

# CaseStudy - CarConnect

## SQL Tables

### Customer Table

```
1  -- Customer Table
2  CREATE TABLE Customer (
3      CustomerID INT PRIMARY KEY auto_increment,
4      FirstName VARCHAR(50),
5      LastName VARCHAR(50),
6      Email VARCHAR(100) UNIQUE,
7      PhoneNumber VARCHAR(15),
8      Address VARCHAR(255),
9      Username VARCHAR(50) UNIQUE,
10     Password VARCHAR(255),
11     RegistrationDate DATETIME
12 );
```

### Vehicle Table

```
1  -- Vehicle Table
2  CREATE TABLE Vehicle (
3      VehicleID INT PRIMARY KEY auto_increment,
4      Model VARCHAR(50),
5      Make VARCHAR(50),
6      Year INT,
7      Color VARCHAR(50),
8      RegistrationNumber VARCHAR(20) UNIQUE,
9      Availability BOOLEAN,
10     DailyRate DECIMAL(10, 2)
11 );
```

### Reservation Table

```
1  -- Reservation Table
2  CREATE TABLE Reservation (
3      ReservationID INT PRIMARY KEY auto_increment,
4      CustomerID INT,
5      VehicleID INT,
6      StartDate DATETIME,
7      EndDate DATETIME,
8      TotalCost DECIMAL(10, 2),
9      Status VARCHAR(20),
10     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
11     FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
12 );
```

### Admin Table

```
1  -- Admin Table
2  CREATE TABLE Admin (
3      AdminID INT PRIMARY KEY auto_increment,
4      FirstName VARCHAR(50),
5      LastName VARCHAR(50),
6      Email VARCHAR(100) UNIQUE,
7      PhoneNumber VARCHAR(15),
8      Username VARCHAR(50) UNIQUE,
9      Password VARCHAR(255),
10     Role VARCHAR(50),
11     JoinDate DATETIME
12 );
```

## Classes

### Customer

```
1 class Customer:
2     def __init__(self, customer_id, first_name, last_name, email, phone_number, address, username, password, registration_date):
3         self.__CustomerID = customer_id
4         self.__FirstName = first_name
5         self.__LastName = last_name
6         self.__Email = email
7         self.__PhoneNumber = phone_number
8         self.__Address = address
9         self.__Username = username
10        self.__Password = password
11        self.__RegistrationDate = registration_date
12
13    # Getter methods
14    1 usage
15    @property
16    def customer_id(self):
17        return self.__CustomerID
18
19    1 usage
20    @property
21    def first_name(self):
22        return self.__FirstName
23
24    1 usage
25    @property
26    def last_name(self):
27        return self.__LastName
28
29    1 usage
30    @property
31    def phone_number(self):
32        return self.__PhoneNumber
33
34    @phone_number.setter
35    def phone_number(self, phone_number):
36        self.__PhoneNumber = phone_number
37
38    @address.setter
39    def address(self, address):
40        self.__Address = address
41
42    @username.setter
43    def username(self, username):
44        self.__Username = username
45
46    @password.setter
47    def password(self, password):
48        self.__Password = password
49
50    @registration_date.setter
51    def registration_date(self, registration_date):
52        self.__RegistrationDate = registration_date
53
54    3 usages (3 dynamic)
55    def authenticate(self, password):
56        return self.__Password == password
```

## Vehicle

```
1 class Vehicle:
2     def __init__(self, vehicle_id, model, make, year, color, registration_number, availability, daily_rate):
3         self.__VehicleID = vehicle_id
4         self.__Model = model
5         self.__Make = make
6         self.__Year = year
7         self.__Color = color
8         self.__RegistrationNumber = registration_number
9         self.__Availability = availability
10        self.__DailyRate = daily_rate
11
12    # Getter methods with @property decorator
13    1 usage
14    @property
15    def vehicle_id(self):
16        return self.__VehicleID
17
18    1 usage
19    @property
20    def model(self):
21        return self.__Model
22
23    1 usage
24    @property
25    def make(self):
26        return self.__Make
27
28    1 usage
29    @property
30    def year(self):
31
32    @make.setter
33    def make(self, make):
34        self.__Make = make
35
36    @year.setter
37    def year(self, year):
38        self.__Year = year
39
40    @color.setter
41    def color(self, color):
42        self.__Color = color
43
44    @registration_number.setter
45    def registration_number(self, registration_number):
46        self.__RegistrationNumber = registration_number
47
48    @availability.setter
49    def availability(self, availability):
50        self.__Availability = availability
51
52    @daily_rate.setter
53    def daily_rate(self, daily_rate):
54        self.__DailyRate = daily_rate
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
```

## Reservation

```
1  class Reservation:
2      def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status):
3          self.__ReservationID = reservation_id
4          self.__CustomerID = customer_id
5          self.__VehicleID = vehicle_id
6          self.__StartDate = start_date
7          self.__EndDate = end_date
8          self.__TotalCost = total_cost
9          self.__Status = status
10
11     # Getter methods with @property decorator
12     @property
13     def reservation_id(self):
14         return self.__ReservationID
15
16     @property
17     def customer_id(self):
18         return self.__CustomerID
19
20     @property
21     def vehicle_id(self):
22         return self.__VehicleID
23
24     @property
25     def start_date(self):
26         return self.__StartDate
```

```
40     # Setter methods with @<property_name>.setter decorator
41     @reservation_id.setter
42     def reservation_id(self, reservation_id):
43         self.__ReservationID = reservation_id
44
45     @customer_id.setter
46     def customer_id(self, customer_id):
47         self.__CustomerID = customer_id
48
49     @vehicle_id.setter
50     def vehicle_id(self, vehicle_id):
51         self.__VehicleID = vehicle_id
52
53     @start_date.setter
54     def start_date(self, start_date):
55         self.__StartDate = start_date
56
57     @end_date.setter
58     def end_date(self, end_date):
59         self.__EndDate = end_date
60
61     @total_cost.setter
62     def total_cost(self, total_cost):
63         self.__TotalCost = total_cost
64
65     @status.setter
66     def status(self, status):
67         self.__Status = status
```

## Admin

3 usages

```
1 class Admin:
2     def __init__(self, admin_id, first_name, last_name, email, phone_number, username, password, role, join_da
3         self.__AdminID = admin_id
4         self.__FirstName = first_name
5         self.__LastName = last_name
6         self.__Email = email
7         self.__PhoneNumber = phone_number
8         self.__Username = username
9         self.__Password = password
10        self.__Role = role
11        self.__JoinDate = join_date
```

# Getter methods with @property decorator

1 usage

@property

```
15 def admin_id(self):
16     return self.__AdminID
```

1 usage

@property

```
19 def first_name(self):
20     return self.__FirstName
```

1 usage

@property

```
23 def last_name(self):
24     return self.__LastName
```

1 usage

@property

```
67 @phone_number.setter
68 def phone_number(self, phone_number):
69     self.__PhoneNumber = phone_number
```

@username.setter

```
72 def username(self, username):
73     self.__Username = username
```

@password.setter

```
76 def password(self, password):
77     self.__Password = password
```

@role.setter

```
80 def role(self, role):
81     self.__Role = role
```

@join\_date.setter

```
84 def join_date(self, join_date):
85     self.__JoinDate = join_date
```

3 usages (3 dynamic)

```
87 def authenticate(self, password):
88     return self.__Password == password
```

## Services

### CustomerService (implements ICustomerService)

```
2 usages
9 class CustomerService(ICustomerService):
10     def __init__(self, db_context):
11         self.db_context = db_context
12
13     2 usages
14     def get_customer_by_id(self, customer_id):
15         query = "SELECT * FROM Customer WHERE CustomerID = %s"
16         params = (customer_id,)
17         result = self.db_context.execute_query(query, params)
18         if result:
19             return Customer(**result[0])
20         else:
21             raise CustomerNotFoundException()
22
23     1 usage (1 dynamic)
24     def get_customer_by_username(self, username):
25         query = "SELECT * FROM Customer WHERE Username = %s"
26         params = (username,)
27         result = self.db_context.execute_query(query, params)
28         if result:
29             return Customer(*result[0])
30         else:
31             raise CustomerNotFoundException()
32
33     1 usage
34     def register_customer(self, customer_data):
35         InputValidator.validate_string(customer_data['FirstName'], field_name: "First Name")
36         InputValidator.validate_string(customer_data['LastName'], field_name: "Last Name")
37         InputValidator.validate_email(customer_data['Email'], field_name: "Email")
```

### VehicleService (implements IVehicleService)

```
6 class VehicleService(IVehicleService):
7     def __init__(self, db_context):
8         self.db_context = db_context
9
10     2 usages (1 dynamic)
11     def get_vehicle_by_id(self, vehicle_id):
12         query = "SELECT * FROM Vehicle WHERE VehicleID = %s"
13         params = (vehicle_id,)
14         result = self.db_context.execute_query(query, params)
15         if result:
16             return result
17         else:
18             raise VehicleNotFoundException(f"Vehicle with ID {vehicle_id} not found.")
19
20     2 usages
21     def get_available_vehicles(self):
22         query = "SELECT * FROM Vehicle WHERE Availability = True"
23         results = self.db_context.execute_query(query)
24         return results
25
26     1 usage
27     def add_vehicle(self, vehicle_data):
28         query = "INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate) V
29         params = (
30             vehicle_data['Model'],
31             vehicle_data['Make'],
```

## ReservationService (implements IReservationService)

```
6 class ReservationService(IReservationService):
7     def __init__(self, db_context):
8         self.db_context = db_context
9
10    1 usage (1 dynamic)
11    def get_reservation_by_id(self, reservation_id):
12        query = "SELECT * FROM Reservation WHERE ReservationID = %s"
13        params = (reservation_id,)
14        result = self.db_context.execute_query(query, params)
15        if result:
16            return Reservation(**result[0])
17
18    def get_reservations_by_customer_id(self, customer_id):
19        query = "SELECT * FROM Reservation WHERE CustomerID = %s"
20        params = (customer_id,)
21        results = self.db_context.execute_query(query, params)
22        return [Reservation(**res) for res in results]
23
24    1 usage
25    def create_reservation(self, reservation_data):
26        query = "INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status) VALUES"
27        params = (
28            reservation_data['CustomerID'],
29            reservation_data['VehicleID'],
30            reservation_data['StartDate'],
31            reservation_data['EndDate'],
```

## AdminService (implements IAdminService)

```
7 class AdminService(IAdminService):
8     def __init__(self, db_context):
9         self.db_context = db_context
10
11    2 usages
12    def get_admin_by_id(self, admin_id):
13        query = "SELECT * FROM Admin WHERE AdminID = %s"
14        params = (admin_id,)
15        result = self.db_context.execute_query(query, params)
16        if result:
17            return Admin(**result[0])
18        else:
19            raise AdminNotFoundException()
20
21    1 usage (1 dynamic)
22    def get_admin_by_username(self, username):
23        query = "SELECT * FROM Admin WHERE Username = %s"
24        params = (username,)
25        result = self.db_context.execute_query(query, params)
26        if result:
27            return Admin(**result[0])
28        else:
29            raise AdminNotFoundException()
30
31    1 usage
32    def register_admin(self, admin_data):
33        query = "INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDat
```

## DatabaseContext

```
6 class DatabaseContext:
7     def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
8         self.host = host
9         self.user = user
10        self.password = password
11        self.database = database
12        self.connection = None
13        self.cursor = None
14
15    1 usage
16    def connect(self):
17        try:
18            self.connection = mysql.connector.connect(
19                host=self.host,
20                user=self.user,
21                password=self.password,
22                database=self.database
23            )
24            self.cursor = self.connection.cursor()
25            print(f"Connected to the database: {self.database}")
26        except mysql.connector.Error as e:
27            raise DatabaseConnectionException(f"Error connecting to the database: {e}")
28
29    def disconnect(self):
30        try:
31            if self.connection:
32                self.connection.close()
```

## ReportGenerator

```
1 class ReportGenerator:
2     def __init__(self, db_context, reservation_service=None, vehicle_service=None):
3         self.reservation_service = reservation_service
4         self.vehicle_service = vehicle_service
5         self.db_context = db_context
6
7     def generate_reservation_report(self, reservation_id):
8         reservation = self.reservation_service.get_reservation_by_id(reservation_id)
9         if reservation:
10            report = f"Reservation Report\nReservation ID: {reservation.get_reservation_id()}\nCustomer: {rese
11            return report
12        return "Reservation not found."
13
14    def generate_vehicle_report(self, vehicle_id):
15        vehicle = self.vehicle_service.get_vehicle_by_id(vehicle_id)
16        if vehicle:
17            report = f"Vehicle Report\nVehicle ID: {vehicle.get_vehicle_id()}\nModel: {vehicle.get_model()}\nM
18            return report
19        return "Vehicle not found."
20
21    1 usage
22    def view_overall_revenue(self):
23        query = "SELECT SUM(TotalCost) AS OverallRevenue FROM Reservation"
24        result = self.db_context.execute_query(query)
25
26        if result:
27            overall_revenue = result[0]['OverallRevenue']
28            print(f"Overall Revenue: ${overall_revenue:.2f}")
29        else:
30            print("No revenue data available.")
```



## Authentication

```
4 class AuthenticationService:
5     def __init__(self, customer_service, admin_service):
6         self.customer_service = customer_service
7         self.admin_service = admin_service
8
9     1 usage
10    def authenticate_customer(self, username, password):
11        customer = self.customer_service.get_customer_by_username(username)
12        if not customer.authenticate(password):
13            raise AuthenticationException("Incorrect Username or Password")
14        if customer:
15            return customer
16        raise AuthenticationException()
17
18    1 usage
19    def authenticate_admin(self, username, password):
20        admin = self.admin_service.get_admin_by_username(username)
21        if not admin.authenticate(password):
22            raise AuthenticationException("Incorrect Username or Password")
23        if admin and admin.authenticate(password):
24            return admin
25        raise AuthenticationException()
```

## Interfaces

### ICustomerService

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

## IVehicleService

```
1 from abc import ABC, abstractmethod
2
3
4 2 usages
5 1 usage (1 dynamic)
6 class IVehicleService(ABC):
7     @abstractmethod
8     def get_vehicle_by_id(self, vehicle_id):
9         pass
10
11     @abstractmethod
12     def get_available_vehicles(self):
13         pass
14
15     @abstractmethod
16     def add_vehicle(self, vehicle_data):
17         pass
18
19     @abstractmethod
20     def update_vehicle(self, vehicle_data):
21         pass
22
23     @abstractmethod
24     def remove_vehicle(self, vehicle_id):
25         pass
```

## IReservationService

```
1 from abc import ABC, abstractmethod
2
3
4 2 usages
5 1 usage (1 dynamic)
6 class IReservationService(ABC):
7     @abstractmethod
8     def get_reservation_by_id(self, reservation_id):
9         pass
10
11     @abstractmethod
12     def get_reservations_by_customer_id(self, customer_id):
13         pass
14
15     @abstractmethod
16     def create_reservation(self, reservation_data):
17         pass
18
19     @abstractmethod
20     def update_reservation(self, reservation_data):
21         pass
22
23     @abstractmethod
24     def cancel_reservation(self, reservation_id):
25         pass
```

## IAdminService

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

## Connect your application to the SQL database

Database connection is done through mysql-python-connector.

DatabaseContext.py

```
7 def __init__(self, host='localhost', user='root', password='root', database='mydatabase'):
8     self.host = host
9     self.user = user
10    self.password = password
11    self.database = database
12    self.connection = None
13    self.cursor = None
14
15 1 usage
16 def connect(self):
17     try:
18         self.connection = mysql.connector.connect(
19             host=self.host,
20             user=self.user,
21             password=self.password,
22             database=self.database
23         )
24         self.cursor = self.connection.cursor()
25         print(f"Connected to the database: {self.database}")
26     except mysql.connector.Error as e:
27         raise DatabaseConnectionException(f"Error connecting to the database: {e}")
```

Initialized the connection before the menu display

MainModult.py

```
405 if __name__ == "__main__":
406     db_context = DatabaseContext(database="CarConnect")
407     db_context.connect()
408     interface = MainModule(db_context)
409     interface.main_menu()
```

## Exceptions

```
AdminNotFoundException.py x
3 usages
1 class AdminNotFoundException(Exception):
2     def __init__(self, message="Admin not found"):
3         self.message = message
4         super().__init__(self.message)
```

```
AdminNotFoundException.py AuthenticationException.py x
5 usages
1 class AuthenticationException(Exception):
2     def __init__(self, message="Authentication failed"):
3         self.message = message
4         super().__init__(self.message)
```

```
AdminNotFoundException.py AuthenticationException.py CustomerNotFoundException.py x
3 usages
1 class CustomerNotFoundException(Exception):
2     def __init__(self, message="Customer not found"):
3         self.message = message
4         super().__init__(self.message)
```

```
Exception.py x AuthenticationException.py CustomerNotFoundException.py DatabaseConnectionException.py x
5 usages
1 class DatabaseConnectionException(Exception):
2     def __init__(self, message="Database connection error"):
3         self.message = message
4         super().__init__(self.message)
```

```
AdminNotFoundException.py DatabaseConnectionException.py ReservationException.py InvalidInputException.py x
8 usages
1 class InvalidInputException(Exception):
2     def __init__(self, message="Invalid input"):
3         self.message = message
4         super().__init__(self.message)
```

```
AdminNotFoundException.py DatabaseConnectionException.py ReservationException.py InvalidInputException.py
4 usages
1 class ReservationException(Exception):
2     def __init__(self, message="Reservation error"):
3         self.message = message
4         super().__init__(self.message)
```

```
AdminNotFoundException.py DatabaseConnectionException.py ReservationException.py VehicleNotFoundException.py x
2 usages
1 class VehicleNotFoundException(Exception):
2     def __init__(self, message="Vehicle not found"):
3         self.message = message
4         super().__init__(self.message)
```

Used all exceptions in appropriate places in the code.

## Unit Testing

### 1. Test customer authentication with invalid credentials.

```
class TestCustomerAuthentication(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)
        self.auth_service = AuthenticationService(self.customer_service)

    def test_invalid_credentials(self):
        invalid_username = "notvatsal"
        invalid_password = "notroot"

        with self.assertRaises(AuthenticationException) as context:
            try:
                self.auth_service.authenticate_customer(invalid_username, invalid_password)
            except Exception as ex:
                print("Invalid Creds Exception: ", ex)
        self.assertIn(member="Incorrect Username or Password", str(context.exception))

    def test_valid_credentials(self):
        valid_username = "vatsal"
        valid_password = "root"

        try:
            self.auth_service.authenticate_customer(valid_username, valid_password)
        except AuthenticationException as e:
            self.fail(f"Unexpected exception raised: {e}")
```

### 2. Test updating customer information.

```
class TestCustomerUpdate(unittest.TestCase):
    def setUp(self):
        db_context = DatabaseContext(database="CarConnect")
        db_context.connect()
        self.customer_service = CustomerService(db_context)

    def test_update_customer_info(self):
        existing_customer_id = "1"
        updated_info = {
            'FirstName': 'newVatsal',
            'LastName': 'newPateL',
            'Email': 'newVatsal@gmail.com',
            'PhoneNumber': '+91 1234567890',
            'Address': 'new Address',
            'CustomerID': existing_customer_id
        }

        try:
            self.customer_service.update_customer(updated_info)

            updated_customer = self.customer_service.get_customer_by_id(existing_customer_id)
            self.assertEqual(updated_info['FirstName'], updated_customer.first_name)
            self.assertEqual(updated_info['LastName'], updated_customer.last_name)
            self.assertEqual(updated_info['Email'], updated_customer.email)
            self.assertEqual(updated_info['PhoneNumber'], updated_customer.phone_number)
            self.assertEqual(updated_info['Address'], updated_customer.address)

        except InvalidInputException as e:
            self.fail(f"Unexpected exception raised: {e}")
```

### 3. Test adding a new vehicle.

```
8 class TestAddNewVehicle(unittest.TestCase):
9     def setUp(self):
10         self.db_context = DatabaseContext(database="CarConnect")
11         self.db_context.connect()
12         self.vehicle_service = VehicleService(self.db_context)
13
14     def test_add_new_vehicle(self):
15         new_vehicle_data = {
16             'Model': 'r15',
17             'Make': 'Yamaha',
18             'Year': 2023,
19             'Color': 'Black',
20             'RegistrationNumber': 'GJ05123',
21             'Availability': 'y', # y for True and n for False
22             'DailyRate': 500.00,
23         }
24
25         try:
26             self.vehicle_service.add_vehicle(new_vehicle_data)
27             curr_cursor = self.db_context.get_current_cursor()
28             new_vehicle_id = curr_cursor.lastrowid
29
30             added_vehicle_result = self.vehicle_service.get_vehicle_by_id(new_vehicle_id)
31             added_vehicle = Vehicle(*added_vehicle_result[0])
32
33             self.assertIsNotNone(added_vehicle)
34             self.assertEqual(new_vehicle_data['Model'], added_vehicle.model)
35             self.assertEqual(new_vehicle_data['Make'], added_vehicle.make)
36             self.assertEqual(new_vehicle_data['Year'], added_vehicle.year)
37             self.assertEqual(new_vehicle_data['Color'], added_vehicle.color)
38             self.assertEqual(new_vehicle_data['RegistrationNumber'], added_vehicle.registration_number)
39             self.assertEqual(new_vehicle_data['Availability'], 'y' if added_vehicle.availability == 1 else 'n')
40             self.assertEqual(new_vehicle_data['DailyRate'], added_vehicle.daily_rate)
41
42         except Exception as e:
43             self.fail(f"Exception raised: {e}")
```

#### 4. Test updating vehicle details.

```
8 class TestUpdateVehicleDetails(unittest.TestCase):
9     def setUp(self):
10         self.db_context = DatabaseContext(database="CarConnect")
11         self.db_context.connect()
12         self.vehicle_service = VehicleService(self.db_context)
13
14     def test_update_vehicle_details(self):
15         updated_vehicle_data = {
16             'VehicleID': 1,
17             'Model': 'Updated Model',
18             'Make': 'Updated Make',
19             'Year': 2023,
20             'Color': 'Updated Color',
21             'RegistrationNumber': 'Updated123',
22             'Availability': False,
23             'DailyRate': 60.0
24         }
25         self.vehicle_service.update_vehicle(updated_vehicle_data)
26
27         updated_vehicle_result = self.vehicle_service.get_vehicle_by_id(updated_vehicle_data['VehicleID'])
28         updated_vehicle = Vehicle(*updated_vehicle_result[0])
29
30         # Check if the details have been updated correctly
31         self.assertEqual(updated_vehicle.model, second: 'Updated Model')
32         self.assertEqual(updated_vehicle.make, second: 'Updated Make')
33         self.assertEqual(updated_vehicle.year, second: 2023)
34         self.assertEqual(updated_vehicle.color, second: 'Updated Color')
35         self.assertEqual(updated_vehicle.registration_number, second: 'Updated123')
36         self.assertFalse(updated_vehicle.availability)
37         self.assertEqual(updated_vehicle.daily_rate, second: 60.0)
```

## 5. Test getting a list of available vehicles.

```
8 class TestGetAvailableVehicles(unittest.TestCase):
9     def setUp(self):
10         db_context = DatabaseContext(database="CarConnect")
11         db_context.connect()
12         self.vehicle_service = VehicleService(db_context)
13
14     def test_get_available_vehicles(self):
15         test_vehicles = [
16             {
17                 'Model': 'Subaru BRZ',
18                 'Make': 'Subaru',
19                 'Year': 2022,
20                 'Color': 'Dark Gray',
21                 'RegistrationNumber': 'JP202212',
22                 'Availability': 'y',
23                 'DailyRate': 700.00,
24             },
25             {
26                 'Model': 'Mitsubishi Eclipse Cross',
27                 'Make': 'Mitsubishi',
28                 'Year': 2023,
29                 'Color': 'Deep Blue',
30                 'RegistrationNumber': 'JP202312',
31                 'Availability': 'n',
32                 'DailyRate': 750.00,
33             },
34             {
35                 'Model': 'Honda HR-V',
36                 'Make': 'Honda',
37                 'Year': 2021,
38                 'Color': 'Burgundy',
39                 'RegistrationNumber': 'JP202112',
40                 'Availability': 'y',
41                 'DailyRate': 720.00,
42             },
43         ]
44
45         for vehicle_data in test_vehicles:
46             self.vehicle_service.add_vehicle(vehicle_data)
47
48         available_vehicles_result = self.vehicle_service.get_available_vehicles()
49         available_vehicles = [Vehicle(*available_vehicle_result) for available_vehicle_result in available_veh
50         for vehicle in available_vehicles:
51             self.assertEqual(vehicle.availability, second: 1)
52
```



## 6. Test getting a list of all vehicles.

```
7 class TestGetAllVehicles(unittest.TestCase):
8     def setUp(self):
9         db_context = DatabaseContext(database="CarConnect")
10        db_context.connect()
11        self.vehicle_service = VehicleService(db_context)
12
13    def test_get_all_vehicles(self):
14        test_vehicles = [
15            {
16                'Model': 'Toyota Corolla',
17                'Make': 'Toyota',
18                'Year': 2022,
19                'Color': 'Silver',
20                'RegistrationNumber': 'XYZ1239',
21                'Availability': 'y',
22                'DailyRate': 50.00,
23            },
24            {
25                'Model': 'Honda Accord',
26                'Make': 'Honda',
27                'Year': 2023,
28                'Color': 'Blue',
29                'RegistrationNumber': 'ABC4569',
30                'Availability': 'y',
31                'DailyRate': 60.00,
32            },
33            {
34                'Model': 'Ford Mustang',
35                'Make': 'Ford',
36                'Year': 2021,
37                'Color': 'Red',
38                'RegistrationNumber': 'DEF7899',
39                'Availability': 'n',
40                'DailyRate': 70.00,
41            },
42        ]
43
44        for vehicle_data in test_vehicles:
45            self.vehicle_service.add_vehicle(vehicle_data)
46
47        all_vehicles_result = self.vehicle_service.get_all_vehicles()
48        all_vehicles = [Vehicle(*vehicle_result) for vehicle_result in all_vehicles_result]
49
50        self.assertGreaterEqual(len(all_vehicles), len(test_vehicles))
51
```

## Conclusion

Overall it is a full-fledged backend and database connection implementation. I recommend you check the project file by file to see all the features and things implemented.

**Thank You!**

\*\*\*\*\*