Case study

CarConnect, a Car Rental Platform

By Akash Kumar

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

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.

```sql
CREATE TABLE Customer (
    CustomerID INT AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(20) ,
    Address VARCHAR(255),
    Username VARCHAR(50) UNIQUE,
    Password VARCHAR(255),
    RegistrationDate DATE,
    primary key (CustomerID)
);
```

```sql
INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
VALUES
('John', 'Doe', 'john.doe@example.com', '123-456-7890', '123 Main St, Anytown, USA', 'johndoe123', 'password123', '2023-01-15'),
('Jane', 'Smith', 'jane.smith@example.com', '987-654-3210', '456 Elm St, Othertown, USA', 'janesmith456', 'letmein456', '2023-02-20'),
('Michael', 'Johnson', 'michael.johnson@example.com', '555-555-5555', '789 Oak St, Anycity, USA', 'michaelj88', 'securepassword', '2023-03-10'),
('Emily', 'Brown', 'emily.brown@example.com', '111-222-3333', '321 Pine St, Newville, USA', 'emilyb', 'p@ssw0rd', '2023-04-05'),
('David', 'Wilson', 'david.wilson@example.com', '444-444-4444', '654 Maple St, Townsville, USA', 'davidwilson', 'david123', '2023-05-12'),
('Sarah', 'Martinez', 'sarah.martinez@example.com', '999-999-9999', '987 Cedar St, Smalltown, USA', 'sarahm', 'ilovesarah', '2023-06-20'),
('Christopher', 'Anderson', 'chris.anderson@example.com', '777-777-7777', '852 Walnut St, Bigcity, USA', 'canderson', 'chrisA123', '2023-07-18')
('Jessica', 'Garcia', 'jessica.garcia@example.com', '333-333-3333', '369 Cherry St, Metropolis, USA', 'jgarcia', 'jessgarc', '2023-08-30'),
('Matthew', 'Taylor', 'matthew.taylor@example.com', '666-666-6666', '741 Birch St, Uptown, USA', 'mattt', 'taylormade', '2023-09-10'),
('Amanda', 'Hernandez', 'amanda.hernandez@example.com', '888-888-8888', '147 Ivy St, Downtown, USA', 'amandah', 'amandapass', '2023-10-25');
```

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.

```sql
CREATE TABLE Vehicle (
    VehicleID INT AUTO_INCREMENT,
    Model VARCHAR(100),
    Make VARCHAR(100),
    Year INT,
    Color VARCHAR(50),
    RegistrationNumber VARCHAR(20) UNIQUE,
    Availability BOOLEAN,
    DailyRate DECIMAL,
    primary key (vehicleID)
);
```

```sql
INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
VALUES
('Civic', 'Honda', 2020, 'Blue', 'ABC123', TRUE, 50.00),
('Accord', 'Honda', 2019, 'Red', 'DEF456', TRUE, 60.00),
('Camry', 'Toyota', 2021, 'Black', 'GHI789', TRUE, 55.00),
('Corolla', 'Toyota', 2018, 'Silver', 'JKL012', TRUE, 45.00),
('Altima', 'Nissan', 2020, 'White', 'MNO345', TRUE, 55.00),
('Sentra', 'Nissan', 2019, 'Gray', 'PQR678', TRUE, 50.00),
('Fusion', 'Ford', 2021, 'Green', 'STU901', TRUE, 65.00),
('Focus', 'Ford', 2017, 'Yellow', 'VWX234', TRUE, 40.00),
('Impala', 'Chevrolet', 2020, 'Brown', 'YZA567', TRUE, 60.00),
('Malibu', 'Chevrolet', 2019, 'Orange', 'BCD890', TRUE, 55.00);
```

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).

```sql
CREATE TABLE Reservation (
    ReservationID INT AUTO_INCREMENT,
    CustomerID INT,
    VehicleID INT,
    StartDate DATETIME,
    EndDate DATETIME,
    TotalCost DECIMAL,
    Status ENUM('pending', 'confirmed', 'completed'),
    primary key (ReservationID),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
);
```

```sql
INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
VALUES
(1, 1, '2024-02-10 09:00:00', '2024-02-15 18:00:00', 250.00, 'confirmed'),
(2, 3, '2024-02-12 10:00:00', '2024-02-18 15:00:00', 330.00, 'confirmed'),
(3, 5, '2024-02-14 11:00:00', '2024-02-16 17:00:00', 110.00, 'completed'),
(4, 7, '2024-02-15 12:00:00', '2024-02-20 12:00:00', 390.00, 'confirmed'),
(5, 9, '2024-02-18 13:00:00', '2024-02-21 14:00:00', 180.00, 'pending'),
(6, 2, '2024-02-20 14:00:00', '2024-02-22 16:00:00', 120.00, 'pending'),
(7, 4, '2024-02-22 15:00:00', '2024-02-25 12:00:00', 135.00, 'confirmed'),
(8, 6, '2024-02-24 16:00:00', '2024-02-27 10:00:00', 200.00, 'pending'),
(9, 8, '2024-02-25 08:00:00', '2024-02-28 18:00:00', 195.00, 'pending'),
(10, 10, '2024-02-28 09:00:00', '2024-03-02 09:00:00', 120.00, 'confirmed');
```

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.

```sql
CREATE TABLE Admin (
    AdminID INT AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    PhoneNumber bigint,
    Username VARCHAR(50) UNIQUE,
    Password VARCHAR(255),
    Role VARCHAR(50),
    JoinDate DATE,
    primary key (AdminID)
);
```

```sql
INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
VALUES
('Admin', 'One', 'admin1@example.com', 1234567890, 'admin1', 'admin1pass', 'SuperAdmin', '2023-01-01'),
('Admin', 'Two', 'admin2@example.com', 9876543210, 'admin2', 'admin2pass', 'Admin', '2023-02-01'),
('Admin', 'Three', 'admin3@example.com', 5555555555, 'admin3', 'admin3pass', 'Admin', '2023-03-01'),
('Admin', 'Four', 'admin4@example.com', 1112223333, 'admin4', 'admin4pass', 'Admin', '2023-04-01'),
('Admin', 'Five', 'admin5@example.com', '4444444444', 'admin5', 'adminpassword', 'Admin', '2024-01-05'),
('Admin', 'Six', 'admin6@example.com', '9999999999', 'admin6', 'adminpassword', 'Admin', '2024-01-06'),
('Admin', 'Seven', 'admin7@example.com', '7777777777', 'admin7', 'adminpassword', 'Admin', '2024-01-07'),
('Admin', 'Eight', 'admin8@example.com', '8888888888', 'admin8', 'adminpassword', 'Admin', '2024-01-08'),
('Admin', 'Nine', 'admin9@example.com', '6666666666', 'admin9', 'adminpassword', 'Admin', '2024-01-09'),
('Admin', 'Ten', 'admin10@example.com', '3333333333', 'admin10', 'adminpassword', 'Admin', '2024-01-10');
```

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

Classes:

• Customer:

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

```python
class Customer:
    def __init__(self,CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate ):
        self.CustomerID = CustomerID
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.PhoneNumber = PhoneNumber
        self.Address = Address
        self.Username = Username
        self.Password = Password
        self.RegistrationDate = RegistrationDate
```

```python
    @property
    def getCustomerid(self):
        return self.CustomerID

    @getCustomerid.setter
    def setCustomerid(self,newCustomerid):
        self.CustomerID = newCustomerid

    @property
    def getFirstName(self):
        return self.FirstName

    @getFirstName.setter
    def setFirstName(self, newFirstName):
        self.FirstName = newFirstName

    @property
    def getEmail(self):
        return self.Email

    @getEmail.setter
    def setEmail(self, newEmail):
        self.Email = newEmail

    @property
    def getPhoneNumber(self):
        return self.PhoneNumber

    @getPhoneNumber.setter
    def setPhoneNumber(self, newPhoneNumber):
        self.PhoneNumber = newPhoneNumber
```

```python
    @property
    def getAddress(self):
        return self.Address

    @getAddress.setter
    def setAddress(self, newAddress):
        self.Address = newAddress

    @property
    def getUsername(self):
        return self.Username

    @getUsername.setter
    def setUsername(self, newUsername):
        self.Username = newUsername

    @property
    def getRegistrationDate(self):
        return self.RegistrationDate

    @getRegistrationDate.setter
    def setUsername(self, newRegistrationDate):
        self.RegistrationDate= newRegistrationDate
```

• Vehicle:

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

```python
class Vehicle:
    def __init__(self,VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate ):
        self.VehicleID = VehicleID
        self.Model =Model
        self.Make = Make
        self.Year = Year
        self.Color = Color
        self.RegistrationNumber = RegistrationNumber
        self.Availability = Availability
        self.DailyRate = DailyRate
```

```python
    @property
    def getVehicleID(self):
        return self.VehicleID
    @getVehicleID.setter
    def setVehicleID(self,newVehicleID):
        self.VehicleID = newVehicleID

    @property
    def getModel(self):
        return self.Model

    @getModel.setter
    def setModel(self, newModel):
        self.Model = newModel

    @property
    def getMake(self):
        return self.Make

    @getMake.setter
    def setModel(self, newMake):
        self.Make = newMake

    @property
    def getYear(self):
        return self.Year

    @getYear.setter
    def setModel(self, newYear):
        self.Year = newYear
```

```python
    @property
    def getColor(self):
        return self.Color

    @getColor.setter
    def setColor(self, newColor):
        self.Color = newColor

    @property
    def getRegistrationNumber(self):
        return self.RegistrationNumber

    @getRegistrationNumber.setter
    def setRegistrationNumber(self, newRegistrationNumber):
        self.RegistrationNumber = newRegistrationNumber

    @property
    def getAvailability(self):
        return self.Availability

    @getAvailability.setter
    def setAvailability(self, newAvailability):
        self.Availability = newAvailability

    @property
    def getDailyRate(self):
        return self.DailyRate

    @getDailyRate.setter
    def setColor(self, newDailyRate):
        self.DailyRate = newDailyRate
```

• Reservation:

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

```python
class Reservation:
    def __init__(self,ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status):
        self.ReservationID = ReservationID
        self.CustomerID = CustomerID
        self.VehicleID = VehicleID
        self.StartDate = StartDate
        self.EndDate = EndDate
        self.TotalCost = TotalCost
        self.Status = Status
```

```python
    @property
    def getReservaionID(self):
        return self.ReservationID

    @getReservaionID.setter
    def setReservaionID(self,newReservaionID):
        self.ReservationID = newReservaionID

    @property
    def getCustomerID(self):
        return self.CustomerID

    @getCustomerID.setter
    def setReservaionID(self, newCustomerID):
        self.CustomerID = newCustomerID

    @property
    def getVehicleID(self):
        return self.VehicleID

    @getVehicleID.setter
    def setVehicleID(self, newVehicleID):
        self.VehicleID = newVehicleID

    @property
    def getStartDate(self):
        return self.StartDate

    @getStartDate.setter
    def setStartDate(self, newStartDate):
        self.StartDate = newStartDate
```

```python
    @property
    def getEndDate(self):
        return self.EndDate

    @getEndDate.setter
    def setEndDate(self, newEndDate):
        self.EndDate = newEndDate

    @property
    def getTotalCost(self):
        return self.TotalCost

    @getTotalCost.setter
    def setTotalCost(self, newTotalCost):
        self.TotalCost = newTotalCost

    @property
    def getStatus(self):
        return self.Status

    @getStatus.setter
    def setStatus(self, newStatus):
        self.Status = newStatus
```

• Admin:

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

```python
class Admin:
    def __init__(self, AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate):
        self.AdminID = AdminID
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.PhoneNumber = PhoneNumber
        self.Username = Username
        self.Password = Password
        self.Role = Role
        self.JoinDate = JoinDate
```

```python
    @property
    def getAdminID(self):
        return self.AdminID
    @getAdminID.setter
    def setAdminID(self,newAdminID):
        self.AdminID = newAdminID

    @property
    def getFirstName(self):
        return self.FirstName

    @getFirstName.setter
    def setFirstName(self, newFirstName):
        self.FirstName = newFirstName

    @property
    def getLastName(self):
        return self.LastName

    @getLastName.setter
    def setLastName(self, newLastName):
        self.LastName = newLastName

    @property
    def getEmail(self):
        return self.Email

    @getEmail.setter
    def setEmail(self, newEmail):
        self.Email = newEmail
```

```python
    @property
    def getPhoneNumber(self):
        return self.PhoneNumber

    @getPhoneNumber.setter
    def setPhoneNumber(self, newPhoneNumber):
        self.PhoneNumber = newPhoneNumber

    @property
    def getUsername(self):
        return self.Username

    @getUsername.setter
    def setUsername(self, newUsername):
        self.Username = newUsername

    @property
    def getRole(self):
        return self.Role

    @getRole.setter
    def setRole(self, newRole):
        self.Role = newRole

    @property
    def getJoinDate(self):
        return self.Role

    @getJoinDate.setter
    def setJoinDate(self, newJoinDate):
        self.JoinDate = newJoinDate
```

Interfaces:

• ICustomerService:

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

```python
from abc import ABC,abstractmethod

class ICustomerService(ABC):

    @abstractmethod
    def GetCustomerById(self):
        pass

    @abstractmethod
    def GetCustomerByUsername(self):
        pass

    @abstractmethod
    def RegisterCustomer(self):
        pass

    @abstractmethod
    def UpdateCustomer(self):
        pass

    @abstractmethod
    def DeleteCustomer(self):
        pass
```

• IVehicleService:

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

```python
from abc import ABC,abstractmethod

class IVehicleService(ABC):

    @abstractmethod
    def GetVehicleById(self):
        pass

    @abstractmethod
    def GetAvailableVehicles(self):
        pass

    @abstractmethod
    def AddVehicle(self):
        pass

    @abstractmethod
    def UpdateVehicle(self):
        pass

    @abstractmethod
    def RemoveVehicle(self):
        pass
```

• IReservationService:

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

```python
from abc import ABC,abstractmethod

class IReservationService(ABC):

    @abstractmethod
    def GetReservationById(self):
        pass

    @abstractmethod
    def GetReservationsByCustomerId(self):
        pass

    @abstractmethod
    def CreateReservation(self):
        pass

    @abstractmethod
    def UpdateReservation(self):
        pass

    @abstractmethod
    def CancelReservation(self):
        pass
```

• IAdminService:

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

```python
from abc import ABC,abstractmethod

class IAdminService(ABC):

    @abstractmethod
    def GetAdminById(self):
        pass

    @abstractmethod
    def GetAdminByUsername(self):
        pass

    @abstractmethod
    def RegisterAdmin(self):
        pass

    @abstractmethod
    def UpdateAdmin(self):
        pass

    @abstractmethod
    def DeleteAdmin(self):
        pass
```

• CustomerService

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

```python
from iCustomerService import ICustomerService
import mysql.connector
from sqlconnection import SqlConnection

class SqlCommand(SqlConnection):
    def __init__(self,host,user,password,database):
        super().__init__(host,user,password,database)

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host = self.host,
                user = self.user,
                password = self.password,
                database = self.database
            )
            print("connected")
        except:
            print("could not connect to database")

```

```python
class CustomerService(ICustomerService,SqlCommand):
    def __init__(self,host,user,password,database):
        super().__init__(host,user,password,database)

    def GetCustomerById(self,Cid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from customer where customerid = %s",  params: (Cid,))
        for i in cursor:
            print(i)
        cursor.close()

    def GetCustomerByUsername(self,username):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from customer where username = %s",  params: (username,))
        for i in cursor:
            print(i)
        cursor.close()
```

```
    def RegisterCustomer(self):
        cursor = self.connection.cursor()
        FirstName = input("Enter first name ")
        LastName = input("Enter last name ")
        Email = input("Enter Email ")
        PhoneNumber = int(input("Enter phonenumber "))
        Username = input("Enter username ")
        Password = input("Enter Password ")
        RegistrationDate = input("Enter registrationDate ")

        cursor.execute( operation: "insert into customer(FirstName, LastName, Email, PhoneNumber, Username, Password,RegistrationDate)"
                        " values(%s,%s,%s,%s,%s,%s,%s)",
                        params: (FirstName, LastName, Email, PhoneNumber, Username, Password,RegistrationDate))
        self.connection.commit()
        cursor.close()

    def UpdateCustomer(self,Cid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "update customer set phonenumber = '1234567893' where customerid = %s", params: (Cid,))
        self.connection.commit()
        cursor.close()

    def DeleteCustomer(self,Cid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "delete from customer where customerid = %s", params: (Cid,))
        self.connection.commit()
        cursor.close()
```

• VehicleService

(implements IVehicleService): • Methods: GetVehicleById, GetAvailableVehicles, AddVehicle, UpdateVehicle, RemoveVehicle

```
class VehicleService(IVehicleService,SqlCommand):
    def __init__(self,host,user,password,database):
        super().__init__(host,user,password,database)

    def GetVehicleById(self,Vid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from vehicle where vehicleid = %s", params: (Vid,))
        for i in cursor:
            print(i)
        cursor.close()


    def GetAvailableVehicles(self):
        cursor = self.connection.cursor()
        cursor.execute("select * from vehicle where Availability = 1")
        for i in cursor:
            print(i)
        cursor.close()
```

```
    def AddVehicle(self):
        cursor = self.connection.cursor()
        Model = input("Enter Model ")
        Make = input("Enter Make ")
        Year = input("Enter Year")
        Color = int(input("Enter Color "))
        RegistrationNumber = input("Enter RegistrationNumber ")
        Availability = input("Enter Availability(1/0) ")
        DailyRate = input("Enter DailyRate ")

        cursor.execute( operation: "insert into admin(Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)"
                        " values(%s,%s,%s,%s,%s,%s,%s,%s)",
                        params: (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate))
        self.connection.commit()
        cursor.close()


    def UpdateVehicle(self,Vid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "update vehicle set Availability = 1 where vehicleid = %s", params: (Vid,))
        self.connection.commit()
        cursor.close()


    def RemoveVehicle(self,Vid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "delete from vehicle where vehicleid = %s", params: (Vid,))
        self.connection.commit()
        cursor.close()
```

• ReservationService

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

```
class ReservationService(IReservationService,SqlCommand):

    def GetReservationById(self,Rid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from reservation where reservationid = %s", params: (Rid,))
        for i in cursor:
            print(i)
        cursor.close()

    def GetReservationsByCustomerId(self,Cid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from reservation where customerid = %s", params: (Cid,))
        for i in cursor:
            print(i)
        cursor.close()
```

```
    def CreateReservation(self):
        cursor = self.connection.cursor()
        CustomerID = input("Enter Customerid ")
        VehicleID = input("Enter Vehicleid ")
        StartDate = input("Enter StartDate ")
        EndDate = int(input("Enter EndDate "))
        TotalCost = input("Enter Totalcost ")
        Status = input("Enter Status ")

        cursor.execute( operation: "insert into reservation(CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)"
                        " values(%s,%s,%s,%s,%s,%s)",
                        params: (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status))
        self.connection.commit()
        cursor.close()


    def UpdateReservation(self,Rid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "update reservation set totalcost = 100 where reservationid = %s", params: (Rid,))
        self.connection.commit()
        cursor.close()


    def CancelReservation(self,Rid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "update reservation set status = 'Cancelled where reservationid = %s", params: (Rid,))
        self.connection.commit()
        cursor.close()

```

• AdminService

(implements IAdminService): • Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

```
class AdminService(IAdminService,SqlCommand):
    def __init__(self,host,user,password,database):
        super().__init__(host,user,password,database)

    def GetAdminById(self,Aid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from Admin where adminid = %s", params: (Aid,))
        for i in cursor:
            print(i)
        cursor.close()

    def GetAdminByUsername(self,username):
        cursor = self.connection.cursor()
        cursor.execute( operation: "select * from Admin where username = %s", params: (username,))
        for i in cursor:
            print(i)
        cursor.close()
```

```python
    def RegisterAdmin(self):
        cursor = self.connection.cursor()
        FirstName = input("Enter first name ")
        LastName = input("Enter last name ")
        Email = input("Enter Email ")
        PhoneNumber = int(input("Enter phonenumber "))
        Username = input("Enter username ")
        Password = input("Enter Password ")
        Role = input("Enter Role ")
        JoinDate = input("Enter joinDate ")
        cursor.execute( operation: "insert into admin(FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)"
                        " values(%s,%s,%s,%s,%s,%s,%s,%s)",
                        params: (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate))
        self.connection.commit()
        cursor.close()


    def UpdateAdmin(self,Aid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "update admin set role = 'SuperAdmin' where adminid = %s", params: (Aid,))
        self.connection.commit()
        cursor.close()

    def DeleteAdmin(self,Aid):
        cursor = self.connection.cursor()
        cursor.execute( operation: "delete from admin where adminid = %s", params: (Aid,))
        self.connection.commit()
        cursor.close()
```

• DatabaseContext:

• A class responsible for handling database connections and interactions.

```python
import mysql.connector

class DatabaseConnection:
    def __init__(self,host,user,password,database):
        self.host = host
        self.user = user
        self.password= password
        self.database = database
        self.connection = None
        self.cursor = None

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            print(f"Connected to {self.database} database")
        except:
            print("Error connecting to database")

```

• ReportGenerator:

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

```python
from DatabaseContext import DatabaseConnection

class ReportGenerator(DatabaseConnection):

    def __init__(self, host, user, password, database):
        super().__init__(host,user,password,database)

    def report(self,Rid,Vid):
        cursor = self.connection.cursor()
        cursor.execute("select * from reservation where ReservationID = %S", (Rid,))
        cursor.fetchall()
        for i in cursor:
            print(i)
        cursor.close()
        cursor = self.connection.cursor()
        cursor.execute("select * from vehicle where VehicleID = %s", (Vid,))
        cursor.fetchall()
        for i in cursor:
            print(i)
        cursor.close()

```

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.

```python
class SqlConnection:
    def __init__(self,host,user,password,database):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None

```

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

```python
import mysql.connector
from sqlconnection import SqlConnection

class SqlCommand(SqlConnection):
    def __init__(self,host,user,password,database):
        super().__init__(host,user,password,database)

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host = self.host,
                user = self.user,
                password = self.password,
                database = self.database
            )
            print("connected")
        except:
            print("could not connect to database")

    def executeQuery(self,query,value):
        cursor = self.connection.cursor()
        cursor.execute(query,value)
        for i in cursor:
            print(i)
        cursor.close()
```

Custom Exceptions:

AuthenticationException:

• Thrown when there is an issue with user authentication. • Example Usage: Incorrect username or password during customer or admin login.

```python
class AuthenticationException(Exception):
    pass

def login(username,password):
    if username == "admin1" and password == "12345":
        print("Logged in Successfully")
    else:
        raise AuthenticationException
try:
    login( username: "aa",  password: "dd")

except AuthenticationException:
    print("incorrect credential")
```

ReservationException:

• Thrown when there is an issue with reservations. • Example Usage: Attempting to make a reservation for a vehicle that is already reserved.

VehicleNotFoundException:

• Thrown when a requested vehicle is not found. • Example Usage: Trying to get details of a vehicle that does not exist.

```
class VehicleNotFoundException(Exception):
    pass

def Vehicle(car):
    cars = ['Honda','Toyota','Nissan','Ford','Chevrolet']
    if car in cars:
        print("Reserved")
    else:
        raise VehicleNotFoundException

try:
    Vehicle("Honda")
except VehicleNotFoundException:
    print("Sorry that is reserved")
```

AdminNotFoundException:

• Thrown when an admin user is not found. • Example Usage: Attempting to access details of an admin that does not exist.

```
class AdminNotFoundException(Exception):
   pass

def admin(username):
    admins = ["adminOne","adminTwo","adminThree","adminFour","adminFive"]
    if username in admins:
        print(f"Showing details of {username}")
    else:
        raise AdminNotFoundException

try:
    admin("adminOne")
except AdminNotFoundException:
    print("This admin doesn't exist")
```

InvalidInputException:

• Thrown when there is invalid input data. • Example Usage: When a required field is missing or has an incorrect format.

```
class InvalidInputException(Exception):
    pass

def inputEmail(email):
    if "@" in email:
        print(f"You have entered {email}")
    else:
        raise InvalidInputException
try:
    inputEmail("Akash.kumar@gmail.com")
except InvalidInputException:
    print("Email is invalid")
```

DatabaseConnectionException:

• Thrown when there is an issue with the database connection. • Example Usage: Unable to establish a connection to the database.

```
class DatabaseConnectionException(Exception):
    pass
def databaseConnection():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="shree420",
        database="carconnect"
    )
    if conn:
            cursor = conn.cursor()
            cursor.execute("show tables")
            for i in cursor:
                print(i)
    else:
        raise DatabaseConnectionException

try:
    databaseConnection()
except DatabaseConnectionException:
    print("enable to connect")
```

Unit Testing:

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

1. Test customer authentication with invalid credentials.

```python
import pytest
import mysql.connector

conn = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "shree420",
    database = "carconnect"
)

def authentication():
    cursor = conn.cursor()
    cursor.execute("select username,password from customer where customerid = 1")
    l = []
    for i in cursor:
        l.append(i)
    return l

def test_authentication():
    assert authentication() == [('johndoe123', 'password123')]
```

2. Test updating customer information.

```python
def updatinCustomer():
    cursor = conn.cursor()
    cursor.execute("update customer set firstname = 'johnny' where customerid = 1")
    conn.commit()
    cursor.close()
    return True

def test_updatingCustomer():
    assert updatinCustomer() == True
```

3. Test adding a new vehicle.

```python
def adding():
    cursor = conn.cursor()
    cursor.execute( operation: "insert into vehicle(Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)"
                    "values(%s,%s,%s,%s,%s,%s,%s)", params: ('Malibu', 'Chevrolet', 2019, 'Red', 'BCD890',1, 55.00))
    conn.commit()
    cursor.close()
    return True

def test_adding():
    assert adding() == True
```

4. Test updating vehicle details.

```
44  def updatingVehicle():
45      cursor = conn.cursor()
46      cursor.execute("update vehicle set model = 'civi' where vehicleid = 1")
47      conn.commit()
48      cursor.close()
49      return True
50
51  def test_updatingVehicle():
52      assert updatingVehicle() == True
```

5. Test getting a list of available vehicles.

```
55  def gettingAvailableVehicle():
56      cursor = conn.cursor()
57      cursor.execute("select * from vehicle where availability = 1")
58      result = cursor.fetchall()
59      return len(result)
60
61
62  def test_gettingAvailableVehicle():
63      assert gettingAvailableVehicle() >= 0
```

6. Test getting a list of all vehicles.

```
66  def gettingAllVehicle():
67      cursor = conn.cursor()
68      cursor.execute("select * from vehicle")
69      result = cursor.fetchall()
70      return len(result)
71
72  def test_gettingAllVehicle():
73      assert gettingAllVehicle() == 10
74
```

```
✔ Tests passed: 6 of 6 tests – 11 ms

C:\Users\prash\PycharmProjects\carConnect\.venv\Scripts\python.exe "C:/Program Fil
 .2/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py" --path C:\Users\prash\
Testing started at 20:17 ...
Launching pytest with arguments C:\Users\prash\PycharmProjects\carConnect\NUnit.py
 C:\Users\prash\PycharmProjects\carConnect

============================= test session starts ==============================
collecting ... collected 6 items

NUnit.py::test_authentication PASSED                                     [ 16%]
NUnit.py::test_updatingCustomer PASSED                                   [ 33%]
NUnit.py::test_adding PASSED                                             [ 50%]
NUnit.py::test_updatingVehicle PASSED                                    [ 66%]
NUnit.py::test_gettingAvailableVehicle PASSED                            [ 83%]
NUnit.py::test_gettingAllVehicle PASSED                                  [100%]

============================== 6 passed in 0.21s ===============================

Process finished with exit code 0
```

Main.py

```python
from exception.CustomExceptions import DatabaseConnectionException
from dao.CustomerService import CustomerService
from dao.VehicleService import VehicleService
from dao.ReservationService import ReservationService

def MainMenuCustomer():
    print("1. Get customer by customerID")
    print("2. Register Customer")
    print("3. update customer")
    print("4. Delete Customer")
    print("0. Exit from Customer")
def MainMenuVehicle():
    print("1. Get vehicle by VehicleID")
    print("2. Get available Vehicle ")
    print("3. Add Vehicle")
    print("4. Update Vehicle")
    print("5. Remove Vehicle")
    print("0. Exit from Vehicle")
def MainMenuReservation():
    print("1. Get Reservation by ReservationID ")
    print("2. create Registration ")
    print("3. update Registration ")
    print("4. Cancel Registration")
    print("0. Exit from Registration ")
```

```python
def main():
    while True:
        print("---Welcome to CarConnect---")
        print("Choose Categories")
        print("1. Customer")
        print("2. Vehicle")
        print("3. Registration")
        print("9. to exit")
        option =int(input("Enter Category "))
        if option == 1:
            cus = CustomerService( host: "localhost", user: "root", password: "shree420", database: "carconnect")
            try:
                cus.connect()
            except DatabaseConnectionException:
                print("error")
            while True:
                print("choose the functionality")
                MainMenuCustomer()
                option = int(input("Enter the option number which are give above "))
                if option == 1:
                    cus.GetCustomerById()
                elif option == 2:
                    cus.RegisterCustomer()
                elif option == 3:
                    cus.UpdateCustomer()
                elif option == 4:
                    cus.DeleteCustomer()
                elif option == 0:
                    cus.disconnect()
                    break
```

```python

    if option == 2:
        veh = VehicleService( host: "localhost", user: "root", password: "shree420", database: "carconnect")
        veh.connect()
        while True:
            print("choose the functionality")
            MainMenuVehicle()
            option = int(input("Enter the option number which are give above "))
            if option == 1:
                veh.GetVehicleById()
            elif option == 2:
                veh.GetAvailableVehicles()
            elif option == 3:
                veh.AddVehicle()
            elif option == 4:
                veh.UpdateVehicle()
            elif option == 5:
                veh.RemoveVehicle()
            elif option == 0:
                break

```

```
77
78          if option == 3:
79              res = ReservationService( host: "localhost", user: "root", password: "shree420", database: "carconnect")
80              res.connect()
81              while True:
82                  print("choose the functionality")
83                  MainMenuReservation()
84                  option = int(input("Enter the option number which are given above "))
85                  if option == 1:
86                      res.GetReservationById()
87                  elif option == 2:
88                      res.CreateReservation()
89                  elif option == 3:
90                      res.UpdateReservation()
91                  elif option == 4:
92                      res.CancelReservation()
93                  elif option == 0:
94                      break
95          if option == 9:
96              print("Logging Out......")
97              break
98
99  if __name__ == "__main__":
100     main()
```

Output:

Customer ->

```
main ×
⋮
C:\Users\prash\PycharmProjects\carConnect\.venv\Scripts\python.exe C:\Users\prash\PycharmProjects\carConnect\main\main.py
---Welcome to CarConnect---
Choose Categories
1. Customer
2. Vehicle
3. Registration
9. to exit
Enter Category 1
connected
choose the functionality
1. Get customer by customerID
2. Register Customer
3. update customer
4. Delete Customer
0. Exit from Customer
Enter the option number which are give above 1
Enter CustomerID 5
(5, 'David', 'Wilson', 'david.wilson@example.com', '444-444-4444', '654 Maple St, Townsville, USA', 'davidwilson', 'david123', datetime.date(2023, 5, 12))
choose the functionality
1. Get customer by customerID
2. Register Customer
3. update customer
4. Delete Customer
0. Exit from Customer
Enter the option number which are give above
```

```
choose the functionality
1. Get customer by customerID
2. Register Customer
3. update customer
4. Delete Customer
0. Exit from Customer
Enter the option number which are given above 2
Enter first name Rahul
Enter last name Kumar
Enter Email rahul.kumar@gmail.com
Enter phonenumber 12345432
Enter username rahul123
Enter Password rahul@123
Enter registrationDate 2024-02-08
choose the functionality
```

In customer's table

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	Email	PhoneNumber	Address	Username	Password	RegistrationDate
1	johnny	Doe	john.doe@example.com	1234567893	123 Main St, Anytown, USA	johndoe123	password123	2023-01-15
2	Jane	Smith	jane.smith@example.com	987-654-3210	456 Elm St, Othertown, USA	janesmith456	letmein456	2023-02-20
3	Michael	Johnson	michael.johnson@example.com	555-555-5555	789 Oak St, Anycity, USA	michaelj88	securepassword	2023-03-10
4	Emily	Brown	emily.brown@example.com	111-222-3333	321 Pine St, Newville, USA	emilyb	p@ssw0rd	2023-04-05
5	David	Wilson	david.wilson@example.com	444-444-4444	654 Maple St, Townsville, USA	davidwilson	david123	2023-05-12
6	Sarah	Martinez	sarah.martinez@example.com	999-999-9999	987 Cedar St, Smalltown, USA	sarahm	ilovesarah	2023-06-20
7	Christopher	Anderson	chris.anderson@example.com	777-777-7777	852 Walnut St, Bigcity, USA	canderson	chrisA123	2023-07-18
8	Jessica	Garcia	jessica.garcia@example.com	333-333-3333	369 Cherry St, Metropolis, USA	jgarcia	jessgarc	2023-08-30
9	Matthew	Taylor	matthew.taylor@example.com	666-666-6666	741 Birch St, Uptown, USA	mattt	taylormade	2023-09-10
10	Amanda	Hernandez	amanda.hernandez@example.com	888-888-8888	147 Ivy St, Downtown, USA	amandah	amandapass	2023-10-25
12	Rahul	Kumar	rahul.kumar@gmail.com	12345432	NULL	rahul123	rahul@123	2024-02-08
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Vehicle ->

```
Enter Category 2
connected
choose the functionality
1. Get vehicle by VehicleID
2. Get available Vehicle
3. Add Vehicle
4. Update Vehicle
5. Remove Vehicle
0. Exit from Vehicle
Enter the option number which are given above 1
Enter VehicleID 4
(4, 'Corolla', 'Toyota', 2018, 'Silver', 'JKL012', 1, Decimal('45'))
```

```
Enter the option number which are given above 3
Enter Model civic
Enter Make honda
Enter Year 2023
Enter Color red
Enter RegistrationNumber djfde
Enter Availability(1/0) 1
Enter DailyRate 60
```

In vehicle's table

VehicleID	Model	Make	Year	Color	RegistrationNumber	Availability	DailyRate
1	civi	Honda	2020	Blue	ABC123	1	50
2	Accord	Honda	2019	Red	DEF456	1	60
3	Camry	Toyota	2021	Black	GHI789	1	55
4	Corolla	Toyota	2018	Silver	JKL012	1	45
5	Altima	Nissan	2020	White	MNO345	1	55
6	Sentra	Nissan	2019	Gray	PQR678	1	50
7	Fusion	Ford	2021	Green	STU901	1	65
8	Focus	Ford	2017	Yellow	VWX234	1	40
9	Impala	Chevrolet	2020	Brown	YZA567	1	60
10	Malibu	Chevrolet	2019	Orange	BCD890	1	55
20	Malibu	Chevrolet	2019	Red	BCD892	1	55
22	Malibu	Chevrolet	2019	Red	BCD84	1	55
23	Malibu	Chevrolet	2019	Red	BCC84	1	55
24	civic	honda	2023	red	djfde	1	60
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Reservation ->

```
Enter Category 3
connected
choose the functionality
1. Get Reservation by ReservationID
2. create Registration
3. update Registration
4. Cancel Registration
0. Exit from Registration
Enter the option number which are given above 1
Enter ReservationID 5
(5, 5, 9, datetime.datetime(2024, 2, 18, 13, 0), datetime.datetime(2024, 2, 21, 14, 0), Decimal('180'), 'pending')
```

```
Enter the option number which are given above 2
Enter Customerid 2
Enter Vehicleid 2
Enter StartDate 2024-02-08
Enter EndDate 2024-02-09
Enter Totalcost 300
Enter Status Confirmed
```

In Reservation's Table

ReservationID	CustomerID	VehicleID	StartDate	EndDate	TotalCost	Status
1	1	1	2024-02-10 09:00:00	2024-02-15 18:00:00	250	confirmed
2	2	3	2024-02-12 10:00:00	2024-02-18 15:00:00	330	confirmed
3	3	5	2024-02-14 11:00:00	2024-02-16 17:00:00	110	completed
4	4	7	2024-02-15 12:00:00	2024-02-20 12:00:00	390	confirmed
5	5	9	2024-02-18 13:00:00	2024-02-21 14:00:00	180	pending
6	6	2	2024-02-20 14:00:00	2024-02-22 16:00:00	120	pending
7	7	4	2024-02-22 15:00:00	2024-02-25 12:00:00	135	confirmed
8	8	6	2024-02-24 16:00:00	2024-02-27 10:00:00	200	pending
9	9	8	2024-02-25 08:00:00	2024-02-28 18:00:00	195	pending
10	10	10	2024-02-28 09:00:00	2024-03-02 09:00:00	120	confirmed
13	2	2	2024-02-08 00:00:00	2024-02-09 00:00:00	300	confirmed
NULL	NULL	NULL	NULL	NULL	NULL	NULL