Delhi      | vikramb    | vbansal    | 2024-06-15       |
|         17 | Meera     | Patel      | meera.patel@example.com | 9876543210  | Mumbai                    | meerapatel | meera@123  | 2025-06-20       |
+------------+-----------+------------+-------------------------+-------------+---------------------------+------------+------------+------------------+
16 rows in set (0.01 sec)
```

3. Test adding a new vehicle.

```
2. Admin
Enter choice (1 or 2): 2
=== Admin Login ===
Admin Username: adminuser0002
Password: admin@456
   Welcome Admin, Alok!


=== ADMIN MENU ===
1. Add Vehicle
2. View All Available Vehicles
3. View Reservation by ID
4. View Reservations by Customer ID
5. Delete Admin Account
6. View Reports
7. Logout
Choose an option (1-7): 1
Model: Swift Dzire
Make: Maruti Suzuki
Year: 2017
Color: Blue
Reg No: TN05DY0061
Rate per day ₹: 5000.00
   Vehicle added.
```

```
mysql> select* from vehicle;
+-----------+--------------+---------------+------+--------+--------------------+---------------------------+-----------+
| VehicleID | Model        | Make          | Year | Color  | RegistrationNumber | Availability              | DailyRate |
+-----------+--------------+---------------+------+--------+--------------------+---------------------------+-----------+
|         1 | Innova Crysta | Toyota        | 2022 | White  | MH01AB1234         | 0x01                      |  12500.00 |
|         2 | XUV700       | Mahindra      | 2023 | Black  | DL05CD2345         | 0x01                      |  13999.00 |
|         3 | City         | Honda         | 2021 | Silver | KA03EF3456         | 0x01                      |  10500.00 |
|         4 | Verna        | Hyundai       | 2022 | Red    | TN07GH4567         | 0x01                      |  11000.00 |
|         5 | Kushaq       | Skoda         | 2023 | Blue   | GJ09IJ5678         | 0x01                      |   9800.00 |
|         6 | Creta        | Hyundai       | 2021 | Grey   | MH12KL6789         | 0x00                      |  11500.00 |
|         7 | Fortuner     | Toyota        | 2023 | White  | DL10MN7890         | 0x01                      |  15500.00 |
|         8 | Thar         | Mahindra      | 2022 | Black  | KA05OP8901         | 0x01                      |  13000.00 |
|         9 | Compass      | Jeep          | 2021 | Red    | KL11QR9012         | 0x01                      |  14200.00 |
|        10 | Seltos       | Kia           | 2022 | Blue   | TN22ST0123         | 0x01                      |  10800.00 |
|        11 | i20          | Hyundai       | 2020 | Silver | UP32UV1234         | 0x01                      |   9500.00 |
|        12 | Altroz       | Tata          | 2021 | Gold   | WB19WX2345         | 0x01                      |   9200.00 |
|        13 | Brezza       | Maruti        | 2022 | White  | RJ14YZ3456         | 0x01                      |   9900.00 |
|        14 | Scorpio-N    | Mahindra      | 2023 | Black  | AP10ZA4567         | 0x01                      |  14900.00 |
|        15 | Venue        | Hyundai       | 2022 | Grey   | CH01XY5678         | 0x01                      |  10200.00 |
|        16 | Swift        | Maruti        | 2022 | Red    | TN10AA1234         | 0x01                      |   3200.00 |
|        17 | Baleno       | Maruti        | 2022 | Blue   | TN12BA2345         | 0x01                      |   9200.00 |
|        18 | Hector       | MG            | 2023 | Black  | MH14HG5678         | 0x01                      |  14500.00 |
|        19 | Magnite      | Nissan        | 2022 | Red    | KA09MG8901         | 0x00                      |   8600.00 |
|        20 | Tiago        | Tata          | 2021 | White  | DL03TG3456         | 0x01                      |   8100.00 |
|        27 | Honda        | City          | 2022 | Red    | TN09CX9999         | 0x01                      |   1800.00 |
|        28 | Swift Dzire  | Maruti Suzuki | 2017 | Blue   | TN05DY0061         | 0x01                      |   5000.00 |
+-----------+--------------+---------------+------+--------+--------------------+---------------------------+-----------+
22 rows in set (0.01 sec)
```

4. Test updating customer details.

```
==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. Get Available Vehicles
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1-10): 2

Update Details:
First name: Mohammed
Last name: Sheriff
Email: Sheriff@gmail.com
Phone: 1234567890
Address: 13 west
Username: root
Password: root123

   Customer details updated successfully.
```

5. Test getting a list of available vehicles.

```
==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. Get Available Vehicles
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1–10): 6
Available Vehicles:
1 | Innova Crysta | White | ₹12500.00
2 | XUV700 | Black | ₹13999.00
3 | City | Silver | ₹10500.00
4 | Verna | Red | ₹11000.00
5 | Kushaq | Blue | ₹9800.00
7 | Fortuner | White | ₹15500.00
8 | Thar | Black | ₹13000.00
9 | Compass | Red | ₹14200.00
10 | Seltos | Blue | ₹10800.00
11 | i20 | Silver | ₹9500.00
12 | Altroz | Gold | ₹9200.00
13 | Brezza | White | ₹9900.00
14 | Scorpio-N | Black | ₹14900.00
15 | Venue | Grey | ₹10200.00
16 | Swift | Red | ₹3200.00
17 | Baleno | Blue | ₹9200.00
18 | Hector | Black | ₹14500.00
20 | Tiago | White | ₹8100.00
27 | Honda | Red | ₹1800.00
28 | Swift Dzire | Blue | ₹5000.00
```

6. Test getting a list of all vehicles.

```
==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. Get Available Vehicles
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1–10): 6
2. Innova Crysta Toyota | ₹12500.00 | White | Reg: MH01AB1234
3. XUV700 Mahindra | ₹13500.00 | Black | Reg: DL08CD2333
4. City Honda | ₹9500.00 | Silver | Reg: KA09FG2345
5. Verna Hyundai | ₹10800.00 | Red | Reg: TN48HT5678
6. Kushaq Skoda | ₹11000.00 | Blue | Reg: GJ37MG4567
7. Creta Hyundai | ₹10500.00 | Grey | Reg: TN01HL6789
8. Fortuner Toyota | ₹18000.00 | White | Reg: DL11NP0987
9. Thar Mahindra | ₹13000.00 | Black | Reg: KA04PB9987
10. Compass Jeep | ₹15500.00 | Black | Reg: KL19DD7681
11. Seltos Kia | ₹10900.00 | Silver | Reg: TN25RT3210
12. i20 Hyundai | ₹9800.00 | Blue | Reg: UP32VV1235
13. Altroz Tata | ₹9400.00 | Gold | Reg: MH12WX3415
14. Brezza Maruti | ₹9900.00 | White | Reg: GJ01ZX2456
15. Scorpio-N Mahindra | ₹12500.00 | Black | Reg: KA53DT9876
16. Venue Hyundai | ₹10200.00 | Grey | Reg: TN41CV1234
17. Swift Maruti | ₹9200.00 | Red | Reg: CH01AA2345
18. Baleno Maruti | ₹10400.00 | White | Reg: KL07TR2456
19. Hector MG | ₹12000.00 | Blue | Reg: TN05HH1992
20. Magnite Nissan | ₹8600.00 | Red | Reg: KA49MG6543
21. Amaze Honda | ₹9600.00 | Red | Reg: TN38CV9999
22. City Honda | ₹9800.00 | Red | Reg: TN09SC3456
23. Swift Dzire Maruti Suzuki | ₹5000.00 | Blue | Reg: TN85DY0061
```

BUSINESS LOGIC ENHANCEMENTS:

To elevate CarConnect beyond a basic rental system, several real-world business rules and enhancements have been implemented. These features not only improve customer experience but also reflect thoughtful, safety-conscious, and customer-centric design, such as making the platform feel practical, intelligent, and ready for real-world adoption.

1. Birthday Loyalty Discount:

- To encourage customer retention and celebrate loyalty, CarConnect offers a **15% discount** on bookings made during a customer's **birthday month**.
- This is automatically applied at the time of reservation and helps foster a long-term relationship between the user and the platform.

TERMINAL OUTPUT:

2. Age-Based Safety Restriction:

To align with legal and safety norms, users **under the age of 15** are **not allowed to make a reservation**. This safeguard prevents misuse and promotes responsible usage, ensuring that the system complies with real-world driving and rental age guidelines.

Any attempt by underage users to book a vehicle results in an appropriate validation message.

```
==== CUSTOMER MENU ====
1. View My Details
2. Update My Details
3. Delete My Account
4. Add Vehicle
5. Get Vehicle by ID
6. View Available Vehicles by Date Range
7. Book a Vehicle
8. View My Reservations
9. Cancel a Reservation
10. Logout
Choose an option (1-10): 7

   Top Featured Vehicle
Vehicle ID : 1
Model      : Civic
Booked     : 4 times

Start Date (YYYY-MM-DD): 2025-07-08
End Date (YYYY-MM-DD): 2025-07-11

   Available Vehicles:
1 | Civic | Red | ₹2000.00
2 | Corolla | Blue | ₹1800.00
3 | Jeep | Matte Black | ₹2100.00

Enter Vehicle ID to book: 2
   You must be at least 15 years old to book a vehicle.
```

3. Featured Vehicle Highlight:

CarConnect automatically highlights the **top-performing vehicle** — the one with the **highest number of bookings**. This feature helps new users identify popular and trusted vehicle options quickly, improving decision-making and reducing hesitation. It also serves as a promotional tool, helping admins recognise and feature high-demand vehicles.

```
   Top Featured Vehicle
Vehicle ID : 1
Model      : Civic
Booked     : 4 times

Start Date (YYYY-MM-DD): 2025-07-07
End Date (YYYY-MM-DD): 2025-07-09

   Available Vehicles:
1 | Civic | Red | ₹2000.00
2 | Corolla | Blue | ₹1800.00
3 | Jeep | Matte Black | ₹2100.00
```

CONCLUSION:

This project was a result of strong collaboration, consistent communication, and shared learning. Each member contributed to planning, coding, testing, and refining features and ensuring a well-rounded, functional system. Our teamwork not only improved the project quality but also enhanced our understanding of real-world software development practices.