# CARCONNECT

# A CAR RENTAL PLATFORM

ANUSH KUMAR

## INTRODUCTION

Using Python and SQL Server, CarConnect is a cutting-edge database-driven vehicle rental management system that prioritizes modularity, maintainability, and practicality. By automating crucial tasks including client registration, vehicle management, reservation processing, and administrative controls, the system seeks to optimize a car rental company's operations.

Features including car availability tracking, customer and admin authentication, reservation creation and modifications, and the compilation of reports on vehicle usage, reservation history, and total income are all supported by the system. Through appropriate use of foreign key restrictions and exception handling, CarConnect guarantees dependable data permanence and integrity by utilizing MySQL Server as the backend database.

CarConnect was created with scalability in mind and may be modified for small-to-medium sized vehicle rental companies who want to automate their processes. It is also a great contender for future revisions that integrate with web frameworks or APIs due to its flexible architecture.

CarConnect is more than simply a project; it's a workable answer to the demands of contemporary rental businesses. All things considered, it offers a strong basis for learning enterprise-level application development with an emphasis on real-time problem-solving, data validation, database interface, and user experience.

**PURPOSE OF THE PROJECT**

The goal of the CarConnect project is to provide a complete management system for rental automobile rentals that streamlines and automates the essential functions of a car rental company. With the help of this system, administrators and clients will be able to effectively handle cars, reservations, and user data by replacing manual, old methods with a centralized digital solution. The purpose of CarConnect is to: Make it easy for clients to register, authenticate, and reserve available cars.

- Permit administrators to keep an eye on client reservations, adjust availability, and manage car inventory.
- By using appropriate database interaction, validation, and authentication, you may guarantee safe data processing.
- Using reporting tools and status updates, give insight into operations.
- Create analytical reports to aid in corporate decision-making, such as revenue summaries, vehicle usage, and reservation history.

**SCOPE OF THE PROJECT:**

All of the key functional areas needed to run a vehicle rental business effectively are covered by the CarConnect system. Its scope includes both customer-facing and administrative features, combining a relational database and organized backend for dependable data management.
The project's primary focus areas are as follows:
Administrative Management:

Admins have the ability to create, edit, and remove profiles. They may check reservation records, manage car data, and create reports to track company success.

Customer management: Clients are able to register, log in, see their personal information, and reserve cars. They have the ability to examine and manage booking history.

Vehicle Management: Admins have the ability to examine, edit, add, or delete vehicle information, such as availability and daily rental costs. This guarantees that the inventory will always be correct and up to date.

Reservation System: By choosing dates and figuring out the total cost, the system enables users to book available cars. Reservation statuses can be updated, canceled, or confirmed by administrators.

Reporting Module: For business insights and performance monitoring, administrators may create a variety of reports, such as those on vehicle usage, reservation history, and income.

Database Integration: To guarantee data permanence, relational integrity, and seamless CRUD operations, every action is linked to a strong MySQL database.

**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.

**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.

**3. Reservation Table:**

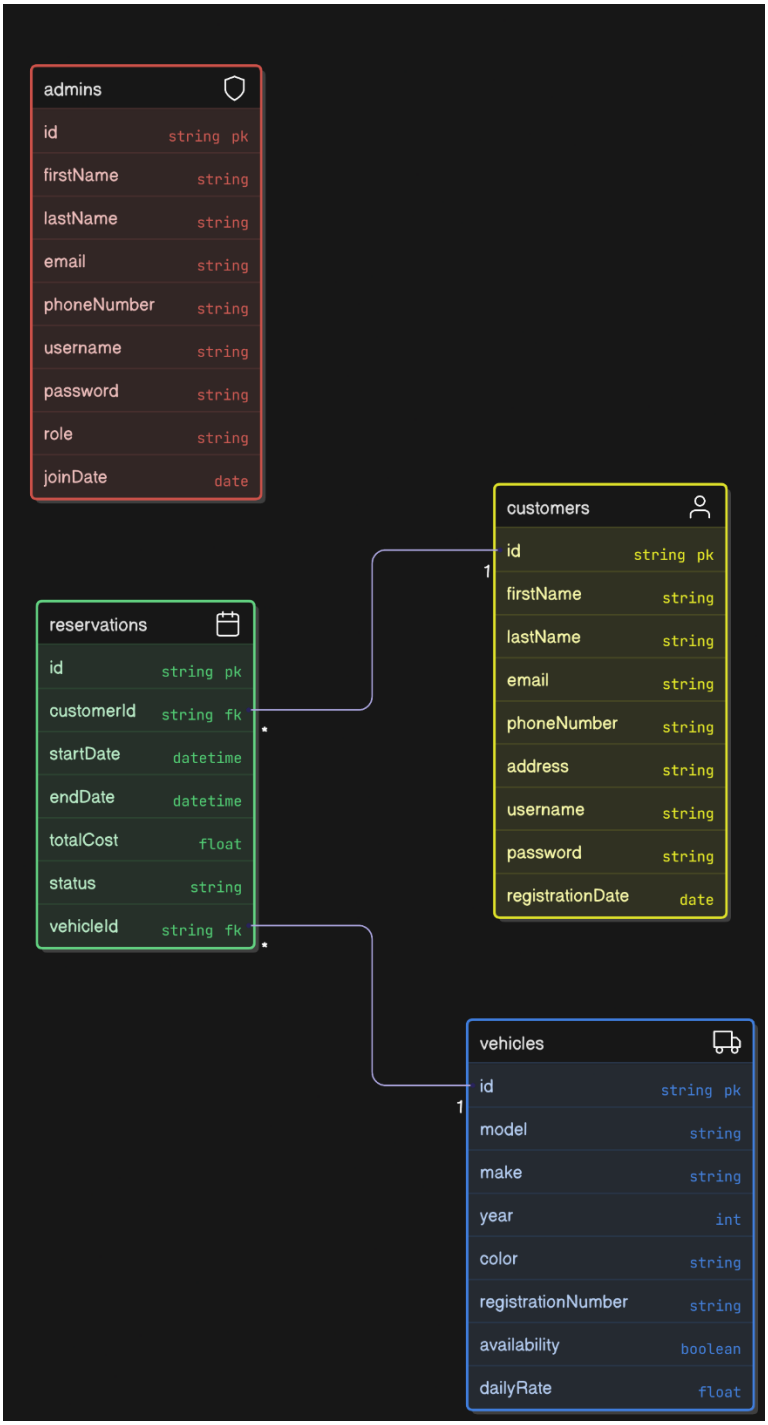
- **ReservationID (Primary Key):** Unique identifier for each reservation.
- **CustomerID (Foreign Key):** Foreign key referencing the Customer table.
- **VehicleID (Foreign Key):** Foreign key referencing the Vehicle table.
- **StartDate:** Date and time of the reservation start.

- **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).

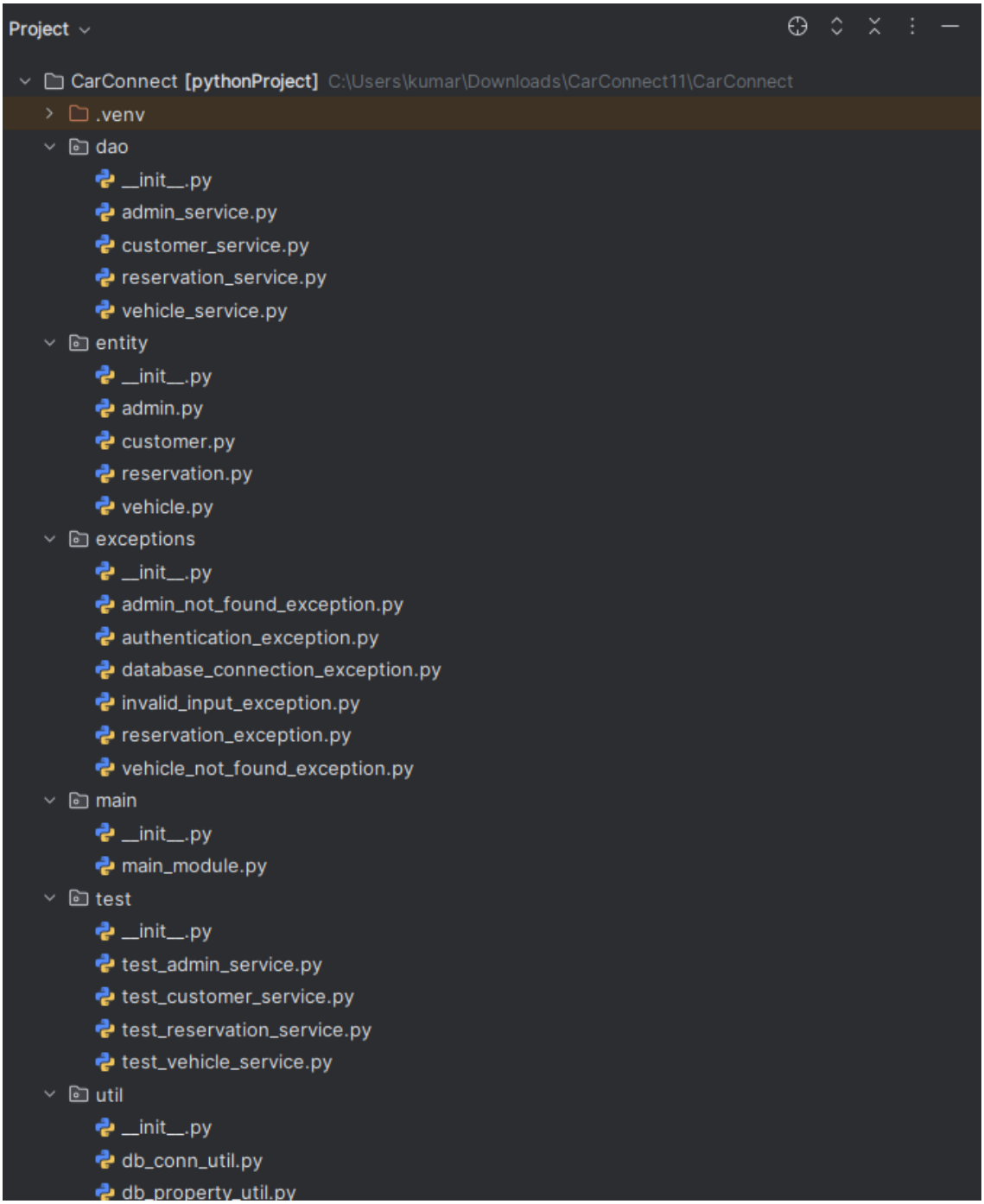
**4. Admin Table:**

- **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.

## ER DIAGARM


admins
id string pk
firstName string
lastName string
email string
phoneNumber string
username string
password string
role string
joinDate date
customers
id string pk
firstName string
lastName string
email string
phoneNumber string
address string
username string
password string
registrationDate date
reservations
id string pk
customerId string fk
startDate datetime
endDate datetime
totalCost float
status string
vehicleId string fk
vehicles
id string pk
model string
make string
year int
color string
registrationNumber string
availability boolean
dailyRate float
1
*
1
*

**PYTHON DRECTORY:**



**SQL QUERIES:**

**SQL DATABASE:**

1. **Creating Database:**

```
create database CarConnect;
use Carconnect;
```

**2. Creating Tables:**

**Customer Table:**

```
CREATE TABLE Customer (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    Address TEXT NOT NULL,
    Username VARCHAR(50) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    RegistrationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

**Vehicle Table:**

```
CREATE TABLE Vehicle (
    VehicleID INT AUTO_INCREMENT PRIMARY KEY,
    Model VARCHAR(50) NOT NULL,
    Make VARCHAR(50) NOT NULL,
    Year INT NOT NULL,
    Color VARCHAR(30) NOT NULL,
    RegistrationNumber VARCHAR(50) UNIQUE NOT NULL,
    Availability BOOLEAN DEFAULT TRUE,
    DailyRate DECIMAL(10,2) NOT NULL
);
```

**Reservation Table:**

```
CREATE TABLE Reservation (
    ReservationID INT AUTO_INCREMENT PRIMARY KEY,
```

```
    CustomerID INT,
    VehicleID INT,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    TotalCost DECIMAL(10,2) NOT NULL,
    Status ENUM('pending', 'confirmed', 'completed', 'cancelled') NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
CASCADE,
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID) ON DELETE CASCADE
);
```

**Admin Table:**

```
CREATE TABLE Admin (
    AdminID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    Username VARCHAR(50) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,  -- Store hashed passwords
    Role ENUM('super admin', 'fleet manager') NOT NULL,
    JoinDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 3. Inserting Sample values

**Customer Table:**

```
INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username,
Password)
VALUES
('Arjun', 'Rao', 'arjun.rao@example.com', '9876543210', '123 MG Road, Bangalore', 'arjunrao',
'pass123'),
('Priya', 'Sharma', 'priya.sharma@example.com', '9123456780', '456 Anna Salai, Chennai',
'priyasharma', 'pass123'),
('Vikram', 'Patel', 'vikram.patel@example.com', '9988776655', '789 FC Road, Pune', 'vikramp',
'pass123'),
('Sneha', 'Kumar', 'sneha.kumar@example.com', '9090909090', '11 Park Street, Kolkata',
'snehak', 'pass123'),
('Ravi', 'Verma', 'ravi.verma@example.com', '8012345678', '88 Marine Drive, Mumbai', 'raviv',
```

'pass123'),
('Divya', 'Singh', 'divya.singh@example.com', '9876501234', '19 Ashok Nagar, Delhi', 'divyasingh', 'pass123'),
('Karan', 'Mehta', 'karan.mehta@example.com', '9234567890', '40 JP Nagar, Bangalore', 'karanm', 'pass123'),
('Meena', 'Iyer', 'meena.iyer@example.com', '9345678901', '17 Purasawalkam, Chennai', 'meenai', 'pass123'),
('Ajay', 'Das', 'ajay.das@example.com', '9123456700', '9 EM Bypass, Kolkata', 'ajayd', 'pass123'),
('Lakshmi', 'Nair', 'lakshmi.nair@example.com', '9988007766', '55 Vyttila, Kochi', 'lakshmin', 'pass123');

**Vehicle Table:**

INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
VALUES
('Swift', 'Maruti', 2021, 'Red', 'KA01AB1234', TRUE, 1200.00),
('City', 'Honda', 2020, 'Black', 'TN02BC5678', TRUE, 1500.00),
('Innova', 'Toyota', 2019, 'Silver', 'MH03CD9101', TRUE, 2000.00),
('i20', 'Hyundai', 2022, 'White', 'DL04EF1122', TRUE, 1300.00),
('Creta', 'Hyundai', 2021, 'Grey', 'KL05GH3344', TRUE, 1800.00),
('Ertiga', 'Maruti', 2020, 'Blue', 'KA06IJ5566', TRUE, 1700.00),
('Fortuner', 'Toyota', 2023, 'Black', 'TN07KL7788', TRUE, 2500.00),
('Baleno', 'Maruti', 2021, 'Red', 'MH08MN9900', TRUE, 1400.00),
('Venue', 'Hyundai', 2022, 'White', 'DL09OP1112', TRUE, 1600.00),
('Altroz', 'Tata', 2020, 'Yellow', 'KL10QR1314', TRUE, 1100.00);

**Reservation Table:**

INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
VALUES
(1, 2, '2025-03-01 10:00:00', '2025-03-05 10:00:00', 6000.00, 'completed'),
(2, 4, '2025-03-03 09:00:00', '2025-03-04 09:00:00', 1300.00, 'completed'),
(3, 1, '2025-03-07 12:00:00', '2025-03-10 12:00:00', 3600.00, 'confirmed'),
(4, 6, '2025-03-11 08:00:00', '2025-03-14 08:00:00', 5100.00, 'cancelled'),
(5, 3, '2025-03-05 18:00:00', '2025-03-06 18:00:00', 2000.00, 'completed'),
(6, 7, '2025-03-08 10:00:00', '2025-03-09 10:00:00', 2500.00, 'confirmed'),
(7, 5, '2025-03-12 11:00:00', '2025-03-13 11:00:00', 1800.00, 'pending'),
(8, 9, '2025-03-14 13:00:00', '2025-03-16 13:00:00', 3200.00, 'pending'),

(9, 8, '2025-03-01 10:00:00', '2025-03-02 10:00:00', 1400.00, 'completed'),
(10, 10, '2025-03-02 09:00:00', '2025-03-04 09:00:00', 2200.00, 'confirmed');

**Admin Table:**

INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role)
VALUES
('Ramesh', 'Iyer', 'ramesh.iyer@carconnect.com', '9999988888', 'rameshadmin', 'admin123', 'super admin'),
('Geeta', 'Menon', 'geeta.menon@carconnect.com', '9888777666', 'geetamenon', 'admin123', 'fleet manager'),
('Suraj', 'Singh', 'suraj.singh@carconnect.com', '9777666555', 'surajsingh', 'admin123', 'fleet manager'),
('Kavita', 'Das', 'kavita.das@carconnect.com', '9666555444', 'kavitadas', 'admin123', 'super admin'),
('Anil', 'Jain', 'anil.jain@carconnect.com', '9555444333', 'aniljain', 'admin123', 'fleet manager'),
('Pooja', 'Rao', 'pooja.rao@carconnect.com', '9444333222', 'poojarao', 'admin123', 'super admin'),
('Naveen', 'Kumar', 'naveen.kumar@carconnect.com', '9333222111', 'naveenk', 'admin123', 'fleet manager'),
('Meera', 'Nair', 'meera.nair@carconnect.com', '9222111000', 'meeranair', 'admin123', 'fleet manager'),
('Rahul', 'Verma', 'rahul.verma@carconnect.com', '9111000099', 'rahulverma', 'admin123', 'super admin'),
('Divya', 'Joshi', 'divya.joshi@carconnect.com', '9000099999', 'divyajoshi', 'admin123', 'fleet manager');

**PYTHON PROGRAM:**

**ENTITY**

**Admin.py:**

```
class Admin:
    def __init__(self, admin_id, first_name, last_name, email, phone, username, password, role,
join_date):
        self.admin_id = admin_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.username = username
        self.password = password
        self.role = role
        self.join_date = join_date

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

**customer.py:**

```
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone, address, username,
password, registration_date):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.address = address
        self.username = username
        self.password = password
        self.registration_date = registration_date

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

**reservation.py:**

```
class Reservation:
    def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost,
status):
        self.reservation_id = reservation_id
        self.customer_id = customer_id
        self.vehicle_id = vehicle_id
        self.start_date = start_date
        self.end_date = end_date
        self.total_cost = total_cost
        self.status = status

    def calculate_total_cost(self, daily_rate, days):
        self.total_cost = daily_rate * days
```

**vehicle.py:**

```
class Vehicle:
    def __init__(self, vehicle_id, model, make, year, color, registration_number, availability,
daily_rate):
        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
```

**DAO:**

**Admin_service.py:**

```
from CarConnect.entity.admin import Admin
from CarConnect.exceptions.admin_not_found_exception import AdminNotFoundException
from CarConnect.exceptions.invalid_input_exception import InvalidInputException
from CarConnect.exceptions.database_connection_exception import
DatabaseConnectionException

class AdminService(Admin):
    def __init__(self, db):
        self.db = db

    def get_admin_by_id(self, admin_id):
        if not isinstance(admin_id, int):
            raise InvalidInputException("Admin ID must be an integer.")
        try:
            query = "SELECT * FROM Admin WHERE AdminID = %s"
            row = self.db.fetch_query(query, (admin_id,))
            if not row:
                raise AdminNotFoundException(f"Admin with ID {admin_id} not found.")
            print("The Admin is:",row)
        except Exception as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")

    def get_admin_by_username(self, username):
        if not isinstance(username, str) or not username.strip():
            raise InvalidInputException("Username must be a non-empty string.")
        try:
            query = "SELECT * FROM Admin WHERE Username = %s"
            row = self.db.fetch_query(query, (username,))
            if not row:
                raise AdminNotFoundException(f"No admin found with username: {username}")
            print("The user is:",row)
        except Exception as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")

    def register_admin(self, admin):
        if not isinstance(admin, Admin):
```

```
            raise InvalidInputException("Invalid admin object.")
        try:
            query = """
                INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username,
Password, Role, JoinDate)
                VALUES (%s, %s, %s, %s, %s, %s, %s, NOW())
            """
            values = (
                admin.first_name, admin.last_name, admin.email,
                admin.phone, admin.username, admin.password, admin.role
            )
            self.db.execute_query(query, values)
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to register admin: {str(e)}")

    def update_admin(self, admin_id, first_name, last_name,email,phone,username,role):
        if not isinstance(admin_id, int):
            raise InvalidInputException("Admin ID must be an integer.")
        try:
            query = ("UPDATE Admin SET FirstName = %s, LastName = %s,"
                     "Email = %s, PhoneNumber = %s, username = %s, role = %s WHERE AdminID
= %s")
            result = self.db.execute_query(query, (first_name, last_name,email, phone,username,role,
admin_id))
            if result == 0:
                raise AdminNotFoundException(f"Admin with ID {admin_id} not found.")
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to update admin: {str(e)}")

    def delete_admin(self, admin_id):
        if not isinstance(admin_id, int):
            raise InvalidInputException("Admin ID must be an integer.")
        try:
            query = "DELETE FROM Admin WHERE AdminID = %s"
            result = self.db.execute_query(query, (admin_id,))
            if result == 0:
                raise AdminNotFoundException(f"Admin with ID {admin_id} not found.")
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to delete admin: {str(e)}")
```

**customer_service.py:**

```
from CarConnect.entity.customer import Customer
from CarConnect.exceptions.invalid_input_exception import InvalidInputException
from CarConnect.exceptions.authentication_exception import AuthenticationException

class CustomerService(Customer):
    def __init__(self, db):
        self.db = db

    def get_customer_by_id(self, customer_id):
        if not isinstance(customer_id, int):
            raise InvalidInputException("Customer ID must be an integer.")
        query = "SELECT * FROM Customer WHERE CustomerID = %s"
        result = self.db.fetch_query(query, (customer_id,))
        if not result:
            raise InvalidInputException(f"Customer with ID {customer_id} not found.")
        print("The Customer: ",result)

    def get_customer_by_username(self, username):
        if not isinstance(username, str) or not username.strip():
            raise InvalidInputException("Username must be a non-empty string.")
        query = "SELECT * FROM Customer WHERE Username = %s"
        result = self.db.fetch_query(query, (username,))
        if not result:
            raise InvalidInputException(f"Customer with username '{username}' not found.")
        print("The Customer by ID: ",result)

    def register_customer(self, customer):
        if not isinstance(customer, Customer):
            raise InvalidInputException("Invalid customer object.")
        query = """
            INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address,
Username, Password, RegistrationDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, NOW())
        """
        self.db.execute_query(query, (
            customer.first_name, customer.last_name, customer.email,
            customer.phone, customer.address, customer.username, customer.password
        ))
```

```
    def update_customer(self, customer_id,first_name,last_name ,email, phone,
address,username):
        if not isinstance(customer_id, int):
            raise InvalidInputException("Customer ID must be an integer.")
        query = """
            UPDATE Customer SET firstname = %s,lastname = %s,
            Email = %s, PhoneNumber = %s, Address = %s,username = %s WHERE CustomerID
= %s
        """
        self.db.execute_query(query, (first_name,last_name,email, phone,
address,username,customer_id))

    def delete_customer(self, customer_id):
        if not isinstance(customer_id, int):
            raise InvalidInputException("Customer ID must be an integer.")
        query = "DELETE FROM Customer WHERE CustomerID = %s"
        self.db.execute_query(query, (customer_id,))

    def authenticate_customer(self, username, password):
        if not isinstance(username, str) or not username.strip():
            raise InvalidInputException("Username must be a non-empty string.")
        if not isinstance(password, str) or not password.strip():
            raise InvalidInputException("Password must be a non-empty string.")

        query = "SELECT * FROM Customer WHERE Username = %s AND Password = %s"
        result = self.db.fetch_query(query, (username, password))
        if not result:
            raise AuthenticationException("Invalid username or password.")
        print("The User is:",result)
```

**reservation_service.py:**

```
from CarConnect.entity.reservation import Reservation
from CarConnect.exceptions.invalid_input_exception import InvalidInputException
from CarConnect.exceptions.reservation_exception import ReservationException

class ReservationService(Reservation):
    def __init__(self, db):
```

```
        self.db = db

    def get_reservation_by_id(self, reservation_id):
        if not isinstance(reservation_id, int):
            raise InvalidInputException("Reservation ID must be an integer.")

        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
        result = self.db.fetch_query(query, (reservation_id,))

        if not result:
            raise ReservationException(f"No reservation found with ID: {reservation_id}")

        return Reservation(*result[0])

    def get_reservations_by_customer_id(self, customer_id):
        if not isinstance(customer_id, int):
            raise InvalidInputException("Customer ID must be an integer.")

        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
        result = self.db.fetch_query(query, (customer_id,))

        if not result:
            raise ReservationException(f"No reservations found for customer ID: {customer_id}")

        for row in result:
            reservation = Reservation(*row)
            print(f"Reservation ID: {reservation.reservation_id}")
            print(f"Vehicle ID   : {reservation.vehicle_id}")
            print(f"Start Date   : {reservation.start_date}")
            print(f"End Date     : {reservation.end_date}")
            print(f"Total Cost   : {reservation.total_cost}")
            print(f"Status       : {reservation.status}")

    def create_reservation(self, reservation):
        if not isinstance(reservation, Reservation):
            raise InvalidInputException("Invalid reservation object.")

        query = """
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost,
Status)
```

```
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    self.db.execute_query(query, (
        reservation.customer_id, reservation.vehicle_id, reservation.start_date,
        reservation.end_date, reservation.total_cost, reservation.status
    ))

def update_reservation(self, reservation_id, status):
    if not isinstance(reservation_id, int):
        raise InvalidInputException("Reservation ID must be an integer.")

    query = "UPDATE Reservation SET Status = %s WHERE ReservationID = %s"
    rowcount = self.db.execute_query(query, (status, reservation_id))

    if rowcount == 0:
        raise ReservationException(f"No reservation found with ID: {reservation_id}")

def cancel_reservation(self, reservation_id):
    if not isinstance(reservation_id, int):
        raise InvalidInputException("Reservation ID must be an integer.")

    query = "DELETE FROM Reservation WHERE ReservationID = %s"
    rowcount = self.db.execute_query(query, (reservation_id,))

    if rowcount == 0:
        raise ReservationException(f"No reservation found with ID: {reservation_id}")

def generate_reservation_history_report(self):
    query = """
        SELECT ReservationID, CustomerID, VehicleID, StartDate, EndDate, Status
        FROM Reservation
        ORDER BY StartDate DESC
    """
    results = self.db.fetch_query(query)
    print("\n--- Reservation History Report ---")
    for row in results:
        print(row)

def generate_vehicle_utilization_report(self):
    query = """
```

```
            SELECT VehicleID, COUNT(*) AS TotalReservations
            FROM Reservation
            GROUP BY VehicleID
            ORDER BY TotalReservations DESC
        """
        results = self.db.fetch_query(query)
        print("\n--- Vehicle Utilization Report ---")
        for row in results:
            print(f"Vehicle ID: {row[0]}, Reservations: {row[1]}")

    def generate_revenue_report(self):
        query = """
            SELECT VehicleID, SUM(TotalCost) AS Revenue
            FROM Reservation
            WHERE Status = 'Completed'
            GROUP BY VehicleID
            ORDER BY Revenue DESC
        """
        results = self.db.fetch_query(query)
        print("\n--- Revenue Report ---")
        for row in results:
            print(f"Vehicle ID: {row[0]}, Revenue: ₹{row[1]:.2f}")
```

**vehicle_service.py:**

```
from CarConnect.entity.vehicle import Vehicle
from CarConnect.exceptions.vehicle_not_found_exception import VehicleNotFoundException
from CarConnect.exceptions.invalid_input_exception import InvalidInputException
from CarConnect.exceptions.database_connection_exception import
DatabaseConnectionException

class VehicleService(Vehicle):
    def __init__(self, db):
        self.db = db

    def get_vehicle_by_id(self, vehicle_id):
        if not isinstance(vehicle_id, int):
            raise InvalidInputException("Vehicle ID must be an integer.")
        try:
            query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
```

```
            row = self.db.fetch_query(query, (vehicle_id,))
            if not row:
                raise VehicleNotFoundException(f"No vehicle found with ID: {vehicle_id}")
            print("The vehicle is: ",row)
        except Exception as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")

    def get_available_vehicles(self):
        try:
            query = "SELECT * FROM Vehicle WHERE Availability = 1"
            rows = self.db.fetch_query(query)
            if not rows:
                raise VehicleNotFoundException("No available vehicles found.")
            return rows
        except Exception as e:
            raise DatabaseConnectionException(f"Database error: {str(e)}")

    def add_vehicle(self, vehicle):
        if not isinstance(vehicle, Vehicle):
            raise InvalidInputException("Invalid vehicle object.")
        try:
            query = """
                INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability,
DailyRate)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """
            values = (
                vehicle.model, vehicle.make, vehicle.year, vehicle.color,
                vehicle.registration_number, vehicle.availability, vehicle.daily_rate
            )
            self.db.execute_query(query, values)
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to add vehicle: {str(e)}")

    def update_vehicle(self, vehicle_id, daily_rate, availability):
        if not isinstance(vehicle_id, int):
            raise InvalidInputException("Vehicle ID must be an integer.")
        try:
            query = """
                UPDATE Vehicle SET DailyRate = %s, Availability = %s WHERE VehicleID = %s
```

```
            """
            result = self.db.execute_query(query, (daily_rate, availability, vehicle_id))
            if result == 0:
                raise VehicleNotFoundException(f"No vehicle found with ID: {vehicle_id}")
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to update vehicle: {str(e)}")

    def remove_vehicle(self, vehicle_id):
        if not isinstance(vehicle_id, int):
            raise InvalidInputException("Vehicle ID must be an integer.")
        try:
            query = "DELETE FROM Vehicle WHERE VehicleID = %s"
            result = self.db.execute_query(query, (vehicle_id,))
            if result == 0:
                raise VehicleNotFoundException(f"No vehicle found with ID: {vehicle_id}")
        except Exception as e:
            raise DatabaseConnectionException(f"Failed to delete vehicle: {str(e)}")
```

**EXCEPTIONS:**

**Admin_not_found_exception.py:**

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

**authentication_exceoption.py:**

```
class AuthenticationException(Exception):
    def __init__(self, message="Invalid username or password."):
        super().__init__(message)
```

**database_connection_exception.py:**

```
class DatabaseConnectionException(Exception):
    def __init__(self, message="Unable to connect to the database."):
        super().__init__(message)
```

**invalid_input_exception.py:**

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

**reservation_exception.py:**

```
class ReservationException(Exception):
    def __init__(self, message="Error in processing the reservation."):
        super().__init__(message)
```

**vehicle_not_found_exception.py:**

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

**UTIL:**

**db_conn_util.py:**

```
import mysql.connector

class DBConnUtil:
    def __init__(self, host="localhost", user="root", password="root", database="CarConnect"):
        self.conn = mysql.connector.connect(host=host, user=user, password=password,
database=database)
        self.cursor = self.conn.cursor()

    def execute_query(self, query, values=None):
        try:
            self.cursor.execute(query, values) if values else self.cursor.execute(query)
            self.conn.commit()
            print("Successful!!!")
        except mysql.connector.Error as e:
            print(f'Error executing query: {e}")

    def fetch_query(self, query, values=None):
        try:
            self.cursor.execute(query, values) if values else self.cursor.execute(query)
            result = self.cursor.fetchall()
```

```
            if result:
                print("Data retrieved successfully!")
            else:
                print("No records found.")
            return result
        except mysql.connector.Error as e:
            print(f"Error fetching data: {e}")
            return []

    def close_connection(self):
        self.cursor.close()
        self.conn.close()
```

**MAIN**

**main.py:**

```
from CarConnect.dao.admin_service import AdminService
from CarConnect.dao.customer_service import CustomerService
from CarConnect.dao.vehicle_service import VehicleService
from CarConnect.dao.reservation_service import ReservationService
from CarConnect.entity.admin import Admin
from CarConnect.entity.customer import Customer
from CarConnect.entity.vehicle import Vehicle
from CarConnect.entity.reservation import Reservation
from CarConnect.util.db_conn_util import DBConnUtil

db = DBConnUtil()
admin_service = AdminService(db)
customer_service = CustomerService(db)
vehicle_service = VehicleService(db)
reservation_service = ReservationService(db)

def admin_menu():
    print("\n--- Admin Menu ---")
    print("1. Register Admin")
    print("2. Get Admin by ID")
    print("3. Get Admin by Username")
    print("4. Update Admin")
    print("5. Delete Admin")

    choice = input("Enter choice: ")
```

```
if choice == '1':
    first = input("First name: ")
    last = input("Last name: ")
    email = input("Email: ")
    phone = input("Phone: ")
    username = input("Username: ")
    password = input("Password: ")
    role = input("Role('super admin', 'fleet manager'): ")

    admin = Admin(None, first, last, email, phone, username, password, role, None)
    admin_service.register_admin(admin)
    print("Admin registered successfully.")

elif choice == '2':
    admin_id = int(input("Admin ID: "))
    admin = admin_service.get_admin_by_id(admin_id)
    print(admin)

elif choice == '3':
    username = input("Enter Username: ")
    admin = admin_service.get_admin_by_username(username)
    print(admin)

elif choice == '4':
    admin_id = int(input("Admin ID: "))
    first_name = input("Enter first name: ")
    last_name = input("Enter last name: ")
    email = input("New Email: ")
    phone = input("New Phone: ")
    username = input("New Username: ")
    role = input("Role('super admin', 'fleet manager'): ")
    admin_service.update_admin(admin_id,first_name,last_name ,email, phone,username, role)
    print("Admin updated.")

elif choice == '5':
    admin_id = int(input("Admin ID: "))
    admin_service.delete_admin(admin_id)
    print("Admin deleted.")
```

```
def customer_menu():
    print("\n--- Customer Menu ---")
    print("1. Register Customer")
    print("2. Get Customer by ID")
    print("3. Get Customer by Username")
    print("4. Update Customer")
    print("5. Delete Customer")
    print("6. Authenticate Customer")

    choice = input("Enter choice: ")

    if choice == '1':
        first = input("First name: ")
        last = input("Last name: ")
        email = input("Email: ")
        phone = input("Phone: ")
        address = input("Address: ")
        username = input("Username: ")
        password = input("Password: ")

        customer = Customer(None, first, last, email, phone, address, username, password, None)
        customer_service.register_customer(customer)
        print("Customer registered.")

    elif choice == '2':
        customer_id = int(input("Customer ID: "))
        customer = customer_service.get_customer_by_id(customer_id)
        print(customer)

    elif choice == '3':
        username = input("Enter Username: ")
        customer = customer_service.get_customer_by_username(username)
        print(customer)

    elif choice == '4':
        customer_id = int(input("Customer ID: "))
        first_name = input("First Name: ")
        last_name = input("Last Name: ")
        email = input("New Email: ")
        phone = input("New Phone: ")
```

```
        address = input("New Address: ")
        username = input("Username: ")
        customer_service.update_customer(customer_id,first_name,last_name, email, phone,
address,username)
        print("Customer updated.")

    elif choice == '5':
        customer_id = int(input("Customer ID: "))
        customer_service.delete_customer(customer_id)
        print("Customer deleted.")

    elif choice == '6':
        username = input("Username: ")
        password = input("Password: ")
        customer = customer_service.authenticate_customer(username, password)
        print(customer)
        print("Authentication successful!")

def vehicle_menu():
    print("\n--- Vehicle Menu ---")
    print("1. Add Vehicle")
    print("2. Get Vehicle by ID")
    print("3. List Available Vehicles")
    print("4. Update Vehicle")
    print("5. Remove Vehicle")

    choice = input("Enter choice: ")

    if choice == '1':
        model = input("Model: ")
        make = input("Make: ")
        year = int(input("Year: "))
        color = input("Color: ")
        reg_no = input("Registration Number: ")
        availability = int(input("Availability (1/0): "))
        daily_rate = float(input("Daily Rate: "))

        vehicle = Vehicle(None, model, make, year, color, reg_no, availability, daily_rate)
        vehicle_service.add_vehicle(vehicle)
        print("Vehicle added.")
```

```
    elif choice == '2':
        vehicle_id = int(input("Vehicle ID: "))
        vehicle = vehicle_service.get_vehicle_by_id(vehicle_id)
        print(vehicle)

    elif choice == '3':
        vehicles = vehicle_service.get_available_vehicles()
        for v in vehicles:
            print(v)

    elif choice == '4':
        vehicle_id = int(input("Vehicle ID: "))
        rate = float(input("New Daily Rate: "))
        availability = int(input("Availability (1/0): "))
        vehicle_service.update_vehicle(vehicle_id, rate, availability)
        print("Vehicle updated.")

    elif choice == '5':
        vehicle_id = int(input("Vehicle ID: "))
        vehicle_service.remove_vehicle(vehicle_id)
        print("Vehicle removed.")

def reservation_menu():
    print("\n--- Reservation Menu ---")
    print("1. Create Reservation")
    print("2. Get Reservation by ID")
    print("3. Get Reservations by Customer ID")
    print("4. Update Reservation Status")
    print("5. Cancel Reservation")
    print("6. Reservation History Report")
    print("7. Generate Vehicle Report")
    print("8. Generate Revenue Report")

    choice = input("Enter choice: ")

    if choice == '1':
        customer_id = int(input("Customer ID: "))
        vehicle_id = int(input("Vehicle ID: "))
        start_date = input("Start Date (YYYY-MM-DD): ")
```

```
        end_date = input("End Date (YYYY-MM-DD): ")
        total_cost = float(input("Total Cost: "))
        status = input("Status: ")

        reservation = Reservation(None, customer_id, vehicle_id, start_date, end_date, total_cost,
status)
        reservation_service.create_reservation(reservation)
        print("Reservation created.")

    elif choice == '2':
        reservation_id = int(input("Reservation ID: "))
        reservation = reservation_service.get_reservation_by_id(reservation_id)
        print(reservation)

    elif choice == '3':
        customer_id = int(input("Customer ID: "))
        reservations = reservation_service.get_reservations_by_customer_id(customer_id)

    elif choice == '4':
        reservation_id = int(input("Reservation ID: "))
        status = input("New Status: ")
        reservation_service.update_reservation(reservation_id, status)
        print("Reservation updated.")

    elif choice == '5':
        reservation_id = int(input("Reservation ID: "))
        reservation_service.cancel_reservation(reservation_id)
        print("Reservation canceled.")

    elif choice == '6':
        reservation_service.generate_reservation_history_report()

    elif choice == '7':
        reservation_service.generate_vehicle_utilization_report()

    elif choice == '8':
        reservation_service.generate_revenue_report()

def main():
    while True:
```

```
        print("\n===== CarConnect Main Menu =====")
        print("1. Admin Services")
        print("2. Customer Services")
        print("3. Vehicle Services")
        print("4. Reservation Services")
        print("0. Exit")

        option = input("Select option: ")

        if option == '1':
            admin_menu()
        elif option == '2':
            customer_menu()
        elif option == '3':
            vehicle_menu()
        elif option == '4':
            reservation_menu()
        elif option == '0':
            print("Exiting CarConnect...")
            break
        else:
            print("Invalid option. Try again.")

if __name__ == "__main__":
    main()
```

**TESTING:**

**test_admin_service.py:**

```
import unittest
from unittest.mock import MagicMock
from CarConnect.entity.admin import Admin
from CarConnect.dao.admin_service import AdminService

class TestAdminService(unittest.TestCase):
    def setUp(self):
        self.mock_db = MagicMock()
        self.service = AdminService(self.mock_db)

    def test_get_admin_by_id_valid(self):
        self.mock_db.fetch_query.return_value = [("John", "Doe")]
        self.service.get_admin_by_id(1)
        self.mock_db.fetch_query.assert_called_once()

    def test_get_admin_by_username_valid(self):
        self.mock_db.fetch_query.return_value = [("admin1", "Admin")]
        self.service.get_admin_by_username("admin1")
        self.mock_db.fetch_query.assert_called_once()

    def test_register_admin_valid(self):
        admin = Admin("John", "Doe", "john@example.com", "1234567890", "admin1",
"pass123", "Manager")
        self.service.register_admin(admin)
        self.mock_db.execute_query.assert_called_once()

    def test_update_admin_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.update_admin(1, "John", "Smith", "johnsmith@example.com", "9876543210",
"johnsmith", "Admin")
        self.mock_db.execute_query.assert_called_once()

    def test_delete_admin_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.delete_admin(1)
        self.mock_db.execute_query.assert_called_once()

if __name__ == "__main__":
    unittest.main()
```

**test_customer_service.py:**

```
import unittest
from unittest.mock import patch, MagicMock
from CarConnect.dao.customer_service import CustomerService
from CarConnect.exceptions.authentication_exception import AuthenticationException
from CarConnect.exceptions.invalid_input_exception import InvalidInputException

class TestCustomerAuthentication(unittest.TestCase):
    def setUp(self):
        mock_db = MagicMock()
        self.customer_service = CustomerService(mock_db)

    @patch('builtins.input', side_effect=["meens", "meens"])
    def test_authentication_with_user_input(self, mock_inputs):
        username = input("Enter username: ")
        password = input("Enter password: ")
        try:
            self.customer_service.authenticate_customer(username, password)
            print("Authentication successful")
        except (AuthenticationException, InvalidInputException) as e:
            print(f"Authentication failed: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    @patch('builtins.input', side_effect=["1", "newemail@example.com", "1234567890", "New
Address"])
    def test_update_customer_info_with_input(self, mock_inputs):
        try:
            customer_id = int(input("Enter customer ID: "))
            email = input("Enter new email: ")
            phone = input("Enter new phone: ")
            address = input("Enter new address: ")
            self.customer_service.update_customer(customer_id, email, phone, address)
            print("Customer info updated")
        except InvalidInputException as e:
            print(f"Invalid input: {e}")
        except Exception as e:
            print(f"Update failed: {e}")
```

```
if __name__ == '__main__':
    unittest.main()
```

**test_reservation_service.py:**

```
import unittest
from unittest.mock import MagicMock
from datetime import date
from CarConnect.entity.reservation import Reservation
from CarConnect.dao.reservation_service import ReservationService

class TestReservationService(unittest.TestCase):
    def setUp(self):
        self.mock_db = MagicMock()
        self.service = ReservationService(self.mock_db)

    def test_get_reservation_by_id_valid(self):
        self.mock_db.fetch_query.return_value = [(1, 2, 3, date(2024, 5, 1), date(2024, 5, 5),
5000.00, "Confirmed")]
        reservation = self.service.get_reservation_by_id(1)
        self.assertIsInstance(reservation, Reservation)
        self.mock_db.fetch_query.assert_called_once()

    def test_get_reservations_by_customer_id_valid(self):
        self.mock_db.fetch_query.return_value = [
            (1, 2, 3, date(2024, 5, 1), date(2024, 5, 5), 5000.00, "Confirmed"),
            (2, 2, 4, date(2024, 6, 1), date(2024, 6, 3), 3000.00, "Pending"),
        ]
        reservations = self.service.get_reservations_by_customer_id(2)
        self.assertTrue(all(isinstance(r, Reservation) for r in reservations))
        self.mock_db.fetch_query.assert_called_once()

    def test_create_reservation_valid(self):
        reservation = Reservation(1, 2, 3, date(2024, 5, 1), date(2024, 5, 5), 5000.00, "Confirmed")
        self.service.create_reservation(reservation)
        self.mock_db.execute_query.assert_called_once()

    def test_update_reservation_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.update_reservation(1, "Cancelled")
```

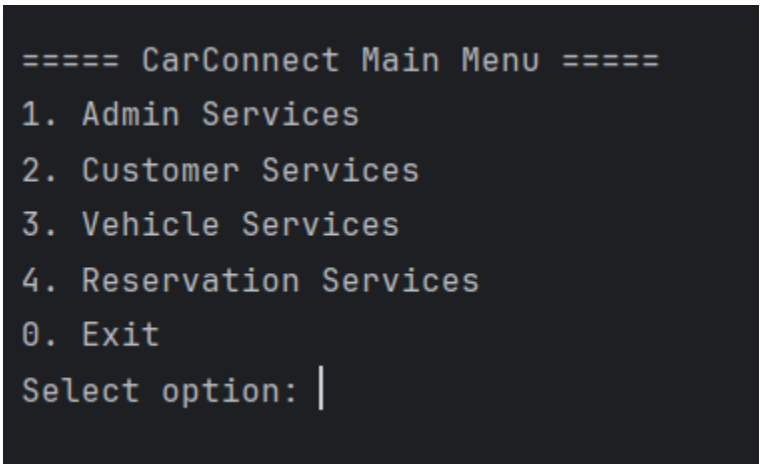
```
        self.mock_db.execute_query.assert_called_once()

    def test_cancel_reservation_valid(self):
        self.mock_db.execute_query.return_value = 1
        self.service.cancel_reservation(1)
        self.mock_db.execute_query.assert_called_once()

if __name__ == "__main__":
    unittest.main()
```

**test_vehicle_service.py:**

```
import unittest
from unittest.mock import MagicMock
from CarConnect.entity.vehicle import Vehicle
from CarConnect.dao.vehicle_service import VehicleService
from CarConnect.exceptions.vehicle_not_found_exception import VehicleNotFoundException

class TestVehicleService(unittest.TestCase):
    def setUp(self):
        self.mock_db = MagicMock()
        self.mock_db.fetch_query.return_value = [
            (1, "Tesla", "Model S", 2023, "Black", "TS1234", 1, 3500.50, 1),
            (2, "Toyota", "Camry", 2022, "White", "TN9876", 1, 2500.00, 1)
        ]
        self.service = VehicleService(self.mock_db)

    def test_add_vehicle(self):
        vehicle = Vehicle(1, "Tesla", "Model S", 2023, "Black", "TS1234", 1, 3500.50)
        try:
            self.service.add_vehicle(vehicle)
            print("Vehicle added successfully.")
        except Exception as e:
            self.fail(f"Vehicle addition failed: {e}")

    def test_update_vehicle(self):
        try:
            self.service.update_vehicle(1, 4000.00, 0)
            print("Vehicle updated successfully.")
        except Exception as e:
```

```
            self.fail(f"Vehicle update failed: {e}")

    def test_get_available_vehicles(self):
        try:
            vehicles = self.service.get_available_vehicles()
            self.assertIsInstance(vehicles, list)
            print("Available vehicles fetched successfully.")
        except VehicleNotFoundException:
            print("No available vehicles found.")
        except Exception as e:
            self.fail(f"Fetching available vehicles failed: {e}")

if __name__ == "__main__":
    unittest.main()
```

**MAIN OUTPUT:**



**ADMIN SERVICE:**

```
===== CarConnect Main Menu =====
1. Admin Services
2. Customer Services
3. Vehicle Services
4. Reservation Services
0. Exit
Select option: 1

--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice:
```

**REGISTER ADMIN:**

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 1
First name: anush
Last name: kumar
Email: anush@gmail.com
Phone: 9874563211
Username: anush
Password: 123
Role('super admin', 'fleet manager'): super admin
Successful!!!
Admin registered successfully.
```

## GET ADMIN ID:

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 2
Admin ID: 11
Data retrieved successfully!
The Admin is: [(11, 'anush', 'kumar', 'anush@gmail.com', '9874563211', 'anush', '123', 'super admin', datetime.datetime(2025, 4, 10, 20, 25, 34))]
None
```

## GET ADMIN BY USER:

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 3
Enter Username: anush
Data retrieved successfully!
The user is: [(11, 'anush', 'kumar', 'anush@gmail.com', '9874563211', 'anush', '123', 'super admin', datetime.datetime(2025, 4, 10, 20, 25, 34))]
None
```

## UPDATE ADMIN:

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 4
Admin ID: 11
Enter first name: anush
Enter last name: kumar
New Email: anushkumar@gmail.com
New Phone: 12364457
New Username: anush
Role('super admin', 'fleet manager'): super admin
Successful!!!
Admin updated.
```

**DELETE ADMIN:**

```
--- Admin Menu ---
1. Register Admin
2. Get Admin by ID
3. Get Admin by Username
4. Update Admin
5. Delete Admin
Enter choice: 5
Admin ID: 10
Successful!!!
Admin deleted.
```

**CUSTOMER SERVICE:**

```
===== CarConnect Main Menu =====
1. Admin Services
2. Customer Services
3. Vehicle Services
4. Reservation Services
0. Exit
Select option: 2

--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice:
```

**REGISTER CUSTOMER:**

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 1
First name: rishi
Last name: karthik
Email: rishi@gmail.com
Phone: 123698547
Address: london
Username: rishi
Password: 123
Successful!!!
Customer registered.
```

**GET CUSTOMER BY ID:**

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 2
Customer ID: 11
Data retrieved successfully!
The Customer:  [(11, 'rishi', 'karthik', 'rishi@gmail.com', '123698547', 'london', 'rishi', '123', datetime.datetime(2025, 4, 10, 20, 30, 7))]
None
```

## GET USER BU USERNAME:

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 3
Enter Username: kavitas
Data retrieved successfully!
The Customer by ID:  [(8, 'Kavita', 'Saxena', 'kavita.saxena@example.com', '3210987654', '78 Tilak Road, Ahmedabad', 'kavitas', 'secure123', datetime.datetime(2025, 4, 10, 20,
None
```

## UPDATE CUSTOMER:

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 4
Customer ID: 11
First Name: rishi
Last Name: karthik
New Email: rishikarthik@gmai.com
New Phone: 123698547
New Address: usa
Username: rishi
Successful!!!
Customer updated.
```

**DELETE CUSTOMER:**

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 5
Customer ID: 10
Successful!!!
Customer deleted.
```

**AUTHENTICATE CUSTOMER:**

```
--- Customer Menu ---
1. Register Customer
2. Get Customer by ID
3. Get Customer by Username
4. Update Customer
5. Delete Customer
6. Authenticate Customer
Enter choice: 6
Username: rishi
Password: 123
Data retrieved successfully!
The User is: [(11, 'rishi', 'karthik', 'rishikarthik@gmai.com', '123698547', 'usa', 'rishi', '123', datetime.datetime(2025, 4, 10, 20, 30, 7))]
None
Authentication successful!
```

**ADD CUSTOMER:**

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 1
Model: TVS
Make: lp
Year: 2020
Color: black
Registration Number: 3469
Availability (1/0): 1
Daily Rate: 12
Successful!!!
Vehicle added.
```

**GET VEHICLE BY ID:**

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 2
Vehicle ID: 11
Data retrieved successfully!
The vehicle is:  [(11, 'TVS', 'lp', 2020, 'black', '3469', 1, Decimal('12.00'))]
None
```

**List available vehicle:**

```
5. Remove Vehicle
Enter choice: 3
Data retrieved successfully!
(1, 'Polo', 'Volkswagen', 2022, 'Blue', 'MH01XY1234', 1, Decimal('1500.00'))
(2, 'Verna', 'Hyundai', 2021, 'Silver', 'DL02YZ5678', 1, Decimal('1800.00'))
(3, 'Thar', 'Mahindra', 2023, 'Black', 'KA03AB9101', 1, Decimal('2200.00'))
(4, 'Nexon', 'Tata', 2022, 'White', 'TN04CD1122', 1, Decimal('1600.00'))
(5, 'Seltos', 'Kia', 2021, 'Red', 'KL05EF3344', 1, Decimal('1900.00'))
(6, 'Brezza', 'Maruti', 2020, 'Orange', 'AP06GH5566', 1, Decimal('1700.00'))
(7, 'Compass', 'Jeep', 2023, 'Grey', 'GJ07IJ7788', 1, Decimal('2800.00'))
(8, 'Harrier', 'Tata', 2022, 'Brown', 'MP08KL9900', 1, Decimal('2100.00'))
(9, 'Sonet', 'Kia', 2021, 'Yellow', 'RJ09MN1112', 1, Decimal('1750.00'))
(10, 'XUV700', 'Mahindra', 2023, 'Blue', 'UP10OP1314', 1, Decimal('2400.00'))
(11, 'TVS', 'lp', 2020, 'black', '3469', 1, Decimal('12.00'))
```

**UPDATE VEHICLE:**

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 4
Vehicle ID: 11
New Daily Rate: 14
Availability (1/0): 1
Successful!!!
Vehicle updated.
```

**REMOVE VEHICLE:**

```
--- Vehicle Menu ---
1. Add Vehicle
2. Get Vehicle by ID
3. List Available Vehicles
4. Update Vehicle
5. Remove Vehicle
Enter choice: 5
Vehicle ID: 10
Successful!!!
Vehicle removed.
```

**CREATE RESERVATION:**

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 1
Customer ID: 11
Vehicle ID: 11
Start Date (YYYY-MM-DD): 2020-05-02
End Date (YYYY-MM-DD): 2020-05-15
Total Cost: 15000
Status: pending
Successful!!!
Reservation created.
```

**GET RESERVATION:**

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 2
Reservation ID: 1
Data retrieved successfully!
<entity.reservation.Reservation object at 0x0000018945EC8790>
```

**GET RESERVATION BY ID:**

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 3
Customer ID: 11
Data retrieved successfully!
Reservation ID: 21
Vehicle ID   : 11
Start Date   : 2020-05-02 00:00:00
End Date     : 2020-05-15 00:00:00
Total Cost   : 15000.00
Status       : pending
```

**UPDATE RESERVATION STATUS:**

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 4
Reservation ID: 21
New Status: completed
Successful!!!
Reservation updated.
```

**CANCEL RESERVATION:**

```
--- Reservation Menu ---
1. Create Reservation
2. Get Reservation by ID
3. Get Reservations by Customer ID
4. Update Reservation Status
5. Cancel Reservation
6. Reservation History Report
7. Generate Vehicle Report
8. Generate Revenue Report
Enter choice: 5
Reservation ID: 21
Successful!!!
Reservation canceled.
```

**RESRVATION HISTORY:**

```
8. Generate Revenue Report
Enter choice: 6
Data retrieved successfully!

--- Reservation History Report ---
(8, 9, 8, datetime.datetime(2025, 4, 15, 13, 0), datetime.datetime(2025, 4, 17, 13, 0), 'pending')
(18, 9, 8, datetime.datetime(2025, 4, 15, 13, 0), datetime.datetime(2025, 4, 17, 13, 0), 'pending')
(7, 6, 4, datetime.datetime(2025, 4, 13, 11, 0), datetime.datetime(2025, 4, 14, 11, 0), 'pending')
(17, 6, 4, datetime.datetime(2025, 4, 13, 11, 0), datetime.datetime(2025, 4, 14, 11, 0), 'pending')
(4, 5, 7, datetime.datetime(2025, 4, 12, 8, 0), datetime.datetime(2025, 4, 15, 8, 0), 'cancelled')
(14, 5, 7, datetime.datetime(2025, 4, 12, 8, 0), datetime.datetime(2025, 4, 15, 8, 0), 'cancelled')
(6, 7, 9, datetime.datetime(2025, 4, 9, 10, 0), datetime.datetime(2025, 4, 10, 10, 0), 'confirmed')
(16, 7, 9, datetime.datetime(2025, 4, 9, 10, 0), datetime.datetime(2025, 4, 10, 10, 0), 'confirmed')
(3, 1, 2, datetime.datetime(2025, 4, 8, 12, 0), datetime.datetime(2025, 4, 11, 12, 0), 'confirmed')
(13, 1, 2, datetime.datetime(2025, 4, 8, 12, 0), datetime.datetime(2025, 4, 11, 12, 0), 'confirmed')
(5, 3, 1, datetime.datetime(2025, 4, 6, 18, 0), datetime.datetime(2025, 4, 7, 18, 0), 'completed')
(15, 3, 1, datetime.datetime(2025, 4, 6, 18, 0), datetime.datetime(2025, 4, 7, 18, 0), 'completed')
(2, 4, 5, datetime.datetime(2025, 4, 5, 9, 0), datetime.datetime(2025, 4, 6, 9, 0), 'completed')
(12, 4, 5, datetime.datetime(2025, 4, 5, 9, 0), datetime.datetime(2025, 4, 6, 9, 0), 'completed')
(9, 8, 6, datetime.datetime(2025, 4, 2, 10, 0), datetime.datetime(2025, 4, 3, 10, 0), 'completed')
(19, 8, 6, datetime.datetime(2025, 4, 2, 10, 0), datetime.datetime(2025, 4, 3, 10, 0), 'completed')
(1, 2, 3, datetime.datetime(2025, 4, 1, 10, 0), datetime.datetime(2025, 4, 3, 10, 0), 'completed')
(11, 2, 3, datetime.datetime(2025, 4, 1, 10, 0), datetime.datetime(2025, 4, 3, 10, 0), 'completed')
```

**GENERATE VEHICLWE REPORT:**

```
Enter choice: 7
Data retrieved successfully!

--- Vehicle Utilization Report ---
Vehicle ID: 1, Reservations: 2
Vehicle ID: 2, Reservations: 2
Vehicle ID: 3, Reservations: 2
Vehicle ID: 4, Reservations: 2
Vehicle ID: 5, Reservations: 2
Vehicle ID: 6, Reservations: 2
Vehicle ID: 7, Reservations: 2
Vehicle ID: 8, Reservations: 2
Vehicle ID: 9, Reservations: 2
```

**GENERATE REVENUE REPORT:**

```
8. Generate Revenue Report
Enter choice: 8
Data retrieved successfully!

--- Revenue Report ---
Vehicle ID: 3, Revenue: ₹8800.00
Vehicle ID: 5, Revenue: ₹3800.00
Vehicle ID: 6, Revenue: ₹3400.00
Vehicle ID: 1, Revenue: ₹3000.00
```