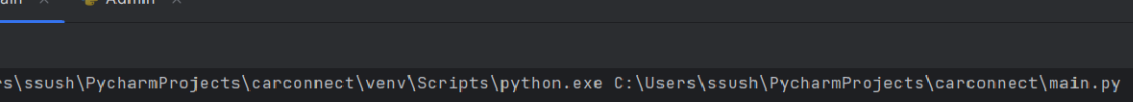


CASE STUDY PYTHON

NAME: - SUSHANT KUMAR SINGH

PROJECT:- CarConnect, a Car Rental Platform

Main Menu



The screenshot shows the Run console in PyCharm. At the top, there are tabs for 'main' and 'Admin'. The console output shows the command to run the script, followed by the program's prompt 'Choose an option:' and a numbered list of options. The user has entered '2' as their choice.

```
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] x
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: vijaythala@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

```
mysql> select * from customers;
```

CustomerID	FirstName	LastName	Email	PhoneNumber	Address	Username	Password	RegistrationDate
1	Sushant	Kumar	sushant763@gmail.com	96546965	Delhi	Sushan373	sush436	2024-02-02
2	Anu	Singh	anu@gmail.com	95465245	main park street	Anu3049	anu494	2024-01-01
4	Ravi	K	ravi8388@gmail.com	95462548	mumbai	Ravier58	ravi4959	2024-02-03
5	Sumita	Patil	sunita399@gmail.com	95466524	South River Park	Sunita737	sunita45	2024-01-01
6	Vijay	T	vijaythola@gmail.com	95462354	Old MG Road	Vijay383	vijay47	2024-02-16

```
5 rows in set (0.01 sec)
```

```
mysql>
```

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)

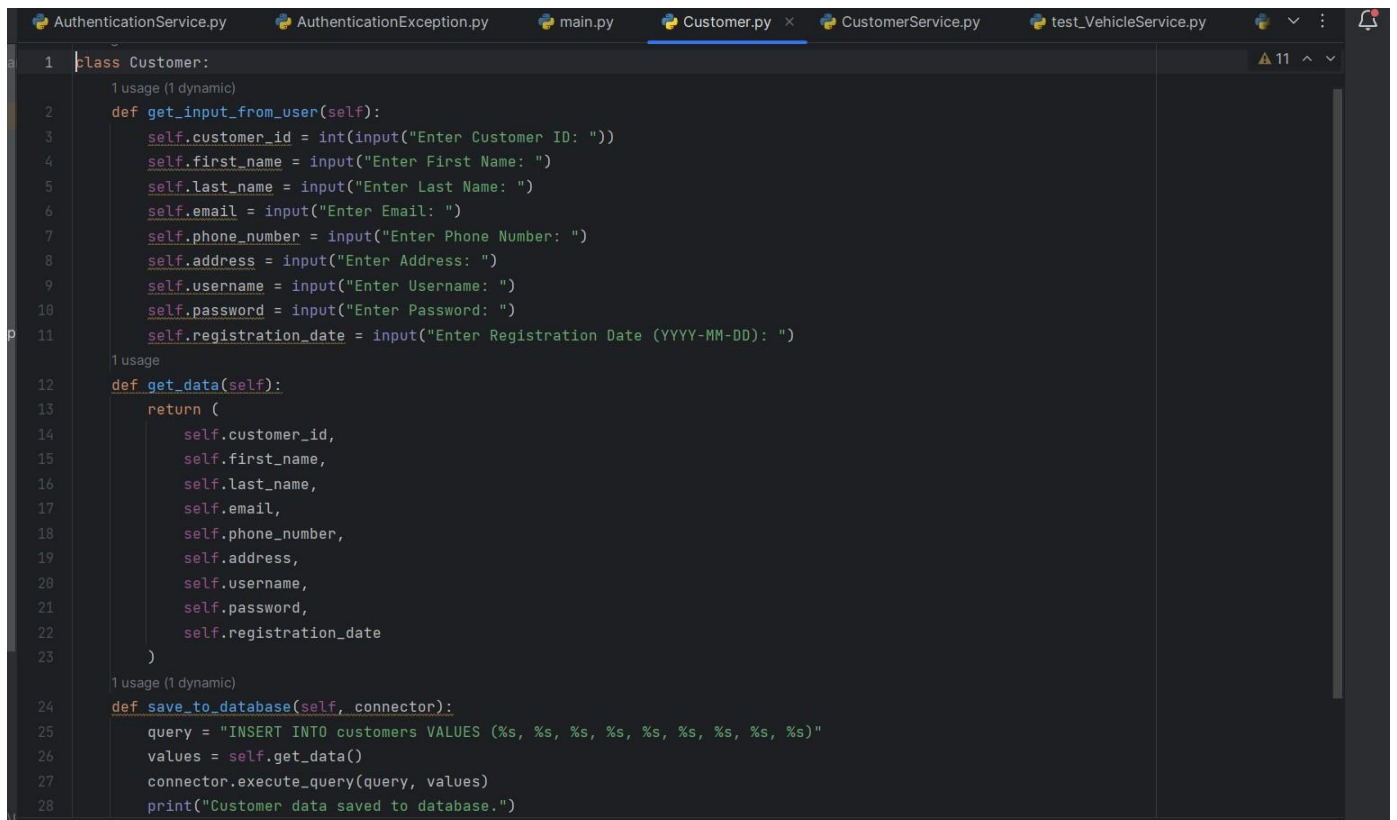
```
CREATE TABLE IF NOT EXISTS admins (  
    AdminID INT PRIMARY KEY,  
    FirstName VARCHAR(255),  
    LastName VARCHAR(255),  
    Email VARCHAR(255),  
    PhoneNumber VARCHAR(15),  
    Username VARCHAR(255),  
    Password VARCHAR(255),  
    Role VARCHAR(255),  
    JoinDate DATE  
);
```

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:

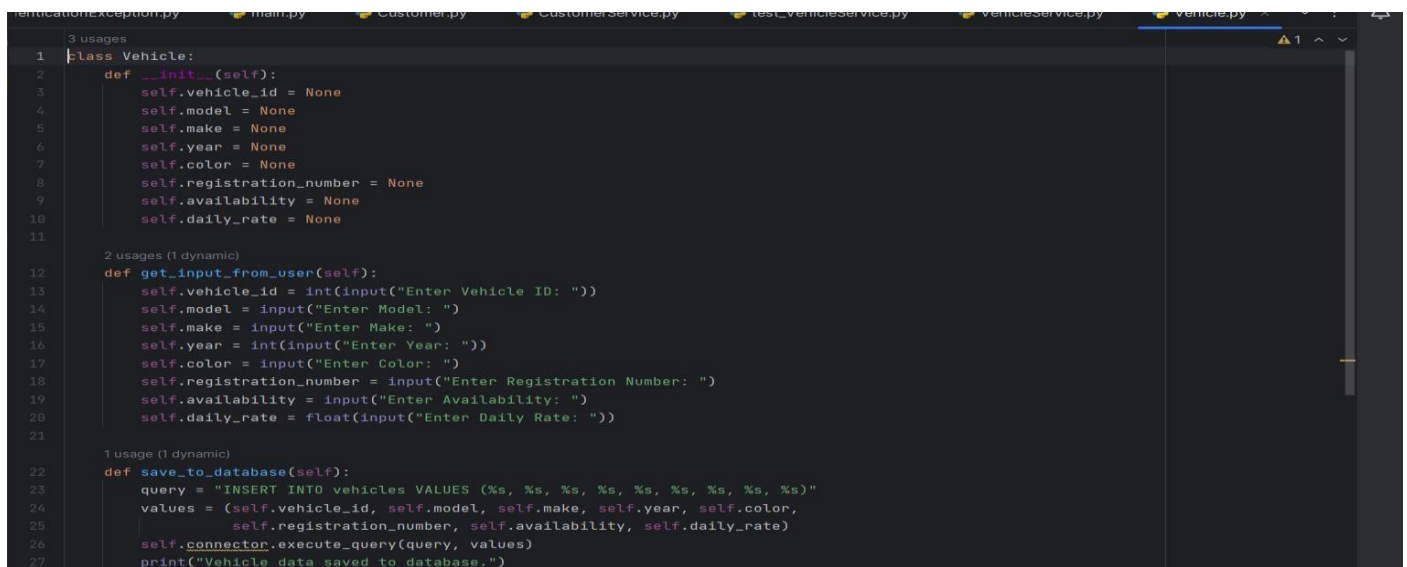
- Properties: CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate
- Methods: Authenticate(password)



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

Admin:

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

```
1 class Admin:
2     def get_input_from_user(self):
3         self.admin_id = int(input("Enter Admin ID: "))
4         self.first_name = input("Enter First Name: ")
5         self.last_name = input("Enter Last Name: ")
6         self.email = input("Enter Email: ")
7         self.phone_number = input("Enter Phone Number: ")
8         self.username = input("Enter Username: ")
9         self.password = input("Enter Password: ")
10        self.role = input("Enter Role: ")
11        self.join_date = input("Enter Join Date (YYYY-MM-DD): ")
12
13    def get_data(self):
14        return (
15            self.admin_id,
16            self.first_name,
17            self.last_name,
18            self.email,
19            self.phone_number,
20            self.username,
21            self.password,
22            self.role,
23            self.join_date
24        )
25
26    def save_to_database(self, connector):
27        query = "INSERT INTO admins VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
28        values = self.get_data()
29        connector.execute_query(query, values)
30        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.py
1 from ICustomerService import ICustomerService
2 from InvalidInputException import InvalidInputException
3 from DatabaseConnectionException import DatabaseConnectionException
4
5 class CustomerService(ICustomerService):
6     def __init__(self, connector):
7         self.connector = connector
8
9     def get_customer_by_id(self, customer_id):
10        try:
11            query = "SELECT * FROM customers WHERE CustomerID = %s"
12            result = self.connector.execute_query(query, (customer_id,), fetch_one=True)
13            if result:
14                customer_data = result
15                print(f"Customer found with ID {customer_id}: {customer_data}")
16            else:
17                print(f"No customer found with ID {customer_id}")
18        except Exception as e:
19            print(f"Error: {e}")
20
21    def get_customer_by_username(self, username):
22        try:
23            query = "SELECT * FROM customers WHERE Username = %s"
24            result = self.connector.execute_query(query, (username,), fetch_one=True)
25            if result:
26                customer_data = result
27                print(f"Customer found with username {username}: {customer_data}")
28            else:
29                print(f"No customer found with username {username}")
30        except Exception as e:
31            print(f"Error: {e}")
```

```
CustomerService.py x test_VehicleService.py VehicleService.py Vehicle.py Reservation.py Admin.py test_CustomerService.py
29        print(f"Error: {e}")
30    1 usage
31    def register_customer(self, customer, connector):
32        customer.get_input_from_user()
33        customer.save_to_database(connector)
34    1 usage (1 dynamic)
35    def save_to_database(self, customer_data, connector):
36        try:
37            connector.execute_query("""
38                INSERT INTO customers
39                (CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
40                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
41            """, customer_data)
42            print("Customer data saved to database.")
43        except Exception as e:
44            print(f"Error: {e}")
45    1 usage
46    def update_customer(self, customer_id, updated_customer):
47        try:
48            # Check if the customer exists
49            existing_customer = self.get_customer_by_id(customer_id)
50            if not existing_customer:
51                raise InvalidInputException("Customer not found")
52
53            query = """
54                UPDATE customers
55                SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Password=?, RegistrationDate=?
56                WHERE CustomerID=?
57            """
58            params = (
59                updated_customer["FirstName"],
```

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

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

```
CustomerService.py  test_VehicleService.py  VehicleService.py x  Vehicle.py  Reservation.py  Admin.py  test_CustomerService.py  ⌵  🔔

29  def add_vehicle(self, vehicle_data):
30      try:
31         self.connector.execute_query("""
32             INSERT INTO vehicles
33             (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}")
   1 usage
39  def update_vehicle(self, vehicle_data):
40      try:
41         query = """
42             UPDATE vehicles
43             SET Model = %s, Make = %s, Year = %s, Color = %s,
44                 RegistrationNumber = %s, Availability = %s, DailyRate = %s
45             WHERE VehicleID = %s
46         """
47         self.connector.execute_query(query, vehicle_data[1:] + (vehicle_data[0],))
48         print(f"Vehicle with ID {vehicle_data[0]} updated successfully.")
49     except Exception as e:
50         print(f"Error: {e}")
   1 usage
51  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  Vehicle.py  Reservation.py  Admin...
1  from IReservationService import IReservationService
2  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     1 usage
31     def create_reservation(self, reservation_data):
```

```
CustomerService.py  test_VehicleService.py  VehicleService.py  ReservationService.py  Vehicle.py  Reservation.py  Admin...
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
53  def cancel_reservation(self, reservation_id):
54      try:
55          query = "DELETE FROM reservations WHERE ReservationID = %s"
56          self.connector.execute_query(query, (reservation_id,))
57          print(f"Reservation with ID {reservation_id} canceled successfully.")
58      except Exception as e:
59          print(f"Error: {e}")
```


AdminService (implements IAdminService):

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

```
CustomerService.py  VehicleService.py  ReservationService.py  AdminService.py  Vehicle.py  Reservation.py  Admin.py  ⌵  🔔
1  from IAdminService import IAdminService
2  2 usages
3  class AdminService(IAdminService):
4      def __init__(self, connector):
5          self.connector = connector
6      def get_admin_by_id(self, admin_id):
7          try:
8              query = "SELECT * FROM admins WHERE AdminID = %s"
9              result = self.connector.execute_query(query, (admin_id,), fetch_one=True)
10             if result:
11                 admin_data = result
12                 print(f"Admin found with ID {admin_id}: {admin_data}")
13             else:
14                 print(f"No admin found with ID {admin_id}")
15         except Exception as e:
16             print(f"Error: {e}")
17     def get_admin_by_username(self, username):
18         try:
19             query = "SELECT * FROM admins WHERE Username = %s"
20             result = self.connector.execute_query(query, (username,), fetch_one=True)
21             if result:
22                 admin_data = result
23                 print(f"Admin found with username {username}: {admin_data}")
24             else:
25                 print(f"No admin found with username {username}")
26         except Exception as e:
27             print(f"Error: {e}")
28     1 usage
29     def register_admin(self, admin_data):
30         try:
31             self.connector.execute_query("""
```

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

DatabaseConnector:

- A class responsible for handling database connections and interactions.

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

```
vice.py  VehicleService.py  ReservationService.py  AdminService.py  AuthenticationService.py  DataConnector.py  Vehicle.py  ⌵  🔔
29      Model VARCHAR(255),
30      Make VARCHAR(255),
31      Year INT,
32      Color VARCHAR(255),
33      RegistrationNumber VARCHAR(255),
34      Availability BOOLEAN,
35      DailyRate FLOAT
36  );
37  """,
38  """
39  CREATE TABLE IF NOT EXISTS reservations (
40      ReservationID INT AUTO_INCREMENT PRIMARY KEY,
41      CustomerID INT,
42      VehicleID INT,
43      StartDate DATE,
44      EndDate DATE,
45      TotalCost FLOAT,
46      Status VARCHAR(255),
47      FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID),
48      FOREIGN KEY (VehicleID) REFERENCES vehicles(VehicleID)
49  );
50  """,
51  """
52  CREATE TABLE IF NOT EXISTS admins (
53      AdminID INT AUTO_INCREMENT PRIMARY KEY,
54      FirstName VARCHAR(255),
55      LastName VARCHAR(255),
56      Email VARCHAR(255),
57      PhoneNumber VARCHAR(15),
58      Username VARCHAR(255),
59      Password VARCHAR(255),
60      RegistrationDate DATE
61  );
62  """
63  for query in create_tables_queries:
64      self.cursor.execute(query)
65  self.connection.commit()
```

```
VehicleService.py  ReservationService.py  AdminService.py  AuthenticationService.py  DataConnector.py  Vehicle.py
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
72     def execute_query(self, query, values=None):
73         try:
74             if values:
75                 self.cursor.execute(query, values)
76             else:
77                 self.cursor.execute(query)
78             self.connection.commit()
79         except mysql.connector.Error as err:
80             print(f"Error: {err}")
81
82     def insert_user_input_data(self, customer_data, vehicle_data, reservation_data, admin_data):
83         try:
84             # Insert customer data
85             self.cursor.execute("""
86                 INSERT INTO customers
87                 (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
88                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
89             """, customer_data)
90             self.cursor.execute("""
91                 INSERT INTO vehicles
92                 (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
93                 VALUES (%s, %s, %s, %s, %s, %s, %s)
94             """, vehicle_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             self.cursor.execute("""
94                 INSERT INTO reservations
95                 (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
96                 VALUES (%s, %s, %s, %s, %s, %s)
97             """, reservation_data)
98             self.cursor.execute("""
99                 INSERT INTO admins
100                 (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
101                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
102             """, admin_data)
103             self.connection.commit()
104         except mysql.connector.Error as err:
105             print(f"Error: {err}")
106
```

Main.py Class

```
AdminNotFoundException.py  InvalidInputException.py  DatabaseConnectionException.py  main.py  Customer.py  CustomerService.py  IAdminService.py
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:
```

```
py  AdminNotFoundException.py  InvalidInputException.py  DatabaseConnectionException.py  main.py  Customer.py  CustomerService.py  IAdminService.py
27         if choice == '1':
28             while True:
29                 print("\nCustomer Options:")
30                 print("1. Register New Customer")
31                 print("2. Update Customer")
32                 print("3. Delete Customer")
33                 print("4. Go Back")
34                 customer_choice = input("Enter your choice (1-4): ")
35                 if customer_choice == '1':
36                     new_customer = Customer()
37                     customer_service.register_customer(new_customer, connector)
38                 elif customer_choice == '2':
39                     customer_id = input("Enter Customer ID to update: ")
40                     updated_customer = Customer()
41                     customer_service.update_customer(customer_id, updated_customer)
42
43                 elif customer_choice == '3':
44                     customer_id = input("Enter Customer ID to delete: ")
45                     customer_service.delete_customer(customer_id)
46                 elif customer_choice == '4':
47                     break
48                 else:
49                     print("Invalid choice. Please enter a number between 1 and 4.")
50             elif choice == '2':
51                 while True:
52                     print("\nVehicle Options:")
53                     print("1. Add New Vehicle")
54
55                     print("2. Update Vehicle")
```

```
AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py x Customer.py CustomerService.py IAdminService.py
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':
```

```
.py AdminNotFoundException.py InvalidInputException.py DatabaseConnectionException.py main.py x Customer.py CustomerService.py IAdminService.py
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()
123
124 if __name__ == "__main__":
125     main()
126
```


Interfaces:

ICustomerService:

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



The screenshot shows a code editor with the file `ICustomerService.py` selected. The code defines an abstract class `ICustomerService` that inherits from `ABC` and implements `abstractmethod`. It contains five abstract methods: `get_customer_by_id`, `get_customer_by_username`, `register_customer`, `update_customer`, and `delete_customer`. Each method is decorated with `@abstractmethod` and has a `pass` statement. The editor's interface includes a top bar with several open files and a right sidebar with a search icon.

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

IVehicleService:

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

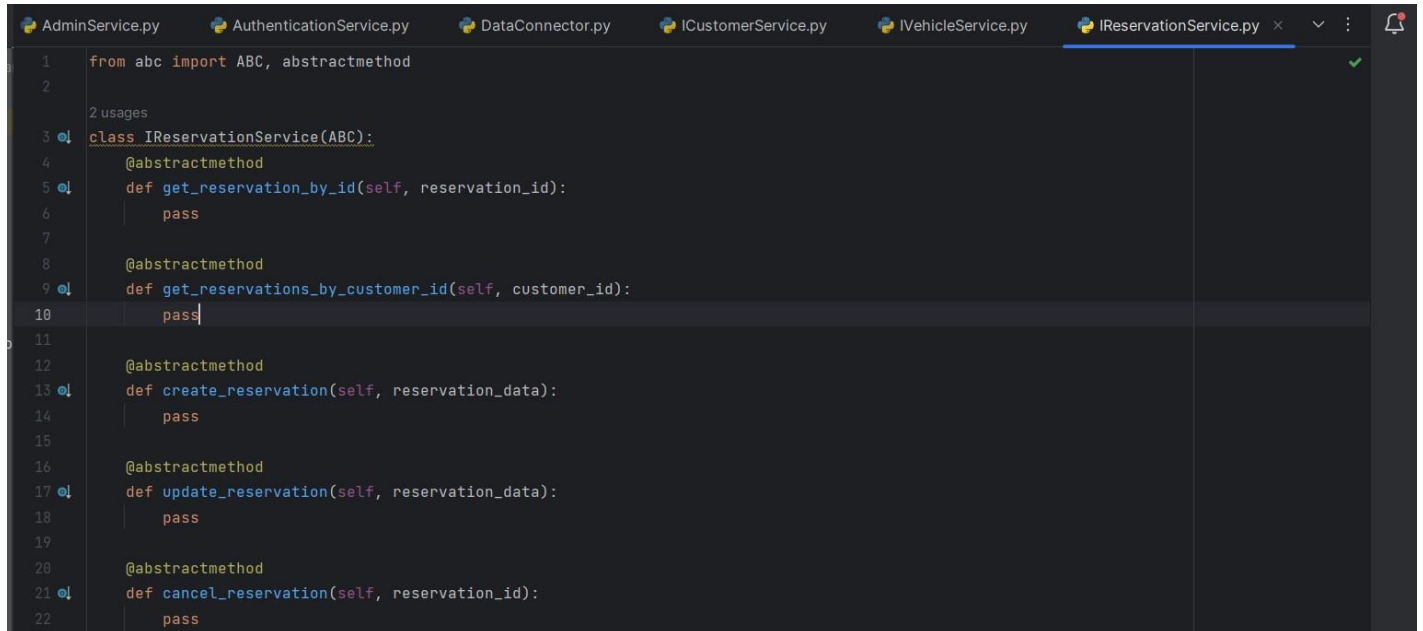


The screenshot shows a code editor with the file `IVehicleService.py` selected. The code defines an abstract class `IVehicleService` that inherits from `ABC` and implements `abstractmethod`. It contains five abstract methods: `get_vehicle_by_id`, `get_available_vehicles`, `add_vehicle`, `update_vehicle`, and `remove_vehicle`. Each method is decorated with `@abstractmethod` and has a `pass` statement. The editor's interface includes a top bar with several open files and a right sidebar with a search icon.

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

IReservationService:

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



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

IAdminService:

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



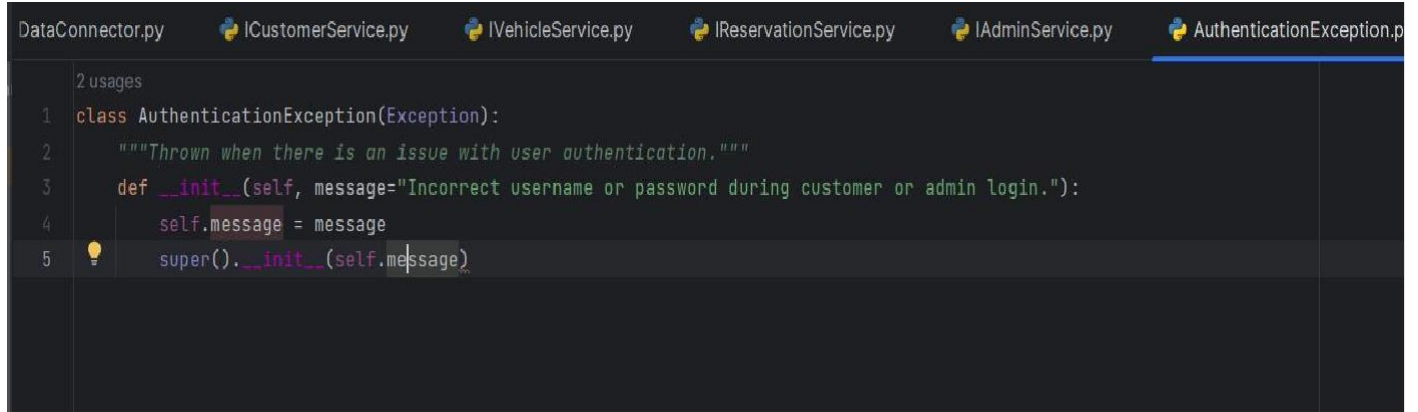
```
1 from abc import ABC, abstractmethod
2
3 2 usages
4 class IAdminService(ABC):
5     @abstractmethod
6     def get_admin_by_id(self, admin_id):
7         pass
8
9     @abstractmethod
10    def get_admin_by_username(self, username):
11        pass
12
13    @abstractmethod
14    def register_admin(self,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

Custom Exceptions:

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

AuthenticationException:

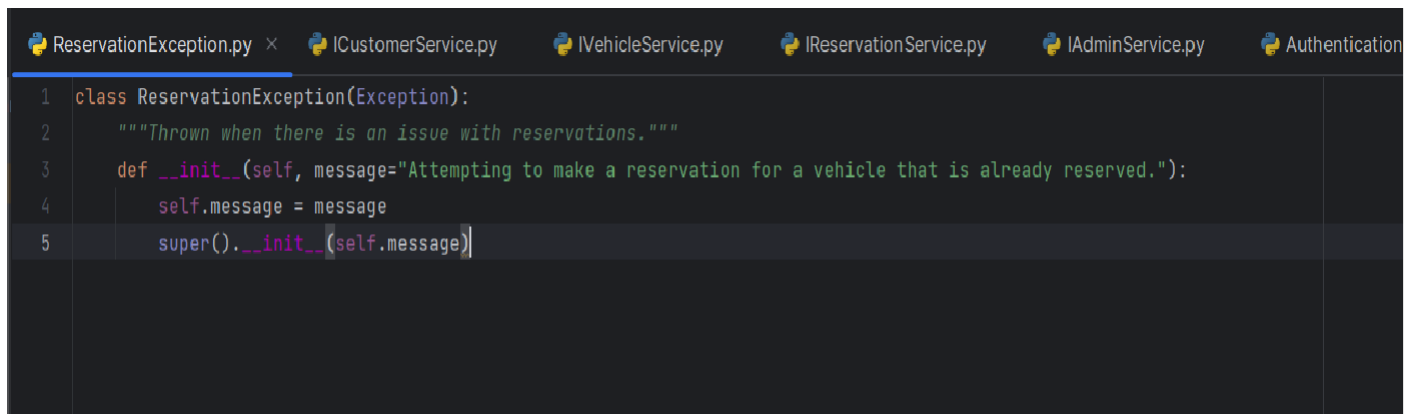
- Thrown when there is an issue with user authentication.
- Example Usage: Incorrect username or password during customer or admin login.



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

ReservationException:

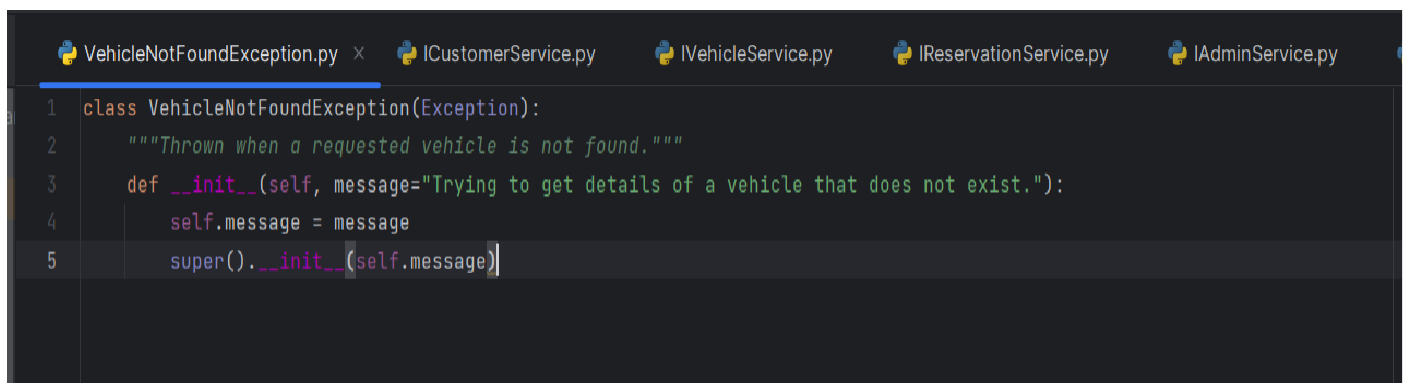
- Thrown when there is an issue with reservations.
- Example Usage: Attempting to make a reservation for a vehicle that is already reserved.



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

```
ception.py  AdminNotFoundException.py  ICustomerService.py  IVehicleService.py  IReservationService.py  IAdminService.py

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.

```
ndException.py  InvalidInputException.py  ICustomerService.py  IVehicleService.py  IReservationService.py  IAdminService.py

3 usages
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.

```
alidInputException.py  DatabaseConnectionException.py  ICustomerService.py  IVehicleService.py  IReservationService.py

2 usages
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.
2. Test updating customer information.
3. Test adding a new vehicle.
4. Test updating vehicle details.
5. Test getting a list of available vehicles.
6. Test getting a list of all vehicles.

```
1 import pytest
2 import mysql.connector
3
4
5 @pytest.fixture(scope="module")
6 def db_connection():
7     connection = mysql.connector.connect(
8         host="localhost",
9         user="root",
10        password="Sushant@9546",
11        database="carconnect1"
12    )
13    yield connection
14    connection.close()
15
16 def authenticate_customer(db_connection):
17     """Authenticate a customer and return the result."""
18     cursor = db_connection.cursor()
19     cursor.execute("SELECT username, password FROM customers WHERE customerid = 1")
20     result = [row for row in cursor]
21     cursor.close()
22     return result
23
24 def test_authenticate_customer(db_connection):
25     """Test the authentication of a customer."""
26     assert authenticate_customer(db_connection) == [('Sushan373', 'sush436')]
27
28 def update_customer(db_connection):
```

```
28 def update_customer(db_connection):
29     """Update a customer's first name and return True on success."""
30     cursor = db_connection.cursor()
31     cursor.execute("UPDATE customers SET firstname = 'Anu' WHERE customerid = 2")
32     db_connection.commit()
33     cursor.close()
34     return True
35
36 def test_update_customer(db_connection):
37     """Test updating a customer's information."""
38     assert update_customer(db_connection) == True
39
40 def add_vehicle(db_connection):
41     """Add a vehicle to the database and return True on success."""
42     cursor = db_connection.cursor()
43     cursor.execute(
44         "INSERT INTO vehicles(Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate) "
45         "VALUES (%s, %s, %s, %s, %s, %s, %s)",
46         ('Audi', 'Sedan', 2023, 'Red', 'DL0125', 1, 5525)
47     )
48     db_connection.commit()
49     cursor.close()
50     return True
51
52 def test_add_vehicle(db_connection):
53     """Test adding a vehicle to the database."""
54     assert add_vehicle(db_connection) == True
55
56 def update_vehicle(db_connection):
```



```

56 def update_vehicle(db_connection):
57     """Update a vehicle's model and return True on success."""
58     cursor = db_connection.cursor()
59     cursor.execute("UPDATE vehicles SET model = 'range' WHERE vehicleid = 3")
60     db_connection.commit()
61     cursor.close()
62     return True
63
64 ▶ def test_update_vehicle(db_connection):
65     """Test updating a vehicle's information."""
66     assert update_vehicle(db_connection) == True
67
68 1 usage
69 def get_available_vehicles(db_connection):
70     """Retrieve the count of available vehicles from the database."""
71     cursor = db_connection.cursor()
72     cursor.execute("SELECT * FROM vehicles WHERE availability = 1")
73     result = cursor.fetchall()
74     cursor.close()
75     return len(result)
76
77 ▶ def test_get_available_vehicles(db_connection):
78     """Test retrieving the count of available vehicles."""
79     assert get_available_vehicles(db_connection) >= 0
80
81 1 usage
82 def get_all_vehicles(db_connection):
83     """Retrieve the count of all vehicles from the database."""
84     cursor = db_connection.cursor()
85     cursor.execute("SELECT * FROM vehicles")
86     result = cursor.fetchall()

```

22 6 ^ v

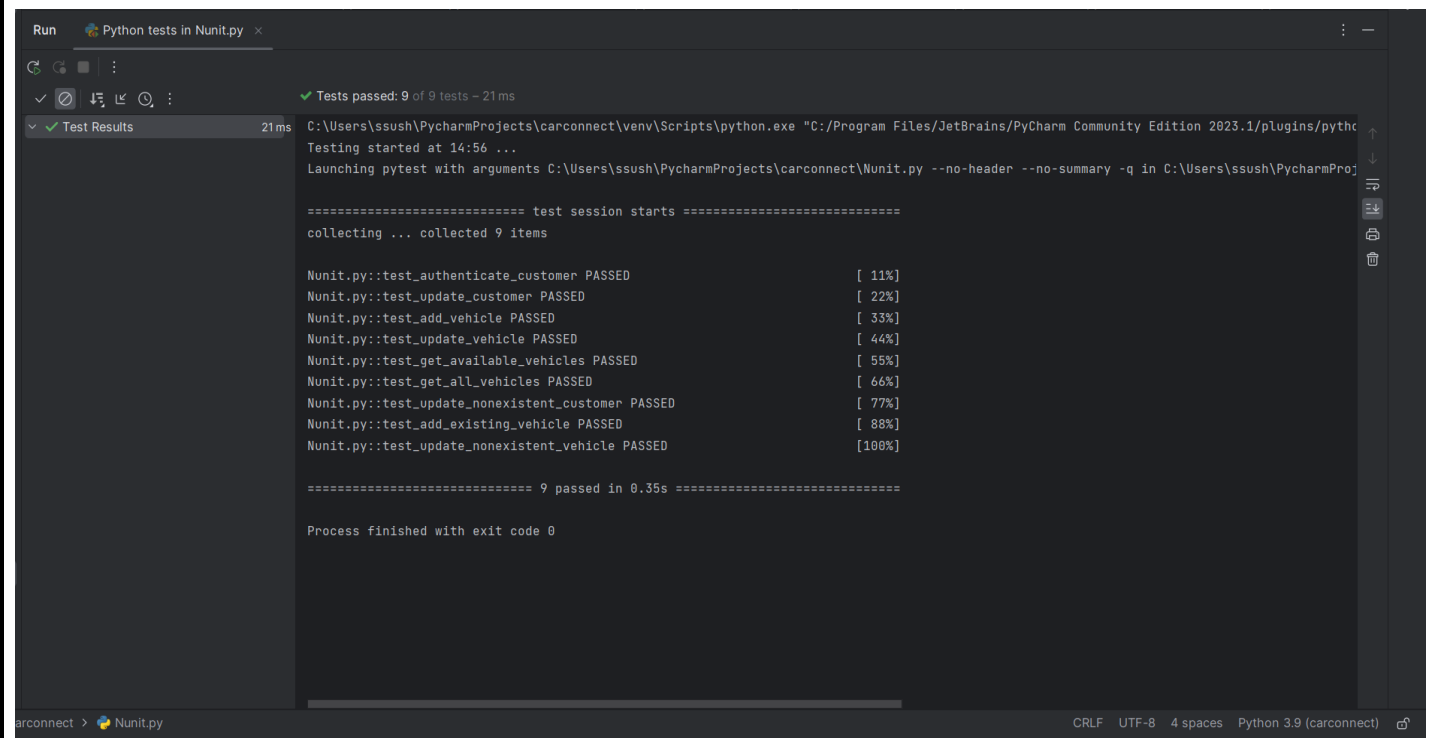
```

81     """Retrieve the count of all vehicles from the database."""
82     cursor = db_connection.cursor()
83     cursor.execute("SELECT * FROM vehicles")
84     result = cursor.fetchall()
85     cursor.close()
86     return len(result)
87
88 ▶ def test_get_all_vehicles(db_connection):
89     """Test retrieving the count of all vehicles."""
90     assert get_all_vehicles(db_connection) >= 0
91
92 # Additional test cases
93
94 """def test_authenticate_nonexistent_customer(db_connection):
95     Test authentication for a nonexistent customer.
96     assert authenticate_customer(db_connection) == []"""
97
98 ▶ def test_update_nonexistent_customer(db_connection):
99     """Test updating information for a nonexistent customer."""
100     assert update_customer(db_connection) == True
101
102 ▶ def test_add_existing_vehicle(db_connection):
103     """Test adding a vehicle that already exists in the database."""
104     assert add_vehicle(db_connection) == True
105
106 ▶ def test_update_nonexistent_vehicle(db_connection):
107     """Test updating information for a nonexistent vehicle."""
108     assert update_vehicle(db_connection) == True
109

```

22 6 ^ v

TEST RESULTS



The screenshot shows the PyCharm Run window for a Python test file named Nunit.py. The window title is "Run Python tests in Nunit.py". The status bar at the bottom indicates the file is "carconnect > Nunit.py" with settings "CRLF UTF-8 4 spaces Python 3.9 (carconnect)".

The Run window displays the following output:

```
✓ Tests passed: 9 of 9 tests - 21 ms
C:\Users\ssush\PycharmProjects\carconnect\venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.1/plugins/pythc
Testing started at 14:56 ...
Launching pytest with arguments C:\Users\ssush\PycharmProjects\carconnect\Nunit.py --no-header --no-summary -q in C:\Users\ssush\PycharmProj

===== test session starts =====
collecting ... collected 9 items

Nunit.py::test_authenticate_customer PASSED [ 11%]
Nunit.py::test_update_customer PASSED [ 22%]
Nunit.py::test_add_vehicle PASSED [ 33%]
Nunit.py::test_update_vehicle PASSED [ 44%]
Nunit.py::test_get_available_vehicles PASSED [ 55%]
Nunit.py::test_get_all_vehicles PASSED [ 66%]
Nunit.py::test_update_nonexistent_customer PASSED [ 77%]
Nunit.py::test_add_existing_vehicle PASSED [ 88%]
Nunit.py::test_update_nonexistent_vehicle PASSED [100%]

===== 9 passed in 0.35s =====

Process finished with exit code 0
```

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	Lamborghini	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: Maneger
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******