

CASE STUDY PYTHON

NAME: - SUSHANT KUMAR SINGH

PROJECT:- CarConnect, a Car Rental Platform

Main Menu

```
Run main x Admin x
C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\carconnect\main.py
Choose an option:
1. Register Customer
2. Add Vehicle
3. Insert Reservation
4. Register Admin
5. Exit
Enter your choice (1-5): 2
```

Input/Output given by User for customer table

```
Project ▾          main.py AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py Customer.py Custom...  
Run main x  
...  
C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\carconnect\main.py  
Choose an option:  
1. Register Customer  
2. Add Vehicle  
3. Insert Reservation  
4. Register Admin  
5. Exit  
Enter your choice (1-5): 1  
  
Customer Options:  
1. Register New Customer  
2. Update Customer  
3. Delete Customer  
4. Go Back  
Enter your choice (1-4): 1  
Enter Customer ID: 6  
Enter First Name: Vijay  
Enter Last Name: T  
Enter Email: vijaythalai@gmail.com  
Enter Phone Number: 95462354  
Enter Address: Old MG Road  
Enter Username: Vijay383  
Enter Password: vijay47  
Enter Registration Date (YYYY-MM-DD): 2024-02-16  
Customer data saved to database.
```

Entered Input successfully saved into our Database

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.

```
mysql> desc customers;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra       |
+-----+-----+-----+-----+-----+
| CustomerID | int    | NO   | PRI  | NULL    | auto_increment |
| FirstName   | varchar(255) | YES  |      | NULL    |              |
| LastName    | varchar(255) | YES  |      | NULL    |              |
| Email        | varchar(255) | YES  |      | NULL    |              |
| PhoneNumber | varchar(15)  | YES  |      | NULL    |              |
| Address     | varchar(255) | YES  |      | NULL    |              |
| Username    | varchar(255) | YES  |      | NULL    |              |
| Password    | varchar(255) | YES  |      | NULL    |              |
| RegistrationDate | date   | YES  |      | NULL    |              |
+-----+-----+-----+-----+-----+
9 rows in set (0.02 sec)
```

```
CREATE TABLE IF NOT EXISTS customers (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Email VARCHAR(255),
    PhoneNumber VARCHAR(15),
    Address VARCHAR(255),
    Username VARCHAR(255),
    Password VARCHAR(255),
    RegistrationDate DATE
);
```

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.

```
mysql> desc vehicles;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra       |
+-----+-----+-----+-----+-----+
| VehicleID | int    | NO   | PRI  | NULL    | auto_increment |
| Model      | varchar(255) | YES  |      | NULL    |              |
| Make       | varchar(255) | YES  |      | NULL    |              |
| Year        | int    | YES  |      | NULL    |              |
| Color      | varchar(255) | YES  |      | NULL    |              |
| RegistrationNumber | varchar(255) | YES  |      | NULL    |              |
| Availability | varchar(255) | YES  |      | NULL    |              |
| DailyRate   | float   | YES  |      | NULL    |              |
+-----+-----+-----+-----+-----+
8 rows in set (0.01 sec)
```

```
CREATE TABLE IF NOT EXISTS vehicles (
    VehicleID INT AUTO_INCREMENT PRIMARY KEY,
    Model VARCHAR(255),
    Make VARCHAR(255),
    Year INT,
    Color VARCHAR(255),
    RegistrationNumber VARCHAR(255),
    Availability VARCHAR(255),
    DailyRate FLOAT
);
```

3. Reservation Table:

```
mysql> desc reservations;
```

Field	Type	Null	Key	Default	Extra
ReservationID	int	NO	PRI	NULL	auto_increment
CustomerID	int	YES	MUL	NULL	
VehicleID	int	YES	MUL	NULL	
StartDate	date	YES		NULL	
EndDate	date	YES		NULL	
TotalCost	float	YES		NULL	
Status	varchar(255)	YES		NULL	

7 rows in set (0.00 sec)

```
CREATE TABLE IF NOT EXISTS reservations (
    ReservationID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    VehicleID INT,
    StartDate DATE,
    EndDate DATE,
    TotalCost FLOAT,
    Status VARCHAR(255),
    FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID),
    FOREIGN KEY (VehicleID) REFERENCES vehicles(VehicleID)
);
```

4) Admin Table

```
mysql> desc admins;
```

Field	Type	Null	Key	Default	Extra
AdminID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(255)	YES		NULL	
LastName	varchar(255)	YES		NULL	
Email	varchar(255)	YES		NULL	
PhoneNumber	varchar(15)	YES		NULL	
Username	varchar(255)	YES		NULL	
Password	varchar(255)	YES		NULL	
Role	varchar(255)	YES		NULL	
JoinDate	date	YES		NULL	

9 rows in set (0.02 sec)

```
CREATE TABLE IF NOT EXISTS admins (
    AdminID INT PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Email VARCHAR(255),
    PhoneNumber VARCHAR(15),
    Username VARCHAR(255),
    Password VARCHAR(255),
    Role VARCHAR(255),
    JoinDate DATE
);
```

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)

```
1 AuthenticationService.py      2 AuthenticationException.py      3 main.py      4 Customer.py      5 CustomerService.py      6 test_VehicleService.py      7 
1 class Customer:
2     1 usage (1 dynamic)
3     def get_input_from_user(self):
4         self.customer_id = int(input("Enter Customer ID: "))
5         self.first_name = input("Enter First Name: ")
6         self.last_name = input("Enter Last Name: ")
7         self.email = input("Enter Email: ")
8         self.phone_number = input("Enter Phone Number: ")
9         self.address = input("Enter Address: ")
10        self.username = input("Enter Username: ")
11        self.password = input("Enter Password: ")
12        self.registration_date = input("Enter Registration Date (YYYY-MM-DD): ")
13
14    1 usage
15    def get_data(self):
16        return (
17            self.customer_id,
18            self.first_name,
19            self.last_name,
20            self.email,
21            self.phone_number,
22            self.address,
23            self.username,
24            self.password,
25            self.registration_date
26        )
27
28    1 usage (1 dynamic)
29    def save_to_database(self, connector):
30        query = "INSERT INTO customers VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
31        values = self.get_data()
32        connector.execute_query(query, values)
33        print("Customer data saved to database.")
```

Vehicle:

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

```
1 AuthenticationException.py      2 main.py      3 Customer.py      4 CustomerService.py      5 test_VehicleService.py      6 vehicleservice.py      7 Vehicle.py      8 
1 class Vehicle:
2     3 usages
3     def __init__(self):
4         self.vehicle_id = None
5         self.model = None
6         self.make = None
7         self.year = None
8         self.color = None
9         self.registration_number = None
10        self.availability = None
11        self.daily_rate = None
12
13    2 usages (1 dynamic)
14    def get_input_from_user(self):
15        self.vehicle_id = int(input("Enter Vehicle ID: "))
16        self.model = input("Enter Model: ")
17        self.make = input("Enter Make: ")
18        self.year = int(input("Enter Year: "))
19        self.color = input("Enter Color: ")
20        self.registration_number = input("Enter Registration Number: ")
21        self.availability = input("Enter Availability: ")
22        self.daily_rate = float(input("Enter Daily Rate: "))
23
24    1 usage (1 dynamic)
25    def save_to_database(self):
26        query = "INSERT INTO vehicles VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
27        values = (self.vehicle_id, self.model, self.make, self.year, self.color,
28                  self.registration_number, self.availability, self.daily_rate)
29        self.connector.execute_query(query, values)
30        print("Vehicle data saved to database.")
```

Reservation:

- Properties: ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status
- Methods: CalculateTotalCost()

```
1.py Customer.py CustomerService.py test_VehicleService.py VehicleService.py Vehicle.py Reservation.py test_CustomerService.py
1  class Reservation:
2      1 usage (1 dynamic)
3      def get_input_from_user(self):
4          self.reservation_id = int(input("Enter Reservation ID: "))
5          self.customer_id = int(input("Enter Customer ID: "))
6          self.vehicle_id = int(input("Enter Vehicle ID: "))
7          self.start_date = input("Enter Start Date (YYYY-MM-DD): ")
8          self.end_date = input("Enter End Date (YYYY-MM-DD): ")
9          self.total_cost = float(input("Enter Total Cost: "))
10         self.status = input("Enter Status: ")
11
12     1 usage
13     def get_data(self):
14         return (
15             self.reservation_id,
16             self.customer_id,
17             self.vehicle_id,
18             self.start_date,
19             self.end_date,
20             self.total_cost,
21             self.status
22         )
23
24     1 usage (1 dynamic)
25     def save_to_database(self, connector):
26         query = "INSERT INTO reservations VALUES (%s, %s, %s, %s, %s, %s, %s)"
27         values = self.get_data()
28         connector.execute_query(query, values)
29         print("Reservation data saved to database.")
```

Admin:

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

```
1.py CustomerService.py test_VehicleService.py VehicleService.py Vehicle.py Reservation.py Admin.py test_CustomerService.py
1  class Admin:
2      1 usage (1 dynamic)
3      def get_input_from_user(self):
4          self.admin_id = int(input("Enter Admin ID: "))
5          self.first_name = input("Enter First Name: ")
6          self.last_name = input("Enter Last Name: ")
7          self.email = input("Enter Email: ")
8          self.phone_number = input("Enter Phone Number: ")
9          self.username = input("Enter Username: ")
10         self.password = input("Enter Password: ")
11         self.role = input("Enter Role: ")
12         self.join_date = input("Enter Join Date (YYYY-MM-DD): ")
13
14     1 usage
15     def get_data(self):
16         return (
17             self.admin_id,
18             self.first_name,
19             self.last_name,
20             self.email,
21             self.phone_number,
22             self.username,
23             self.password,
24             self.role,
25             self.join_date
26         )
27
28     1 usage (1 dynamic)
29     def save_to_database(self, connector):
30         query = "INSERT INTO admins VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
31         values = self.get_data()
32         connector.execute_query(query, values)
33         print("Admin data saved to database.")
```

CustomerService (implements ICustomerService):

- Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer

```
CustomerService.py x test_VehicleService.py VehicleService.py Vehicle.py Reservation.py Admin.py test_CustomerService v ▲ 4 ▲ 10 ▾ ⌂
1 from ICustomerService import ICustomerService
2 from InvalidInputException import InvalidInputException
3 from DatabaseConnectionException import DatabaseConnectionException
2 usages
4 class CustomerService(ICustomerService):
5     def __init__(self, connector):
6         self.connector = connector
1 usage
7     def get_customer_by_id(self, customer_id):
8         try:
9             query = "SELECT * FROM customers WHERE CustomerID = %s"
10            result = self.connector.execute_query(query, (customer_id,), fetch_one=True)
11            if result:
12                customer_data = result
13                print(f"Customer found with ID {customer_id}: {customer_data}")
14            else:
15                print(f"No customer found with ID {customer_id}")
16        except Exception as e:
17            print(f"Error: {e}")
18
19     def get_customer_by_username(self, username):
20         try:
21             query = "SELECT * FROM customers WHERE Username = %s"
22             result = self.connector.execute_query(query, (username,), fetch_one=True)
23             if result:
24                 customer_data = result
25                 print(f"Customer found with username {username}: {customer_data}")
26             else:
27                 print(f"No customer found with username {username}")
28         except Exception as e:
29             print(f"Error: {e}")
```

```
CustomerService.py x test_VehicleService.py VehicleService.py Vehicle.py Reservation.py Admin.py test_CustomerService v ▲ 4 ▲ 10 ▾ ⌂
29     print(f"Error: {e}")
1 usage
30     def register_customer(self, customer, connector):
31         customer.get_input_from_user()
32         customer.save_to_database(connector)
1 usage (1 dynamic)
33     def save_to_database(self, customer_data, connector):
34         try:
35             connector.execute_query("""
36                 INSERT INTO customers
37                     (CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
38                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
39             """, customer_data)
40             print("Customer data saved to database.")
41         except Exception as e:
42             print(f"Error: {e}")
1 usage
43     def update_customer(self, customer_id, updated_customer):
44         try:
45             # Check if the customer exists
46             existing_customer = self.get_customer_by_id(customer_id)
47             if not existing_customer:
48                 raise InvalidInputException("Customer not found")
49
50             query = """
51                 UPDATE customers
52                 SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Password=?, RegistrationDate=?
53                 WHERE CustomerID=?
54             """
55             params = (
56                 updated_customer["FirstName"],
```

```
CustomerService.py x test_VehicleService.py VehicleService.py Vehicle.py Reservation.py Admin.py test_CustomerService v ▲ 4 ▲ 10 ▾ ⌂
56             updated_customer["LastName"],
57             updated_customer.get("Email", ""),
58             updated_customer.get("PhoneNumber", ""),
59             updated_customer.get("Address", ""),
60             updated_customer["Password"],
61             updated_customer.get("RegistrationDate", str(existing_customer.RegistrationDate)),
62             customer_id,
63         )
64         self.db_context.execute_query(query, params)
65     except InvalidInputException as iie:
66         raise iie
67     except Exception as e:
68         raise DatabaseConnectionException(str(e))
1 usage
69     def delete_customer(self, customer_id):
70         try:
71             reservations = self.connector.execute_query("SELECT * FROM reservations WHERE CustomerID = %s", (customer_id,), fetch_all=True)
72             if reservations:
73                 for reservation in reservations:
74                     reservation_id = reservation[0]
75                     self.connector.execute_query("DELETE FROM reservations WHERE ReservationID = %s", (reservation_id,))
76                     self.connector.execute_query("DELETE FROM customers WHERE CustomerID = %s", (customer_id,))
77                     print(f"Customer with ID {customer_id} deleted successfully.")
78             except Exception as e:
79                 print(f"Error: {e}")
80         print(f"Error: {e}")
```

VehicleService (implements IVehicleService):

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

```
CustomerService.py test_VehicleService.py VehicleService.py x Vehicle.py Reservation.py Admin.py test_CustomerService ... 6 1 from IVehicleService import IVehicleService 2 usages 3 class VehicleService(IVehicleService): 4     def __init__(self, connector): 5         self.connector = connector 6     def get_vehicle_by_id(self, vehicle_id): 7         try: 8             query = "SELECT * FROM vehicles WHERE VehicleID = %s" 9             result = self.connector.execute_query(query, (vehicle_id,), fetch_one=True)10             if result:11                 vehicle_data = result12                 print(f"Vehicle found with ID {vehicle_id}: {vehicle_data}")13             else:14                 print(f"No vehicle found with ID {vehicle_id}")15         except Exception as e:16             print(f"Error: {e}")17     1 usage18     def get_available_vehicles(self):19         try:20             query = "SELECT * FROM vehicles WHERE Availability = 'Available'"21             result = self.connector.execute_query(query, fetch_one=False)22             if result:23                 available_vehicles = result24                 print("Available Vehicles:")25                 for vehicle in available_vehicles:26                     print(vehicle)27             else:28                 print("No available vehicles.")29         except Exception as e:30             print(f"Error: {e}")31     1 usage
```

```
CustomerService.py test_VehicleService.py VehicleService.py x Vehicle.py Reservation.py Admin.py test_CustomerService ... 6 29 def add_vehicle(self, vehicle_data):30     try:31         self.connector.execute_query("""32             INSERT INTO vehicles33                 (VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)34             VALUES (%s, %s, %s, %s, %s, %s, %s, %s)35             """, vehicle_data)36         print("Vehicle data saved to database.")37     except Exception as e:38         print(f"Error: {e}")39     1 usage40     def update_vehicle(self, vehicle_data):41         try:42             query = """43                 UPDATE vehicles44                 SET Model = %s, Make = %s, Year = %s, Color = %s,45                 RegistrationNumber = %s, Availability = %s, DailyRate = %s46                 WHERE VehicleID = %s47             """48             self.connector.execute_query(query, vehicle_data[1:] + (vehicle_data[0],))49             print(f"Vehicle with ID {vehicle_data[0]} updated successfully.")50         except Exception as e:51             print(f"Error: {e}")52     1 usage51     def remove_vehicle(self, vehicle_id):52         try:53             query = "DELETE FROM vehicles WHERE VehicleID = %s"54             self.connector.execute_query(query, (vehicle_id,))55             print(f"Vehicle with ID {vehicle_id} removed successfully.")56         except Exception as e:57             print(f"Error: {e}")
```

ReservationService (implements IReservationService):

- Methods: GetReservationById, GetReservationsByCustomerId, CreateReservation, UpdateReservation, CancelReservation

```
CustomerService.py test_VehicleService.py VehicleService.py ReservationService.py x Vehicle.py Reservation.py Admin..py
1 from IReservationService import IReservationService
2 usages
3 class ReservationService(IReservationService):
4     def __init__(self, connector):
5         self.connector = connector
6     def get_reservation_by_id(self, reservation_id):
7         try:
8             query = "SELECT * FROM reservations WHERE ReservationID = %s"
9             result = self.connector.execute_query(query, (reservation_id,), fetch_one=True)
10            if result:
11                reservation_data = result
12                print(f"Reservation found with ID {reservation_id}: {reservation_data}")
13            else:
14                print(f"No reservation found with ID {reservation_id}")
15        except Exception as e:
16            print(f"Error: {e}")
17    def get_reservations_by_customer_id(self, customer_id):
18        try:
19            query = "SELECT * FROM reservations WHERE CustomerID = %s"
20            result = self.connector.execute_query(query, (customer_id,), fetch_one=False)
21            if result:
22                customer_reservations = result
23                print(f"Reservations for Customer ID {customer_id}:")
24                for reservation in customer_reservations:
25                    print(reservation)
26            else:
27                print(f"No reservations found for Customer ID {customer_id}")
28        except Exception as e:
29            print(f"Error: {e}")
30    def create_reservation(self, reservation_data):
31        try:
32            self.connector.execute_query("""
33                INSERT INTO reservations
34                (ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
35                VALUES (%s, %s, %s, %s, %s, %s, %s)
36                """, reservation_data)
37            print("Reservation data saved to database.")
38        except Exception as e:
39            print(f"Error: {e}")
40    def update_reservation(self, reservation_data):
41        try:
42            query = """
43                UPDATE reservations
44                SET CustomerID = %s, VehicleID = %s, StartDate = %s, EndDate = %s,
45                TotalCost = %s, Status = %s
46                WHERE ReservationID = %s
47                """
48            self.connector.execute_query(query, reservation_data[1:] + (reservation_data[0],))
49            print(f"Reservation with ID {reservation_data[0]} updated successfully.")
50        except Exception as e:
51            print(f"Error: {e}")
52    def cancel_reservation(self, reservation_id):
53        try:
54            query = "DELETE FROM reservations WHERE ReservationID = %s"
55            self.connector.execute_query(query, (reservation_id,))
56            print(f"Reservation with ID {reservation_id} canceled successfully.")
57        except Exception as e:
58            print(f"Error: {e}")
```

```
CustomerService.py test_VehicleService.py VehicleService.py ReservationService.py x Vehicle.py Reservation.py Admin..py
29 def create_reservation(self, reservation_data):
30     try:
31         self.connector.execute_query("""
32             INSERT INTO reservations
33             (ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
34             VALUES (%s, %s, %s, %s, %s, %s, %s)
35             """, reservation_data)
36         print("Reservation data saved to database.")
37     except Exception as e:
38         print(f"Error: {e}")
39     1 usage
40 def update_reservation(self, reservation_data):
41     try:
42         query = """
43             UPDATE reservations
44             SET CustomerID = %s, VehicleID = %s, StartDate = %s, EndDate = %s,
45             TotalCost = %s, Status = %s
46             WHERE ReservationID = %s
47             """
48         self.connector.execute_query(query, reservation_data[1:] + (reservation_data[0],))
49         print(f"Reservation with ID {reservation_data[0]} updated successfully.")
50     except Exception as e:
51         print(f"Error: {e}")
52     1 usage
51 def cancel_reservation(self, reservation_id):
52     try:
53         query = "DELETE FROM reservations WHERE ReservationID = %s"
54         self.connector.execute_query(query, (reservation_id,))
55         print(f"Reservation with ID {reservation_id} canceled successfully.")
56     except Exception as e:
57         print(f"Error: {e}")
```

AdminService (implements IAdminService):

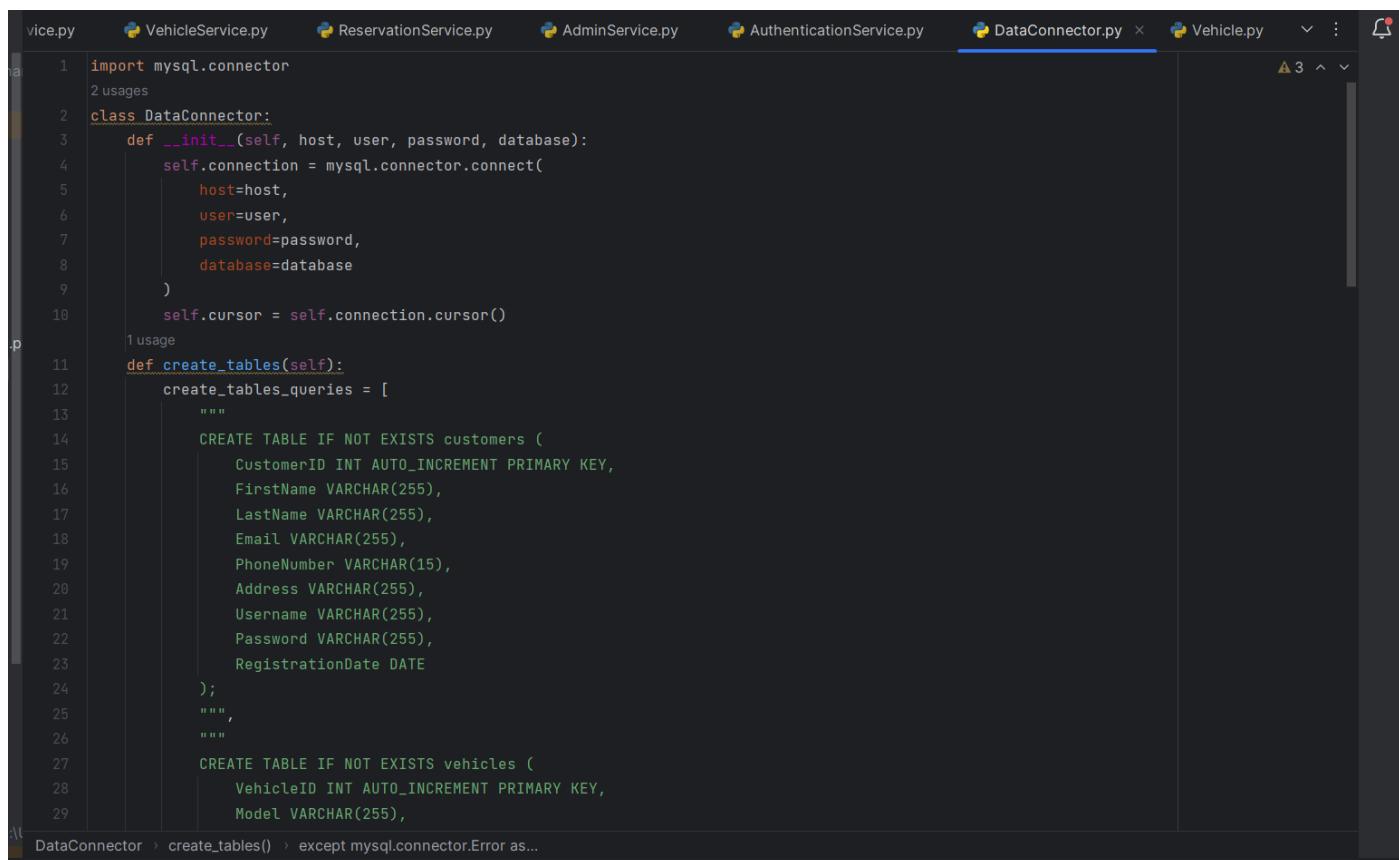
- Methods: GetAdminById, GetAdminByUsername, RegisterAdmin, UpdateAdmin, DeleteAdmin

```
CustomerService.py VehicleService.py ReservationService.py AdminService.py x Vehicle.py Reservation.py Admin.py ▾ : 🔍
1 from IAdminService import IAdminService
2 usages
3
4 class AdminService(IAdminService):
5     def __init__(self, connector):
6         self.connector = connector
7
8     def get_admin_by_id(self, admin_id):
9         try:
10             query = "SELECT * FROM admins WHERE AdminID = %s"
11             result = self.connector.execute_query(query, (admin_id,), fetch_one=True)
12             if result:
13                 admin_data = result
14                 print(f"Admin found with ID {admin_id}: {admin_data}")
15             else:
16                 print(f"No admin found with ID {admin_id}")
17         except Exception as e:
18             print(f"Error: {e}")
19
20     def get_admin_by_username(self, username):
21         try:
22             query = "SELECT * FROM admins WHERE Username = %s"
23             result = self.connector.execute_query(query, (username,), fetch_one=True)
24             if result:
25                 admin_data = result
26                 print(f"Admin found with username {username}: {admin_data}")
27             else:
28                 print(f"No admin found with username {username}")
29         except Exception as e:
30             print(f"Error: {e}")
31
32     def register_admin(self, admin_data):
33         try:
34             self.connector.execute_query("""
35                 INSERT INTO admins
36                     (AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
37                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
38                 """, admin_data)
39             print("Admin data saved to database.")
40         except Exception as e:
41             print(f"Error: {e}")
42
43     def update_admin(self, admin_data):
44         try:
45             query = """
46                 UPDATE admins
47                     SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber = %s,
48                         Username = %s, Password = %s, Role = %s, JoinDate = %
49                         WHERE AdminID = %s
50
51                 self.connector.execute_query(query, admin_data[1:] + (admin_data[0],))
52                 print(f"Admin with ID {admin_data[0]} updated successfully.")
53             except Exception as e:
54                 print(f"Error: {e}")
55
56     def delete_admin(self, admin_id):
57         try:
58             query = "DELETE FROM admins WHERE AdminID = %s"
59             self.connector.execute_query(query, (admin_id,))
60             print(f"Admin with ID {admin_id} deleted successfully.")
61         except Exception as e:
62             print(f"Error: {e}")
63
64 AdminService -> get_admin_by_id() -> try
```

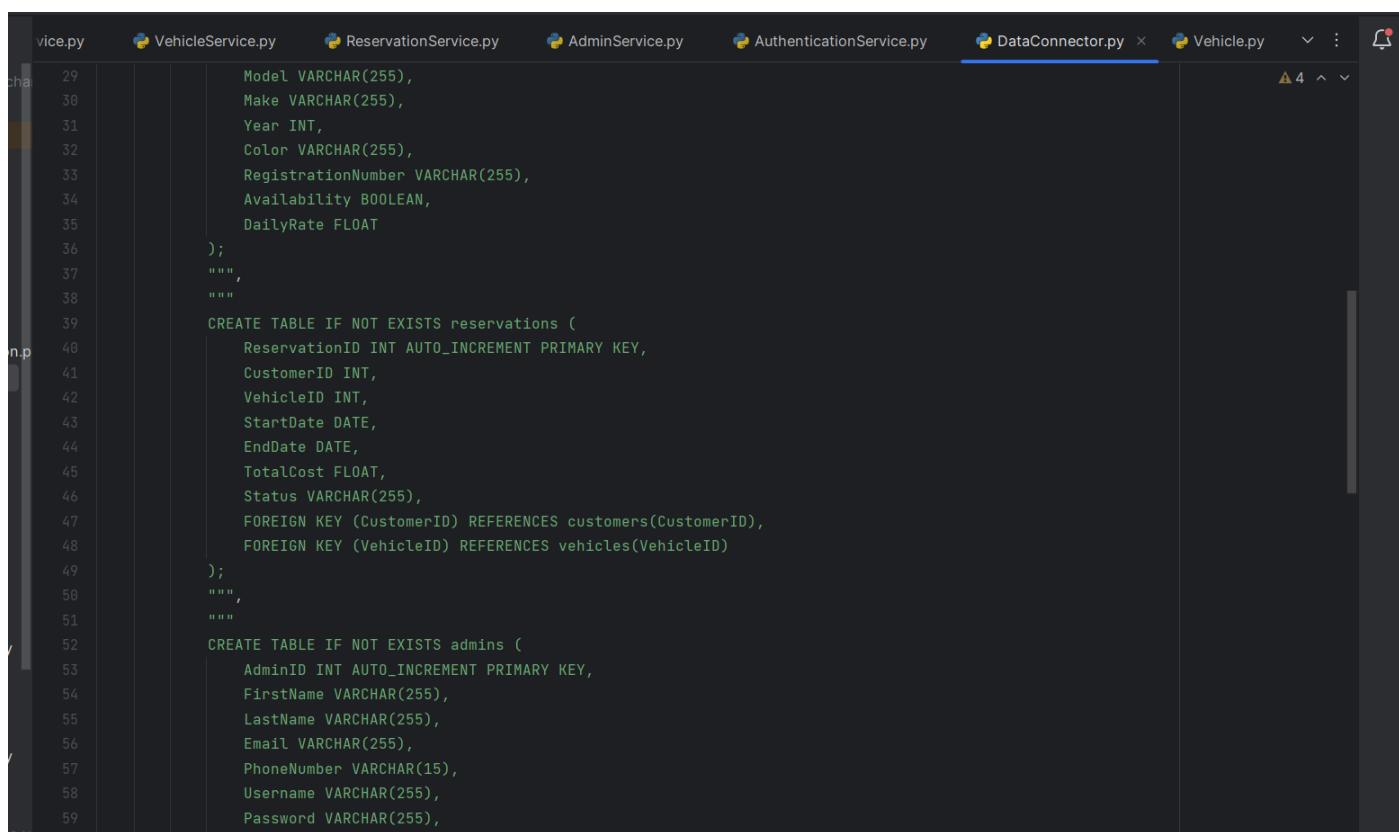
```
CustomerService.py VehicleService.py ReservationService.py AdminService.py x Vehicle.py Reservation.py Admin.py ▾ : 🔍
29
30         self.connector.execute_query("""
31             INSERT INTO admins
32                 (AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
33                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
34             """, admin_data)
35             print("Admin data saved to database.")
36         except Exception as e:
37             print(f"Error: {e}")
38
39     def update_admin(self, admin_data):
40         try:
41             query = """
42                 UPDATE admins
43                     SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber = %s,
44                         Username = %s, Password = %s, Role = %s, JoinDate = %
45                         WHERE AdminID = %s
46
47                 self.connector.execute_query(query, admin_data[1:] + (admin_data[0],))
48                 print(f"Admin with ID {admin_data[0]} updated successfully.")
49             except Exception as e:
50                 print(f"Error: {e}")
51
52     def delete_admin(self, admin_id):
53         try:
54             query = "DELETE FROM admins WHERE AdminID = %s"
55             self.connector.execute_query(query, (admin_id,))
56             print(f"Admin with ID {admin_id} deleted successfully.")
57         except Exception as e:
58             print(f"Error: {e}")
59
60 AdminService -> get_admin_by_id() -> try
```

DatabaseConnector:

- A class responsible for handling database connections and interactions.



```
1 import mysql.connector
2 usages
3
4 class DataConnector:
5     def __init__(self, host, user, password, database):
6         self.connection = mysql.connector.connect(
7             host=host,
8             user=user,
9             password=password,
10            database=database
11        )
12        self.cursor = self.connection.cursor()
13
14    1 usage
15    def create_tables(self):
16        create_tables_queries = [
17            """
18                CREATE TABLE IF NOT EXISTS customers (
19                    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
20                    FirstName VARCHAR(255),
21                    LastName VARCHAR(255),
22                    Email VARCHAR(255),
23                    PhoneNumber VARCHAR(15),
24                    Address VARCHAR(255),
25                    Username VARCHAR(255),
26                    Password VARCHAR(255),
27                    RegistrationDate DATE
28                );
29                """,
30                """
31                CREATE TABLE IF NOT EXISTS vehicles (
32                    VehicleID INT AUTO_INCREMENT PRIMARY KEY,
33                    Model VARCHAR(255),
34                    Make VARCHAR(255),
35                    Year INT,
36                    Color VARCHAR(255),
37                    RegistrationNumber VARCHAR(255),
38                    Availability BOOLEAN,
39                    DailyRate FLOAT
40                );
41                """,
42                """
43                CREATE TABLE IF NOT EXISTS reservations (
44                    ReservationID INT AUTO_INCREMENT PRIMARY KEY,
45                    CustomerID INT,
46                    VehicleID INT,
47                    StartDate DATE,
48                    EndDate DATE,
49                    TotalCost FLOAT,
50                    Status VARCHAR(255),
51                    FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID),
52                    FOREIGN KEY (VehicleID) REFERENCES vehicles(VehicleID)
53                );
54                """,
55                """
56                CREATE TABLE IF NOT EXISTS admins (
57                    AdminID INT AUTO_INCREMENT PRIMARY KEY,
58                    FirstName VARCHAR(255),
59                    LastName VARCHAR(255),
60                    Email VARCHAR(255),
61                    PhoneNumber VARCHAR(15),
62                    Username VARCHAR(255),
63                    Password VARCHAR(255),
64                    RegistrationDate DATE
65                );
66            """
67        ]
68
69        self.cursor.executemany(*create_tables_queries)
```

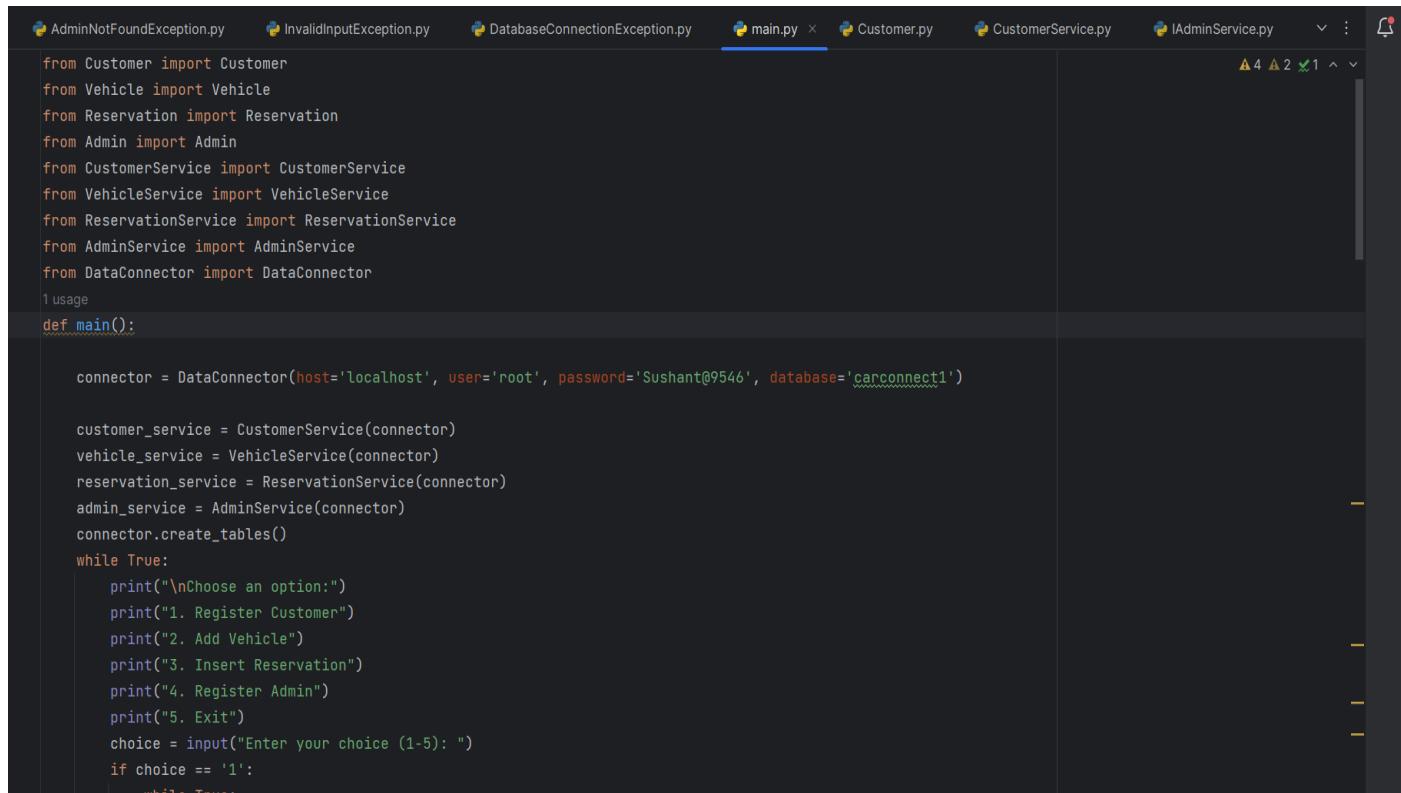


```
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

```
vice.py VehicleService.py ReservationService.py AdminService.py AuthenticationService.py DataConnector.py x Vehicle.py v : □ 4 ▲ ▼
62     );
63     """
64 ]
65     try:
66         for query in create_tables_queries:
67             self.cursor.execute(query)
68             self.connection.commit()
69     except mysql.connector.Error as err:
70         print(f"Error: {err}")
71     def execute_query(self, query, values=None):
72         try:
73             if values:
74                 self.cursor.execute(query, values)
75             else:
76                 self.cursor.execute(query)
77             self.connection.commit()
78         except mysql.connector.Error as err:
79             print(f"Error: {err}")
80     def insert_user_input_data(self, customer_data, vehicle_data, reservation_data, admin_data):
81         try:
82             # Insert customer data
83             self.cursor.execute("""
84                 INSERT INTO customers
85                     (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
86                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
87             """, customer_data)
88             self.cursor.execute("""
89                 INSERT INTO vehicles
90                     (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
91                     VALUES (%s, %s, %s, %s, %s, %s, %s)
92             """, vehicle_data)
93
94
95
96
97
98
99
100
101
102
103
104
105
106
```

```
51     try:
52         # Insert customer data
53         self.cursor.execute("""
54             INSERT INTO customers
55                 (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
56                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
57             """, customer_data)
58         self.cursor.execute("""
59             INSERT INTO vehicles
60                 (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
61                 VALUES (%s, %s, %s, %s, %s, %s, %s)
62             """, vehicle_data)
63         self.cursor.execute("""
64             INSERT INTO reservations
65                 (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
66                 VALUES (%s, %s, %s, %s, %s, %s)
67             """, reservation_data)
68         self.cursor.execute("""
69             INSERT INTO admins
70                 (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
71                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
72             """, admin_data)
73         self.connection.commit()
74     except mysql.connector.Error as err:
75         print(f"Error: {err}")
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
```

Main.py Class



```
AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py Customer.py CustomerService.py IAdminService.py ▾ ▾ 4 2 1 ▾ ▾ : 

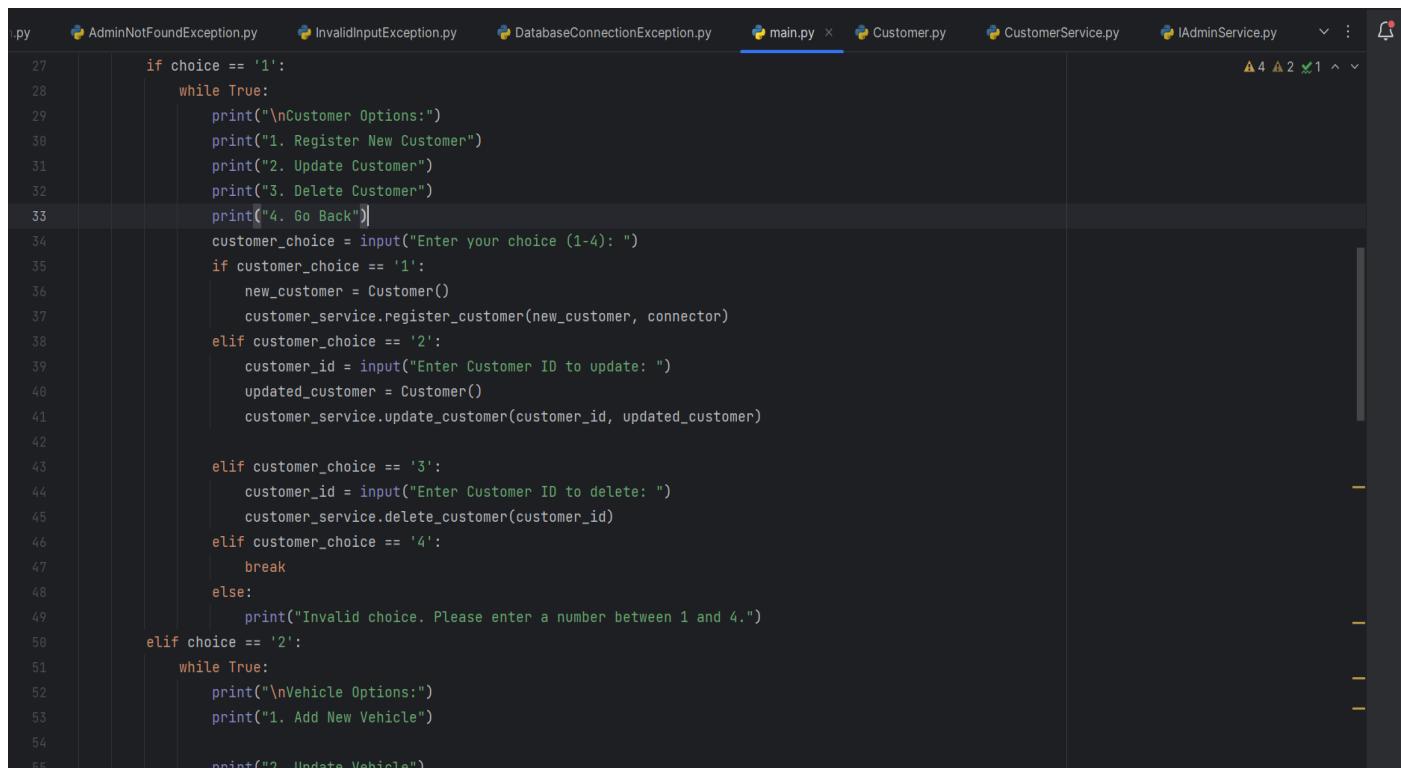
from Customer import Customer
from Vehicle import Vehicle
from Reservation import Reservation
from Admin import Admin
from CustomerService import CustomerService
from VehicleService import VehicleService
from ReservationService import ReservationService
from AdminService import AdminService
from DataConnector import DataConnector

1 usage
def main():

    connector = DataConnector(host='localhost', user='root', password='Sushant@9546', database='carconnect1')

    customer_service = CustomerService(connector)
    vehicle_service = VehicleService(connector)
    reservation_service = ReservationService(connector)
    admin_service = AdminService(connector)
    connector.create_tables()

    while True:
        print("\nChoose an option:")
        print("1. Register Customer")
        print("2. Add Vehicle")
        print("3. Insert Reservation")
        print("4. Register Admin")
        print("5. Exit")
        choice = input("Enter your choice (1-5): ")
        if choice == '1':
            while True:
```



```
if choice == '1':
    while True:
        print("\nCustomer Options:")
        print("1. Register New Customer")
        print("2. Update Customer")
        print("3. Delete Customer")
        print("4. Go Back")
        customer_choice = input("Enter your choice (1-4): ")
        if customer_choice == '1':
            new_customer = Customer()
            customer_service.register_customer(new_customer, connector)
        elif customer_choice == '2':
            customer_id = input("Enter Customer ID to update: ")
            updated_customer = Customer()
            customer_service.update_customer(customer_id, updated_customer)

        elif customer_choice == '3':
            customer_id = input("Enter Customer ID to delete: ")
            customer_service.delete_customer(customer_id)
        elif customer_choice == '4':
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
    elif choice == '2':
        while True:
            print("\nVehicle Options:")
            print("1. Add New Vehicle")
            print("2. Update Vehicle")
```

```
AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py Customer.py CustomerService.py IAdminService.py ▾ ▾ 4 2 ✘ 1 ▾ ▾
print("2. Update Vehicle")
print("3. Remove Vehicle")
print("4. Go Back")
vehicle_choice = input("Enter your choice (1-4): ")
if vehicle_choice == '1':
    new_vehicle = Vehicle()
    new_vehicle.get_input_from_user()
    vehicle_service.add_vehicle(new_vehicle.get_data_for_database())
elif vehicle_choice == '2':
    vehicle_id = input("Enter Vehicle ID to update: ")
    updated_vehicle = Vehicle()
    vehicle_service.update_vehicle(vehicle_id, updated_vehicle)
elif vehicle_choice == '3':
    vehicle_id = input("Enter Vehicle ID to remove: ")
    vehicle_service.remove_vehicle(vehicle_id)
elif vehicle_choice == '4':
    break
else:
    print("Invalid choice. Please enter a number between 1 and 4.")
elif choice == '3':
    while True:
        print("\nReservation Options:")
        print("1. Create New Reservation")
        print("2. Update Reservation")
        print("3. Cancel Reservation")
        print("4. Go Back")
        reservation_choice = input("Enter your choice (1-4): ")
        if reservation_choice == '1':
            new_reservation = Reservation()
            reservation_service.create_reservation(new_reservation)
        elif reservation_choice == '2':
            reservation_id = input("Enter Reservation ID to update: ")
            updated_reservation = Reservation()
            reservation_service.update_reservation(reservation_id, updated_reservation)
        elif reservation_choice == '3':
            reservation_id = input("Enter Reservation ID to cancel: ")
            reservation_service.cancel_reservation(reservation_id)
        elif reservation_choice == '4':
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
    elif choice == '4':
        while True:
    elif choice == '5':
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 5.")
connector.close_connection()

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

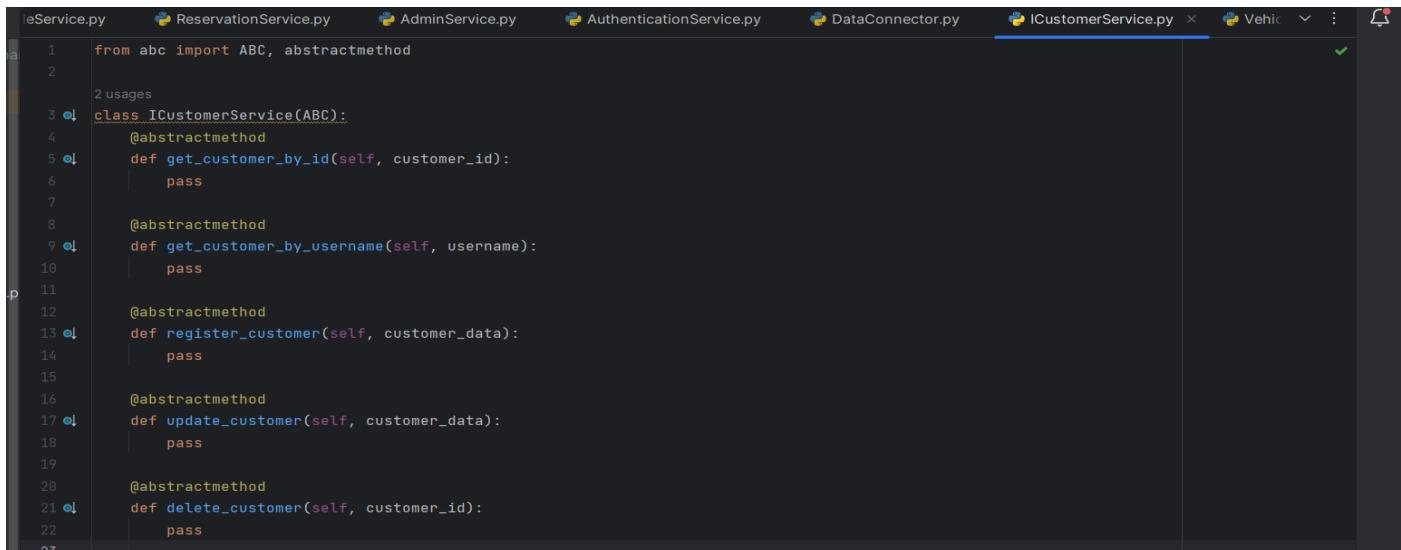
```
1.py AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py Customer.py CustomerService.py IAdminService.py ▾ ▾ 4 2 ✘ 1 ▾ ▾
78     print("2. Update Reservation")
79     print("3. Cancel Reservation")
80     print("4. Go Back")
81     reservation_choice = input("Enter your choice (1-4): ")
82     if reservation_choice == '1':
83         new_reservation = Reservation()
84         reservation_service.create_reservation(new_reservation)
85     elif reservation_choice == '2':
86         reservation_id = input("Enter Reservation ID to update: ")
87         updated_reservation = Reservation()
88         reservation_service.update_reservation(reservation_id, updated_reservation)
89     elif reservation_choice == '3':
90         reservation_id = input("Enter Reservation ID to cancel: ")
91         reservation_service.cancel_reservation(reservation_id)
92     elif reservation_choice == '4':
93         break
94     else:
95         print("Invalid choice. Please enter a number between 1 and 4.")
96     elif choice == '4':
97         while True:
118     elif choice == '5':
119         break
120     else:
121         print("Invalid choice. Please enter a number between 1 and 5.")
122 connector.close_connection()

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

Interfaces:

ICustomerService:

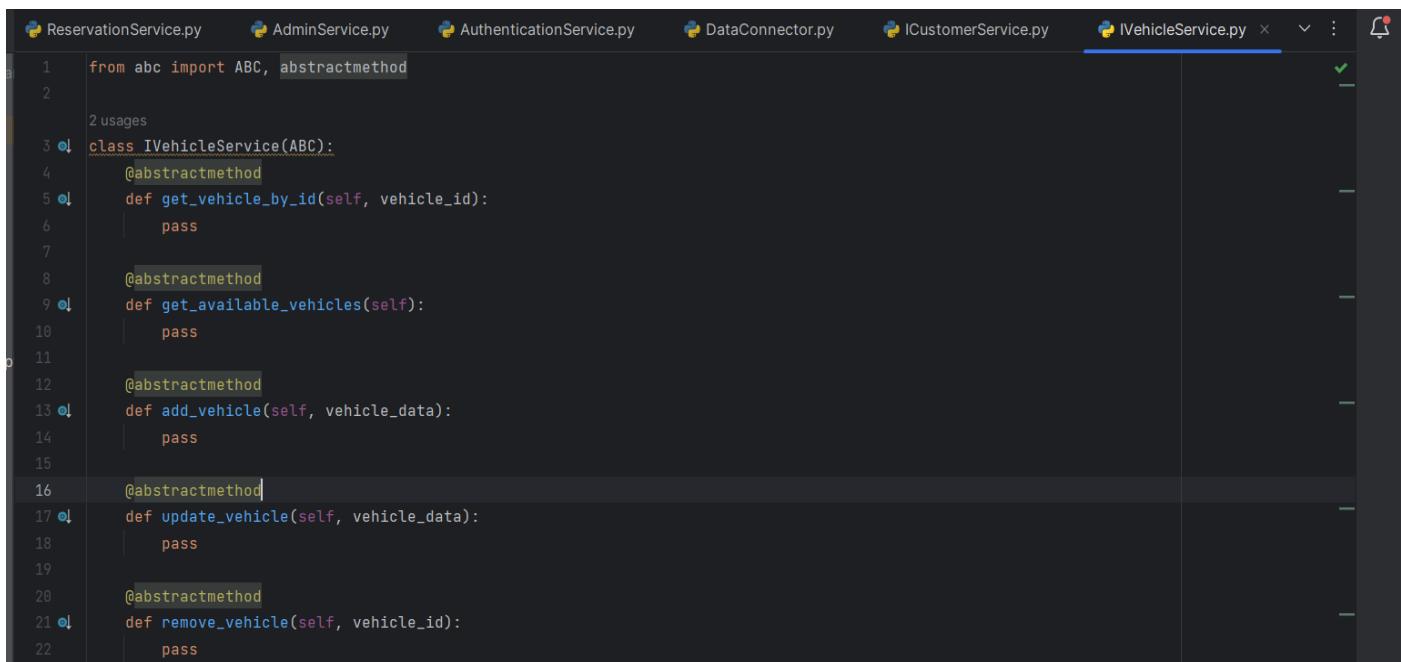
- GetCustomerById(customerId)
- GetCustomerByUsername(username)
- RegisterCustomer(customerData)
- UpdateCustomer(customerData)
- DeleteCustomer(customerId)



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

IVehicleService:

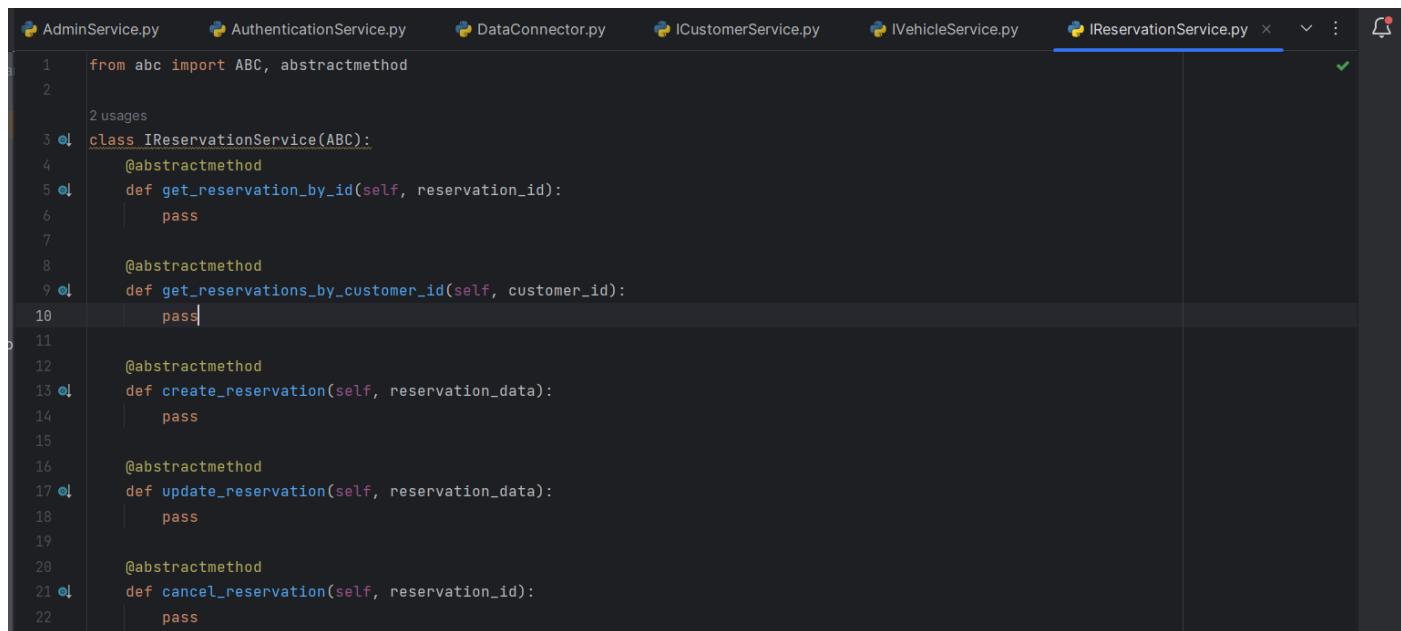
- GetVehicleById(vehicleId)
- GetAvailableVehicles()
- AddVehicle(vehicleData)
- UpdateVehicle(vehicleData)
- RemoveVehicle(vehicleId)



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

IReservationService:

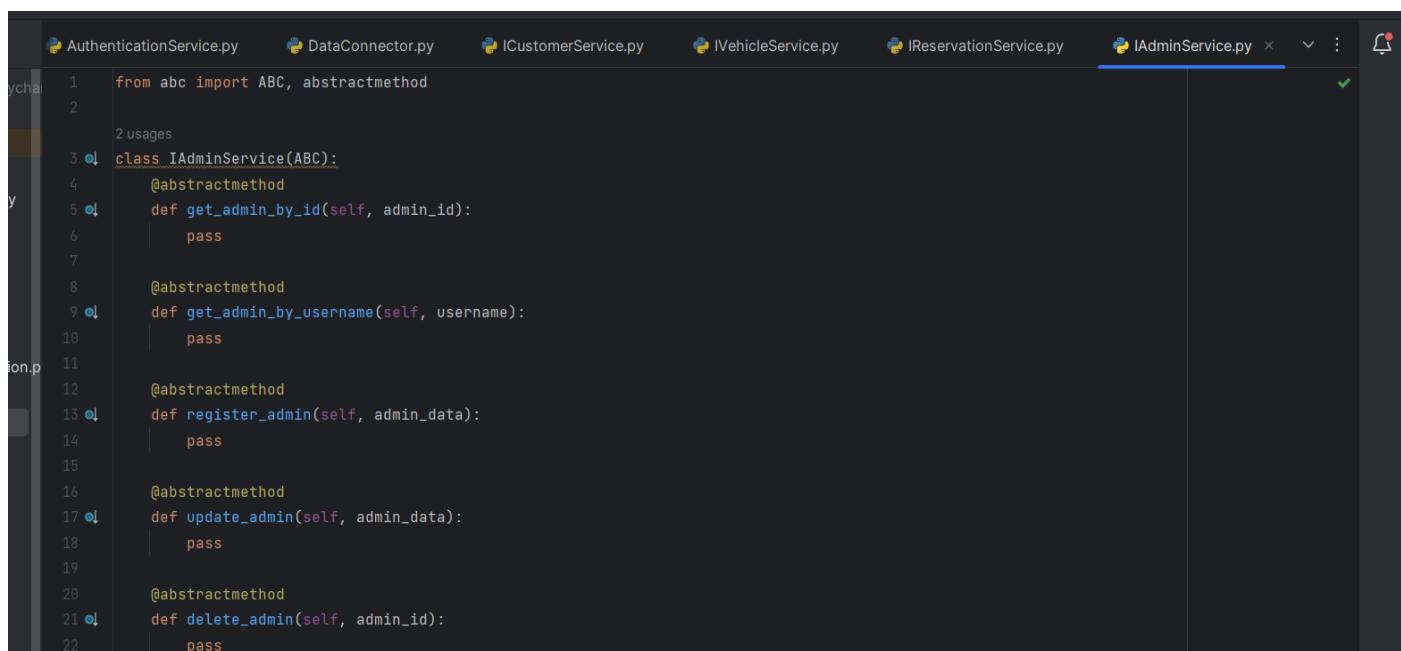
- GetReservationById(reservationId)
- GetReservationsByCustomerId(customerId)
- CreateReservation(reservationData)
- UpdateReservation(reservationData)
- CancelReservation(reservationId)



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

IAdminService:

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



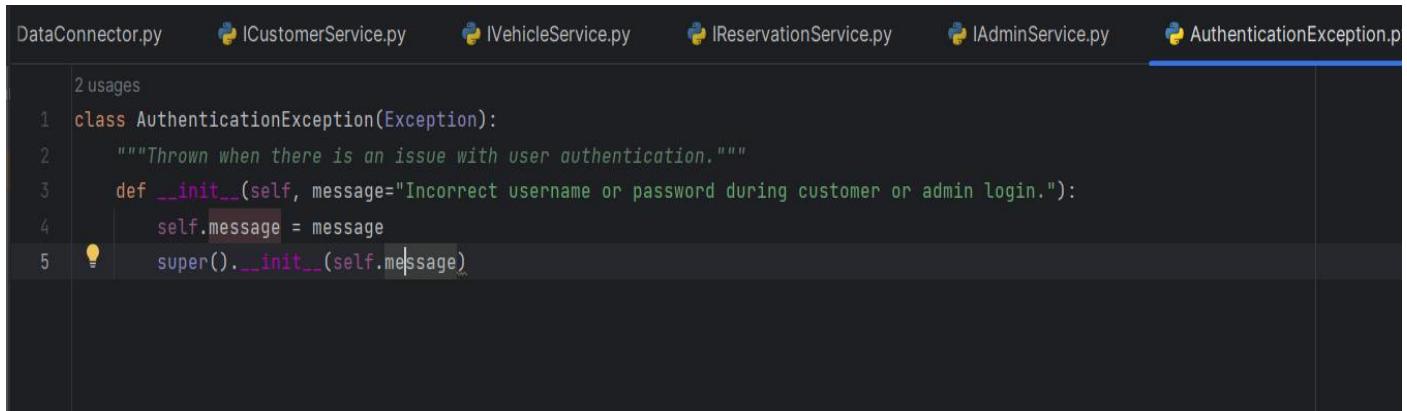
```
1 from abc import ABC, abstractmethod
2
3 class IAdminService(ABC):
4     @abstractmethod
5     def get_admin_by_id(self, admin_id):
6         pass
7
8     @abstractmethod
9     def get_admin_by_username(self, username):
10        pass
11
12     @abstractmethod
13     def register_admin(self, admin_data):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

Custom Exceptions:

Note: Each and every exceptions is connected to there respective modules in different classes.

AuthenticationException:

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

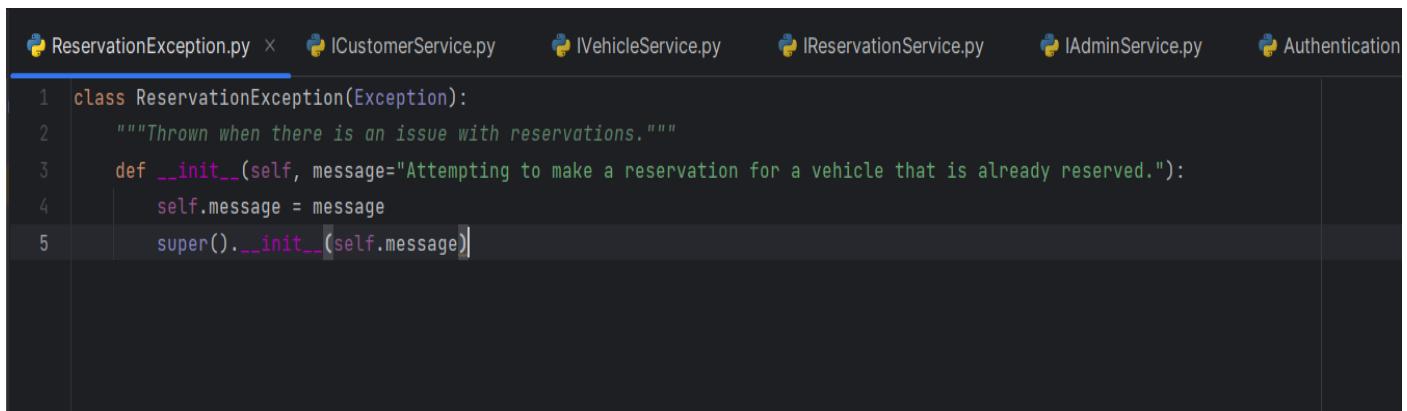


```
DataConnector.py   ICustomerService.py   IVehicleService.py   IReservationService.py   IAdminService.py   AuthenticationException.py

1  2 usages
2  class AuthenticationException(Exception):
3      """Thrown when there is an issue with user authentication."""
4      def __init__(self, message="Incorrect username or password during customer or admin login."):
5          self.message = message
6          super().__init__(self.message)
```

ReservationException:

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

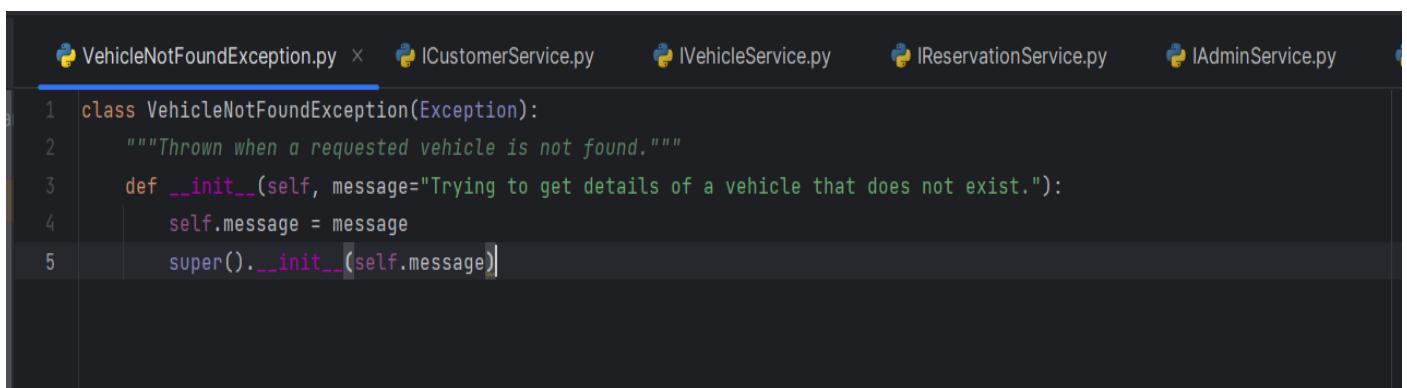


```
ReservationException.py x   ICustomerService.py   IVehicleService.py   IReservationService.py   IAdminService.py   AuthenticationException.py

1  class ReservationException(Exception):
2      """Thrown when there is an issue with reservations."""
3      def __init__(self, message="Attempting to make a reservation for a vehicle that is already reserved."):
4          self.message = message
5          super().__init__(self.message)
```

VehicleNotFoundException:

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

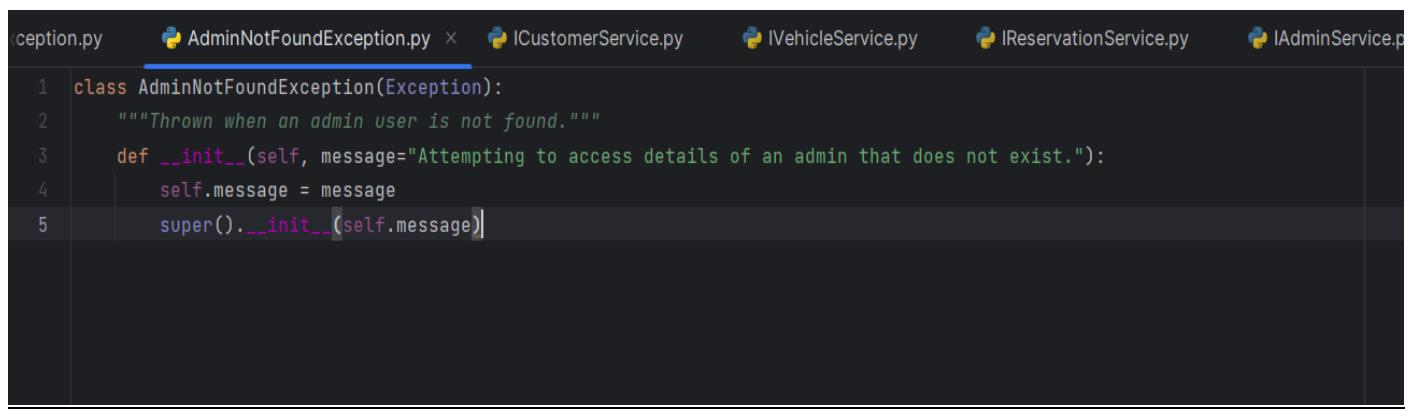


```
VehicleNotFoundException.py x   ICustomerService.py   IVehicleService.py   IReservationService.py   IAdminService.py

1  class VehicleNotFoundException(Exception):
2      """Thrown when a requested vehicle is not found."""
3      def __init__(self, message="Trying to get details of a vehicle that does not exist."):
4          self.message = message
5          super().__init__(self.message)
```

AdminNotFoundException:

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

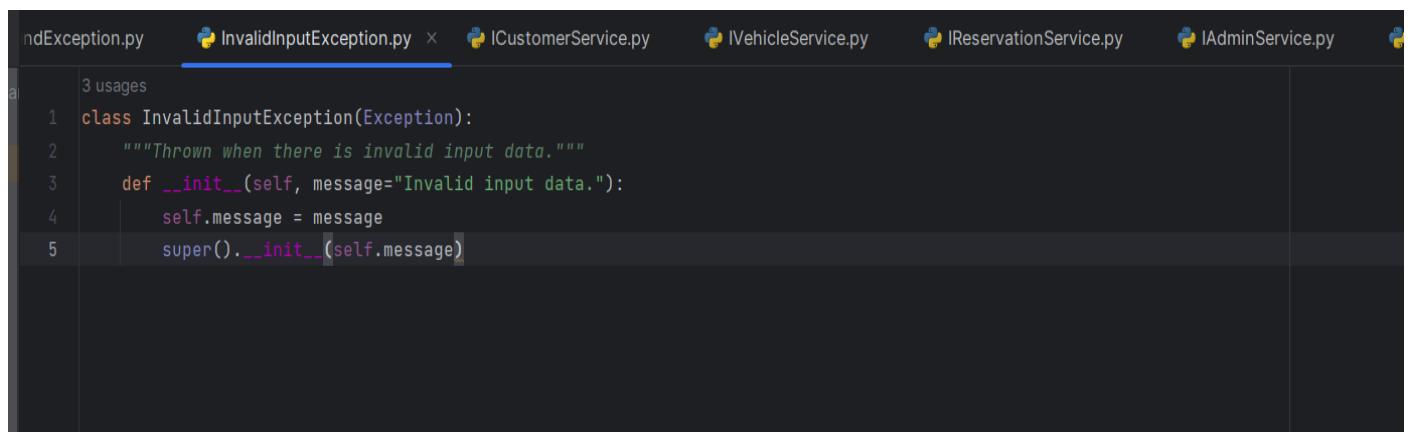


A screenshot of a code editor showing a file named AdminNotFoundException.py. The code defines a class AdminNotFoundException that inherits from Exception. It includes a docstring stating "Thrown when an admin user is not found." and an __init__ method that initializes self.message with the message "Attempting to access details of an admin that does not exist." Other files like ICustomerService.py, IVehicleService.py, IReservationService.py, and IAdminService.py are visible in the background.

```
1 class AdminNotFoundException(Exception):
2     """Thrown when an admin user is not found."""
3     def __init__(self, message="Attempting to access details of an admin that does not exist."):
4         self.message = message
5         super().__init__(self.message)
```

InvalidInputException:

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

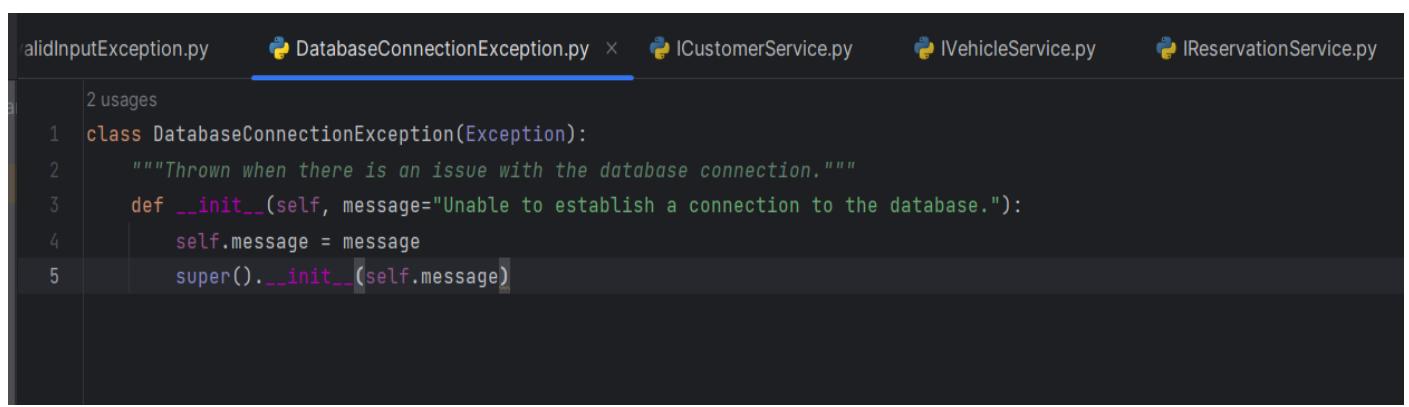


A screenshot of a code editor showing a file named InvalidInputException.py. The code defines a class InvalidInputException that inherits from Exception. It includes a docstring stating "Thrown when there is invalid input data." and an __init__ method that initializes self.message with the message "Invalid input data.". Other files like ICustomerService.py, IVehicleService.py, IReservationService.py, and IAdminService.py are visible in the background.

```
1 class InvalidInputException(Exception):
2     """Thrown when there is invalid input data."""
3     def __init__(self, message="Invalid input data."):
4         self.message = message
5         super().__init__(self.message)
```

DatabaseConnectionException:

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



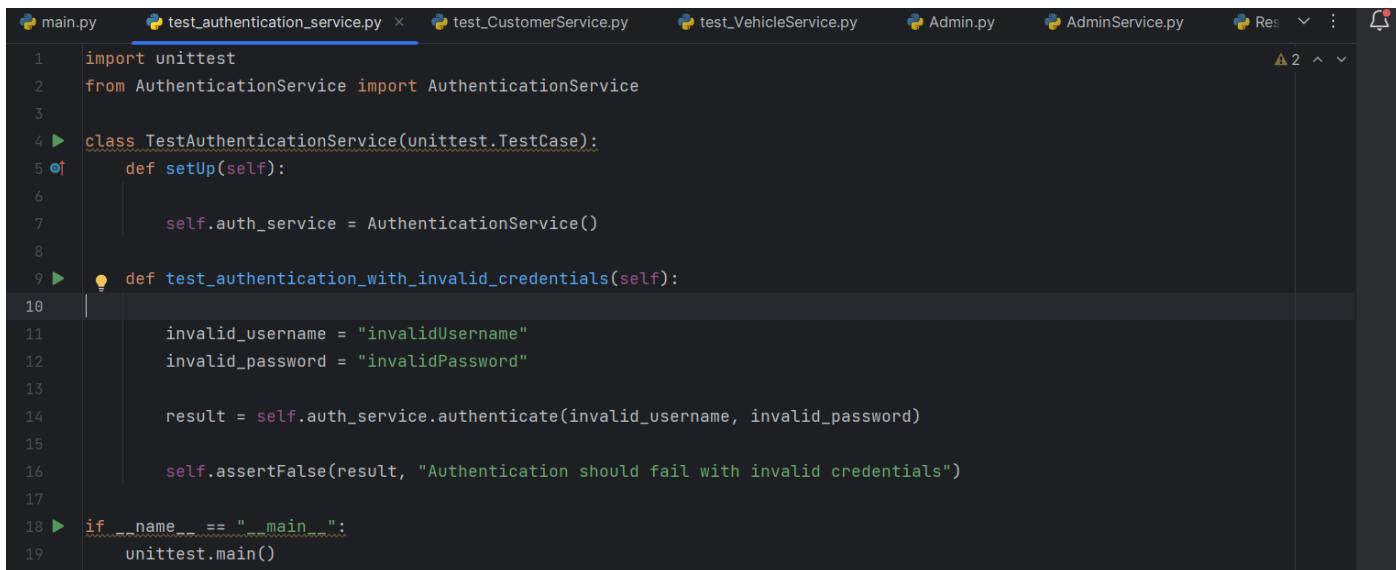
A screenshot of a code editor showing a file named DatabaseConnectionException.py. The code defines a class DatabaseConnectionException that inherits from Exception. It includes a docstring stating "Thrown when there is an issue with the database connection." and an __init__ method that initializes self.message with the message "Unable to establish a connection to the database.". Other files like ICustomerService.py, IVehicleService.py, IReservationService.py, and IAdminService.py are visible in the background.

```
1 class DatabaseConnectionException(Exception):
2     """Thrown when there is an issue with the database connection."""
3     def __init__(self, message="Unable to establish a connection to the database."):
4         self.message = message
5         super().__init__(self.message)
```

Unit Testing:

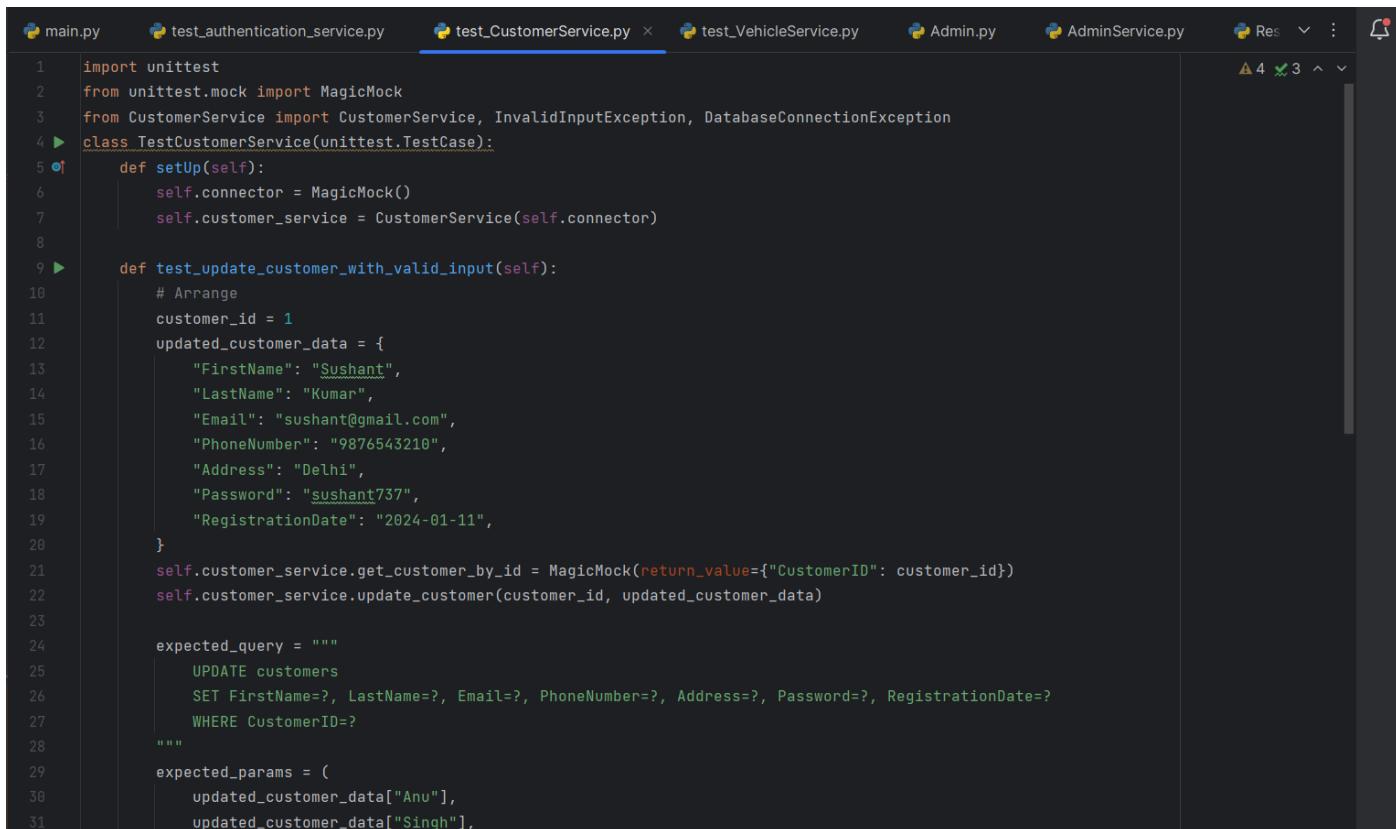
Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of NUnit test cases for various components of the system:

1. Test customer authentication with invalid credentials.



```
main.py      test_authentication_service.py ×  test_CustomerService.py      test_VehicleService.py      Admin.py      AdminService.py      Res      :      🔍
1 import unittest
2 from AuthenticationService import AuthenticationService
3
4 class TestAuthenticationService(unittest.TestCase):
5     def setUp(self):
6         self.auth_service = AuthenticationService()
7
8     def test_authentication_with_invalid_credentials(self):
9         invalid_username = "invalidUsername"
10        invalid_password = "invalidPassword"
11
12        result = self.auth_service.authenticate(invalid_username, invalid_password)
13
14        self.assertFalse(result, "Authentication should fail with invalid credentials")
15
16    if __name__ == "__main__":
17        unittest.main()
```

2. Test updating customer information.



```
main.py      test_authentication_service.py      test_CustomerService.py ×  test_VehicleService.py      Admin.py      AdminService.py      Res      :      🔍
1 import unittest
2 from unittest.mock import MagicMock
3 from CustomerService import CustomerService, InvalidInputException, DatabaseConnectionException
4 class TestCustomerService(unittest.TestCase):
5     def setUp(self):
6         self.connector = MagicMock()
7         self.customer_service = CustomerService(self.connector)
8
9     def test_update_customer_with_valid_input(self):
10        # Arrange
11        customer_id = 1
12        updated_customer_data = {
13            "FirstName": "Sushant",
14            "LastName": "Kumar",
15            "Email": "sushant@gmail.com",
16            "PhoneNumber": "9876543210",
17            "Address": "Delhi",
18            "Password": "sushant737",
19            "RegistrationDate": "2024-01-11",
20        }
21        self.customer_service.get_customer_by_id = MagicMock(return_value={"CustomerID": customer_id})
22        self.customer_service.update_customer(customer_id, updated_customer_data)
23
24        expected_query = """
25            UPDATE customers
26            SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Password=?, RegistrationDate=?
27            WHERE CustomerID=?
28        """
29        expected_params = (
30            updated_customer_data["Anu"],
31            updated_customer_data["Singh"],
```

```

31     updated_customer_data["Singh"],
32     updated_customer_data.get("anu@gmail.co", ""),
33     updated_customer_data.get("9523456", ""),
34     updated_customer_data.get("Mumbai", ""),
35     updated_customer_data.get("anu9398"),
36     updated_customer_data.get("2024-01-11", ""),
37     customer_id,
38   )
39   self.connector.execute_query.assert_called_once_with(expected_query, expected_params)
40 def test_update_customer_with_invalid_customer_id(self):
41   invalid_customer_id = 999
42   updated_customer_data = {
43     "FirstName": "UpdatedFirstName",
44     "LastName": "UpdatedLastName",
45   }
46   self.customer_service.get_customer_by_id = MagicMock(return_value=None)
47   with self.assertRaises(InvalidInputException):
48     self.customer_service.update_customer(invalid_customer_id, updated_customer_data)
49 def test_update_customer_with_database_connection_error(self):
50   customer_id = 1
51   updated_customer_data = {
52     "FirstName": "sushant",
53     "LastName": "kumar",
54   }

```

```

Terminal Local + × : - -----
self = <AuthenticationService.AuthenticationService object at 0x0000021CC04817F0>, username = 'invalidUsername', password = 'invalidPassword'

def authenticate(self, username, password):
    valid_username = "root"
    valid_password = "Sushant@9546"

    if username == valid_username and password == valid_password:
        return True
    else:
        raise AuthenticationException("Invalid username or password")
E     AuthenticationException.AuthenticationException: Invalid username or password

AuthenticationService.py:12: AuthenticationException
=====
===== short test summary info =====
FAILED test_CustomerService.py::TestCustomerService::test_update_customer_with_valid_input - DatabaseConnectionException.DatabaseConnectionException: 'dict' object has no attribute 'RegistrationDate'
FAILED test_VehicleService.py::TestVehicleService::test_get_available_vehicles - TypeError: object of type 'NoneType' has no len()
FAILED test_authentication_service.py::TestAuthenticationService::test_authentication_with_invalid_credentials - AuthenticationException.AuthenticationException: Invalid username or password
===== 3 failed, 2 passed in 0.36s =====
(venv) PS C:\Users\ssush\PycharmProjects\carconnect>

```

3. Test adding a new vehicle.

```

1 import unittest
2 from unittest.mock import MagicMock
3 from VehicleService import VehicleService
4
5 class TestVehicleService(unittest.TestCase):
6   def setUp(self):
7     self.connector = MagicMock()
8     self.vehicle_service = VehicleService(self.connector)
9
10  def test_get_available_vehicles(self):
11    self.connector.execute_query.return_value = [
12      (1, 'Sedan', 'Toyota', 2022, 'Blue', 'ABC123', 'Available', 50.0),
13      (2, 'SUV', 'Ford', 2023, 'Red', 'XYZ789', 'Available', 60.0)
14    ]
15    available_vehicles = self.vehicle_service.get_available_vehicles()
16    expected_query = "SELECT * FROM vehicles WHERE Availability = 'Available'"
17    self.connector.execute_query.assert_called_once_with(expected_query, fetch_one=False)
18
19    self.assertEqual(len(available_vehicles), 2, "Expected 2 available vehicles")
20    self.assertEqual(available_vehicles[0]["Model"], "Sedan", "Incorrect vehicle model")
21    self.assertEqual(available_vehicles[1]["DailyRate"], 60.0, "Incorrect daily rate")
22
23  if __name__ == "__main__":
24    unittest.main()
25

```

```
DatabaseConnectionException.p TestVehicleService > test_add_new_vehicle_with_database...
Terminal Local + v : -
FAILED test_VehicleService.py::TestVehicleService::test_add_new_vehicle_with_valid_input - AssertionError: expected call not found.
FAILED test_authentication_service.py::TestAuthenticationService::test_authentication_with_invalid_credentials - AuthenticationException.AuthenticationException: Invalid username or password
=====
===== 3 failed, 1 passed in 0.32s =====
(venv) PS C:\Users\ssush\PycharmProjects\carconnect> pytest -v
=====
===== test session starts =====
platform win32 -- Python 3.9.13, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\ssush\PycharmProjects\carconnect
collected 4 items

test_VehicleService.py::TestVehicleService::test_add_new_vehicle_with_database_connection_error PASSED [ 25%]
test_VehicleService.py::TestVehicleService::test_add_new_vehicle_with_invalid_input FAILED [ 50%]
test_VehicleService.py::TestVehicleService::test_add_new_vehicle_with_valid_input FAILED [ 75%]
test_authentication_service.py::TestAuthenticationService::test_authentication_with_invalid_credentials FAILED [100%]

=====
===== FAILURES =====
-----
TestVehicleService.test_add_new_vehicle_with_invalid_input
-----
self = <test_VehicleService.TestVehicleService testMethod=test_add_new_vehicle_with_invalid_input>
```

4. Test updating vehicle details. 5. Test getting a list of available vehicles. 6. Test getting a list of all vehicles.

The screenshot shows a code editor with several tabs at the top: main.py, test_authentication_service.py, test_CustomerService.py, test_VehicleService.py, test_update_vehicle_details.py (the active tab), and Admin.py. The code in the editor is a Python unittest for a VehicleService. It includes a setUp method and a test_update_vehicle_with_valid_input method. The test uses a MagicMock for the database connection and vehicle service. It constructs an updated vehicle data dictionary with various fields like Model, Make, Year, Color, RegistrationNumber, Availability, and DailyRate. It then calls update_vehicle on the service with the vehicle ID and the updated data, asserting that the expected SQL UPDATE query was generated with the correct parameters.

```
1 import unittest
2 from unittest.mock import MagicMock
3 from VehicleService import VehicleService
4 from InvalidInputException import InvalidInputException
5 from DatabaseConnectionException import DatabaseConnectionException
6 class TestVehicleService(unittest.TestCase):
7     def setUp(self):
8         self.connector = MagicMock()
9         self.vehicle_service = VehicleService(self.connector)
10
11    def test_update_vehicle_with_valid_input(self):
12        # Arrange
13        vehicle_id = 1
14        updated_vehicle_data = {
15            "Model": "UpdatedModel",
16            "Make": "UpdatedMake",
17            "Year": 2023,
18            "Color": "UpdatedColor",
19            "RegistrationNumber": "UpdatedRegNumber",
20            "Availability": "UpdatedAvailability",
21            "DailyRate": 60.0,
22        }
23        self.vehicle_service.get_vehicle_by_id = MagicMock(return_value={"VehicleID": vehicle_id})
24        self.vehicle_service.update_vehicle(vehicle_id, updated_vehicle_data)
25        expected_query = """
26            UPDATE vehicles
27            SET Model=? , Make=? , Year=? , Color=? , RegistrationNumber=? , Availability=? , DailyRate=?
28            WHERE VehicleID=?
29        """
30        expected_params = (
31            updated_vehicle_data["SUV"],
```

The screenshot shows a code editor with several tabs open at the top, including 'main.py', 'test_authentication_service.py', 'test_CustomerService.py', 'test_VehicleService.py', 'test_update_vehicle_details.py' (which is the active tab), and 'Admin.py'. The code in the editor is a test suite for a vehicle service. It includes fixtures for a database connection and vehicle service, and uses MagicMock to simulate database interactions. The tests cover updating a vehicle with valid and invalid IDs, as well as handling database connection errors.

```
31     updated_vehicle_data["SUV"],
32     updated_vehicle_data["RangeRover"],
33     updated_vehicle_data["2023"],
34     updated_vehicle_data["Black"],
35     updated_vehicle_data["DL0125"],
36     updated_vehicle_data["Available"],
37     updated_vehicle_data["2000"],
38     vehicle_id["2"],
39 )
40 self.connector.execute_query.assert_called_once_with(expected_query, expected_params)
41 def test_update_vehicle_with_invalid_vehicle_id(self):
42     invalid_vehicle_id = 999
43     updated_vehicle_data = {
44         "Model": "SUV",
45     }
46     self.vehicle_service.get_vehicle_by_id = MagicMock(return_value=None)
47     with self.assertRaises(InvalidInputException):
48         self.vehicle_service.update_vehicle(invalid_vehicle_id, updated_vehicle_data)
49
50 def test_update_vehicle_with_database_connection_error(self):
51     vehicle_id = 1
52     updated_vehicle_data = {
53         "Model": "Sedan",
54     }
55     self.vehicle_service.get_vehicle_by_id = MagicMock(return_value={"VehicleID": vehicle_id})
56     self.connector.execute_query.side_effect = DatabaseConnectionException("Database connection error")
57     with self.assertRaises(DatabaseConnectionException):
58         self.vehicle_service.update_vehicle(vehicle_id, updated_vehicle_data)
59
60 if __name__ == "__main__":
61     unittest.main()
```

OUTPUTS

```
C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\carconnect\main.py

Choose an option:
1. Register Customer
2. Add Vehicle
3. Insert Reservation
4. Register Admin
5. Exit
Enter your choice (1-5): 2

Vehicle Options:
1. Add New Vehicle
2. Update Vehicle
3. Remove Vehicle
4. Go Back
Enter your choice (1-4): 1
Enter Vehicle ID: 5
Enter Model: Truck
Enter Make: Tesla
Enter Year: 2023
Enter Color: Silver
Enter Registration Number: DL012545
Enter Availability: Available
Enter Daily Rate: 7000
Vehicle data saved to database.
```

DATA SUCCESSFULLY SAVED INTO OUR DATABASE

```
mysql> select * from vehicles;
+-----+-----+-----+-----+-----+-----+-----+-----+
| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability | DailyRate |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Sedan | Toyota | 2022 | Blue | ABC123 | Available | 50 |
| 2 | SUV | RangeRover | 2023 | Black | DL0125 | Available | 2000 |
| 3 | Sedan | Audi | 2024 | Red | DL0125 | Available | 5525 |
| 4 | Supercar | Lamborgini | 2024 | Yellow | DL4525 | Available | 9542360 |
| 5 | Truck | Tesla | 2023 | Silver | DL012545 | Available | 7000 |
+-----+-----+-----+-----+-----+-----+-----+-----+
rows in set (0.00 sec)
```

2) RESERVATION MENU

```
Run main x Admin x

C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\carconnect\main.py

Choose an option:
1. Register Customer
2. Add Vehicle
3. Insert Reservation
4. Register Admin
5. Exit
Enter your choice (1-5): 3

Reservation Options:
1. Create New Reservation
2. Update Reservation
3. Cancel Reservation
4. Go Back
Enter your choice (1-4): 1
Enter Reservation ID: 105
Enter Customer ID: 2
Enter Vehicle ID: 3
Enter Start Date (YYYY-MM-DD): 2024-01-02
Enter End Date (YYYY-MM-DD): 2024-01-11
Enter Total Cost: 5000
Enter Status: Booked
Reservation data saved to database.
```

DATA SUCCESSFULLY SAVED INTO OUR RESERVATION TABLE AND DATABASE

```
mysql> SELECT * FROM reservations;
+-----+-----+-----+-----+-----+-----+-----+
| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
+-----+-----+-----+-----+-----+-----+-----+
| 101 | 1 | 1 | 2024-01-02 | 2024-01-03 | 5642 | Booked |
| 102 | 2 | 2 | 2024-02-10 | 2024-02-12 | 5248 | Booked |
| 103 | 4 | 3 | 2024-02-03 | 2024-02-06 | 9652 | Booked |
| 104 | 5 | 4 | 2024-02-11 | 2024-02-15 | 9653 | Booked |
| 105 | 2 | 3 | 2024-01-02 | 2024-01-11 | 5000 | Booked |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

3) ADDING AND UPDATING NEW VALUE INTO ADMIN TABLE

```
C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\carconnect\main.py

Choose an option:
1. Register Customer
2. Add Vehicle
3. Insert Reservation
4. Register Admin
5. Exit
Enter your choice (1-5): 4

Admin Options:
1. Register New Admin
2. Update Admin
3. Delete Admin
4. Go Back
Enter your choice (1-4): 1
Enter Admin ID: 6
Enter First Name: Khusi
Enter Last Name: Kumari
Enter Email: kushi@gmail.com
Enter Phone Number: 9546235254
Enter Username: kushi384
Enter Password: kushi564
Enter Role: Manager
Enter Join Date (YYYY-MM-DD): 2024-02-03
Admin data saved to database.
```

DATA SUCCESSFULLY SAVED INTO OUR ADMIN TABLE AND DATABASE

```
mysql> select * from admins;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| AdminID | FirstName | LastName | Email           | PhoneNumber | Username | Password | Role      | JoinDate   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     1   | Rama     | Kumar    | rama66@gmail.com | 9654854254  | Rama35   | rama647   | Maneger   | 2024-01-03 |
|     2   | Aman     | K        | aman344@yahoo.com | 9654235685  | Aman38   | aman456   | Supplier   | 2024-01-09 |
|     3   | Amrita   | Rani    | amrita3738@gmail.co | 6532542584 | Amrita748 | amrit4984 | Maneger   | 2024-02-03 |
|     4   | Sushanta | Singh   | sushant736@gmail.com | 8546325865 | Sushant33 | sushant34 | GlobalHeadOfSale | 2024-02-01 |
|     6   | Kushi    | Kumari  | kushi@gmail.com   | 95465254   | kushi384 | kushi564  | Maneger   | 2024-02-03 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)

mysql> -
```

******Thank You******