CASE STUDY PYTHON

NAME: - SUSHANT KUMAR SINGH

PROJECT:- CarConnect, a Car Rental Platform

Main Menu

```
Run   main   Admin

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   AdminNotFoundException.py   InvalidInputException.py   DatabaseConnectionException.py   main.py   Customer.py

Run   main

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@gmai.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        | vijaythala@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
);
```

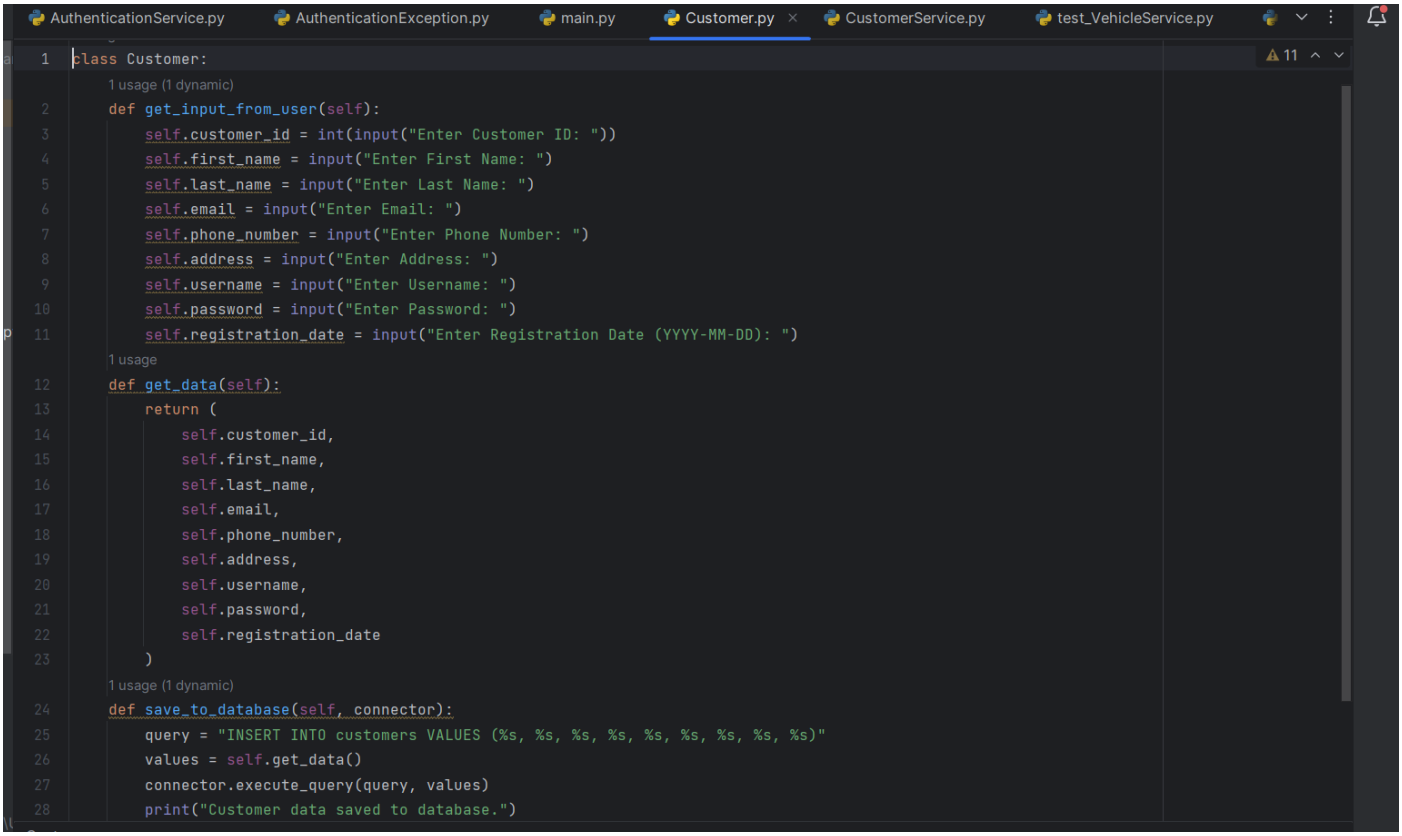
Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Classes:

Customer:

• Properties: CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate

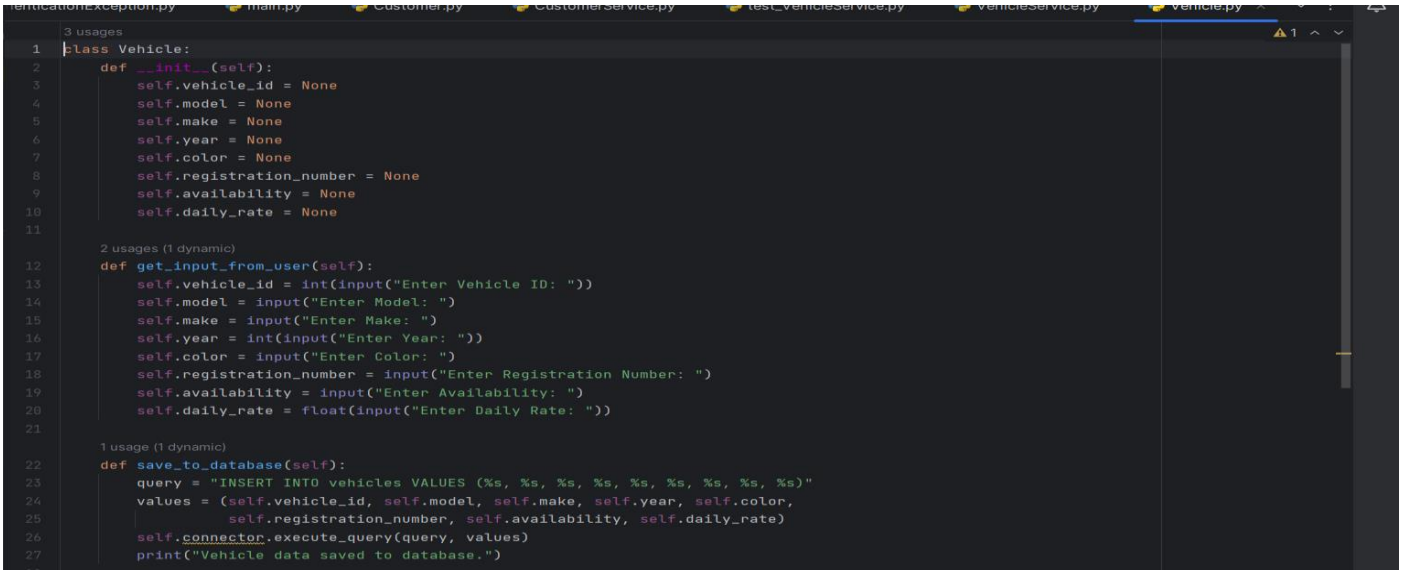
• Methods: Authenticate(password)



```python
class Customer:
    1 usage (1 dynamic)
    def get_input_from_user(self):
        self.customer_id = int(input("Enter Customer ID: "))
        self.first_name = input("Enter First Name: ")
        self.last_name = input("Enter Last Name: ")
        self.email = input("Enter Email: ")
        self.phone_number = input("Enter Phone Number: ")
        self.address = input("Enter Address: ")
        self.username = input("Enter Username: ")
        self.password = input("Enter Password: ")
        self.registration_date = input("Enter Registration Date (YYYY-MM-DD): ")
    1 usage
    def get_data(self):
        return (
            self.customer_id,
            self.first_name,
            self.last_name,
            self.email,
            self.phone_number,
            self.address,
            self.username,
            self.password,
            self.registration_date
        )
    1 usage (1 dynamic)
    def save_to_database(self, connector):
        query = "INSERT INTO customers VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = self.get_data()
        connector.execute_query(query, values)
        print("Customer data saved to database.")
```

Vehicle:

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



```python
3 usages
class Vehicle:
    def __init__(self):
        self.vehicle_id = None
        self.model = None
        self.make = None
        self.year = None
        self.color = None
        self.registration_number = None
        self.availability = None
        self.daily_rate = None

    2 usages (1 dynamic)
    def get_input_from_user(self):
        self.vehicle_id = int(input("Enter Vehicle ID: "))
        self.model = input("Enter Model: ")
        self.make = input("Enter Make: ")
        self.year = int(input("Enter Year: "))
        self.color = input("Enter Color: ")
        self.registration_number = input("Enter Registration Number: ")
        self.availability = input("Enter Availability: ")
        self.daily_rate = float(input("Enter Daily Rate: "))

    1 usage (1 dynamic)
    def save_to_database(self):
        query = "INSERT INTO vehicles VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = (self.vehicle_id, self.model, self.make, self.year, self.color,
                  self.registration_number, self.availability, self.daily_rate)
        self.connector.execute_query(query, values)
        print("Vehicle data saved to database.")
```

Reservation:

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

• Methods: CalculateTotalCost()

```python
3 usages
class Reservation:
    1 usage (1 dynamic)
    def get_input_from_user(self):
        self.reservation_id = int(input("Enter Reservation ID: "))
        self.customer_id = int(input("Enter Customer ID: "))
        self.vehicle_id = int(input("Enter Vehicle ID: "))
        self.start_date = input("Enter Start Date (YYYY-MM-DD): ")
        self.end_date = input("Enter End Date (YYYY-MM-DD): ")
        self.total_cost = float(input("Enter Total Cost: "))
        self.status = input("Enter Status: ")

    1 usage
    def get_data(self):
        return (
            self.reservation_id,
            self.customer_id,
            self.vehicle_id,
            self.start_date,
            self.end_date,
            self.total_cost,
            self.status
        )

    1 usage (1 dynamic)
    def save_to_database(self, connector):
        query = "INSERT INTO reservations VALUES (%s, %s, %s, %s, %s, %s, %s)"
        values = self.get_data()
        connector.execute_query(query, values)
        print("Reservation data saved to database.")
```

Admin:

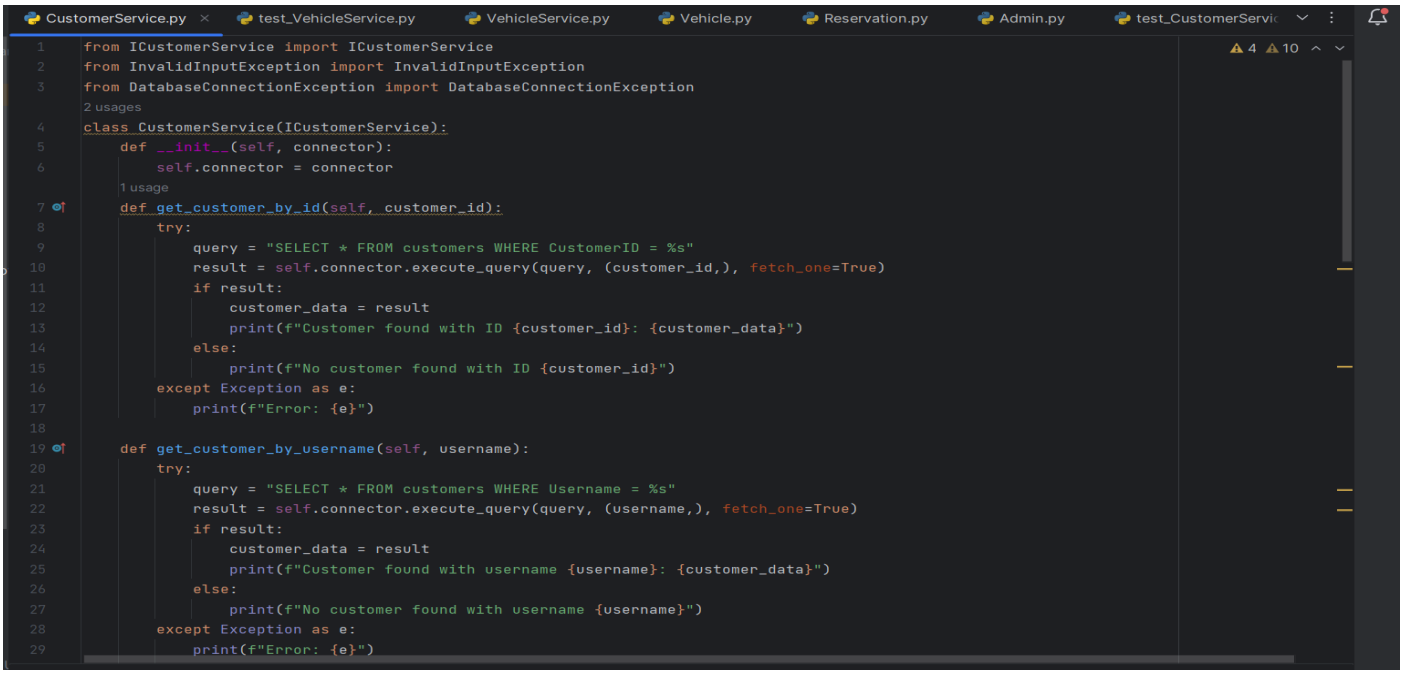
• Properties: AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate

• Methods: Authenticate(password)

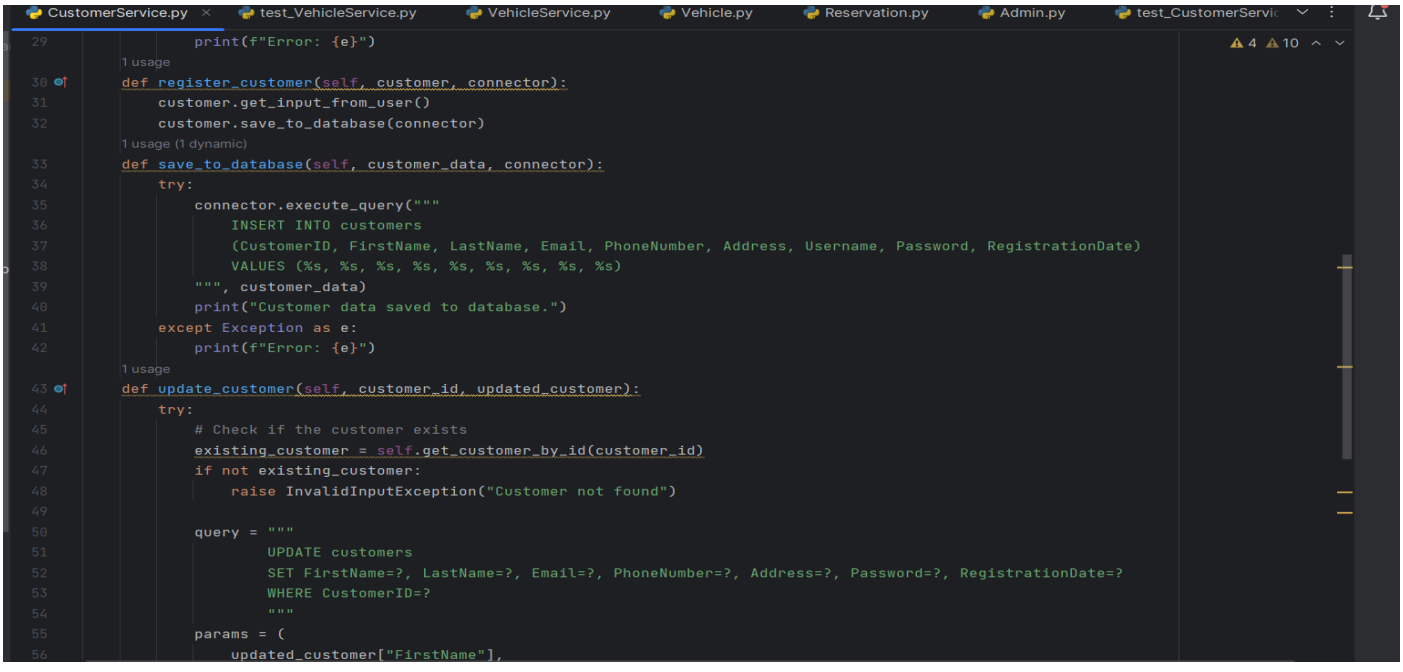
```python
class Admin:
    1 usage (1 dynamic)
    def get_input_from_user(self):
        self.admin_id = int(input("Enter Admin ID: "))
        self.first_name = input("Enter First Name: ")
        self.last_name = input("Enter Last Name: ")
        self.email = input("Enter Email: ")
        self.phone_number = input("Enter Phone Number: ")
        self.username = input("Enter Username: ")
        self.password = input("Enter Password: ")
        self.role = input("Enter Role: ")
        self.join_date = input("Enter Join Date (YYYY-MM-DD): ")
    1 usage
    def get_data(self):
        return (
            self.admin_id,
            self.first_name,
            self.last_name,
            self.email,
            self.phone_number,
            self.username,
            self.password,
            self.role,
            self.join_date
        )
    1 usage (1 dynamic)
    def save_to_database(self, connector):
        query = "INSERT INTO admins VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = self.get_data()
        connector.execute_query(query, values)
        print("Admin data saved to database.")
```

CustomerService (implements ICustomerService):

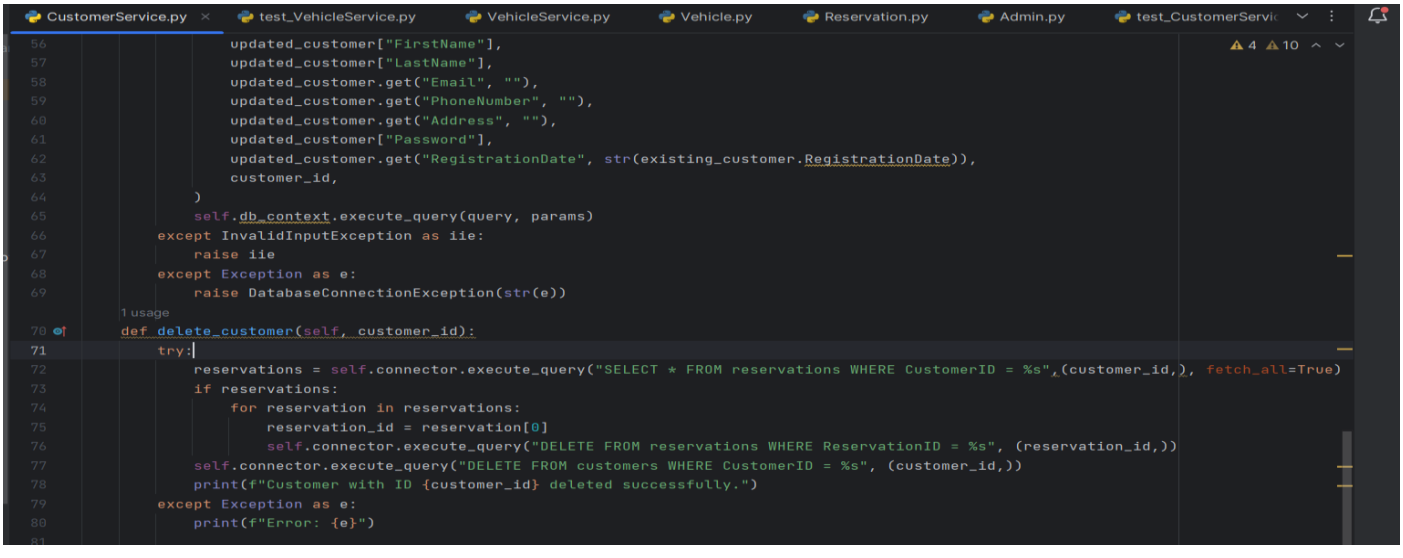
• Methods: GetCustomerById, GetCustomerByUsername, RegisterCustomer, UpdateCustomer, DeleteCustomer



```python
from ICustomerService import ICustomerService
from InvalidInputException import InvalidInputException
from DatabaseConnectionException import DatabaseConnectionException
2 usages
class CustomerService(ICustomerService):
    def __init__(self, connector):
        self.connector = connector
    1 usage
    def get_customer_by_id(self, customer_id):
        try:
            query = "SELECT * FROM customers WHERE CustomerID = %s"
            result = self.connector.execute_query(query, (customer_id,), fetch_one=True)
            if result:
                customer_data = result
                print(f"Customer found with ID {customer_id}: {customer_data}")
            else:
                print(f"No customer found with ID {customer_id}")
        except Exception as e:
            print(f"Error: {e}")

    def get_customer_by_username(self, username):
        try:
            query = "SELECT * FROM customers WHERE Username = %s"
            result = self.connector.execute_query(query, (username,), fetch_one=True)
            if result:
                customer_data = result
                print(f"Customer found with username {username}: {customer_data}")
            else:
                print(f"No customer found with username {username}")
        except Exception as e:
            print(f"Error: {e}")
```



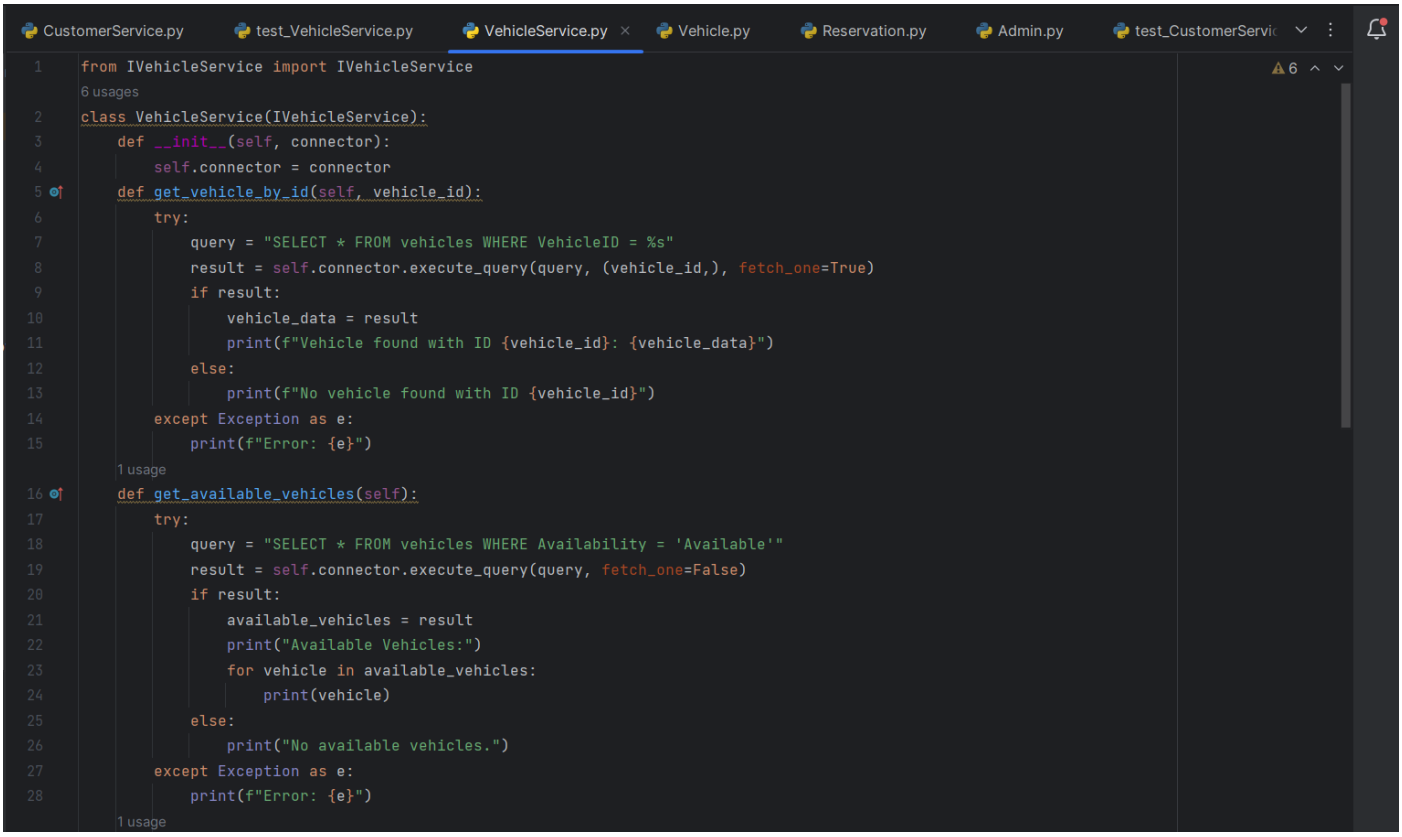
```python
            print(f"Error: {e}")
    1 usage
    def register_customer(self, customer, connector):
        customer.get_input_from_user()
        customer.save_to_database(connector)
    1 usage (1 dynamic)
    def save_to_database(self, customer_data, connector):
        try:
            connector.execute_query("""
                INSERT INTO customers
                (CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
            """, customer_data)
            print("Customer data saved to database.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def update_customer(self, customer_id, updated_customer):
        try:
            # Check if the customer exists
            existing_customer = self.get_customer_by_id(customer_id)
            if not existing_customer:
                raise InvalidInputException("Customer not found")

            query = """
                    UPDATE customers
                    SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Password=?, RegistrationDate=?
                    WHERE CustomerID=?
                    """
            params = (
                updated_customer["FirstName"],
```



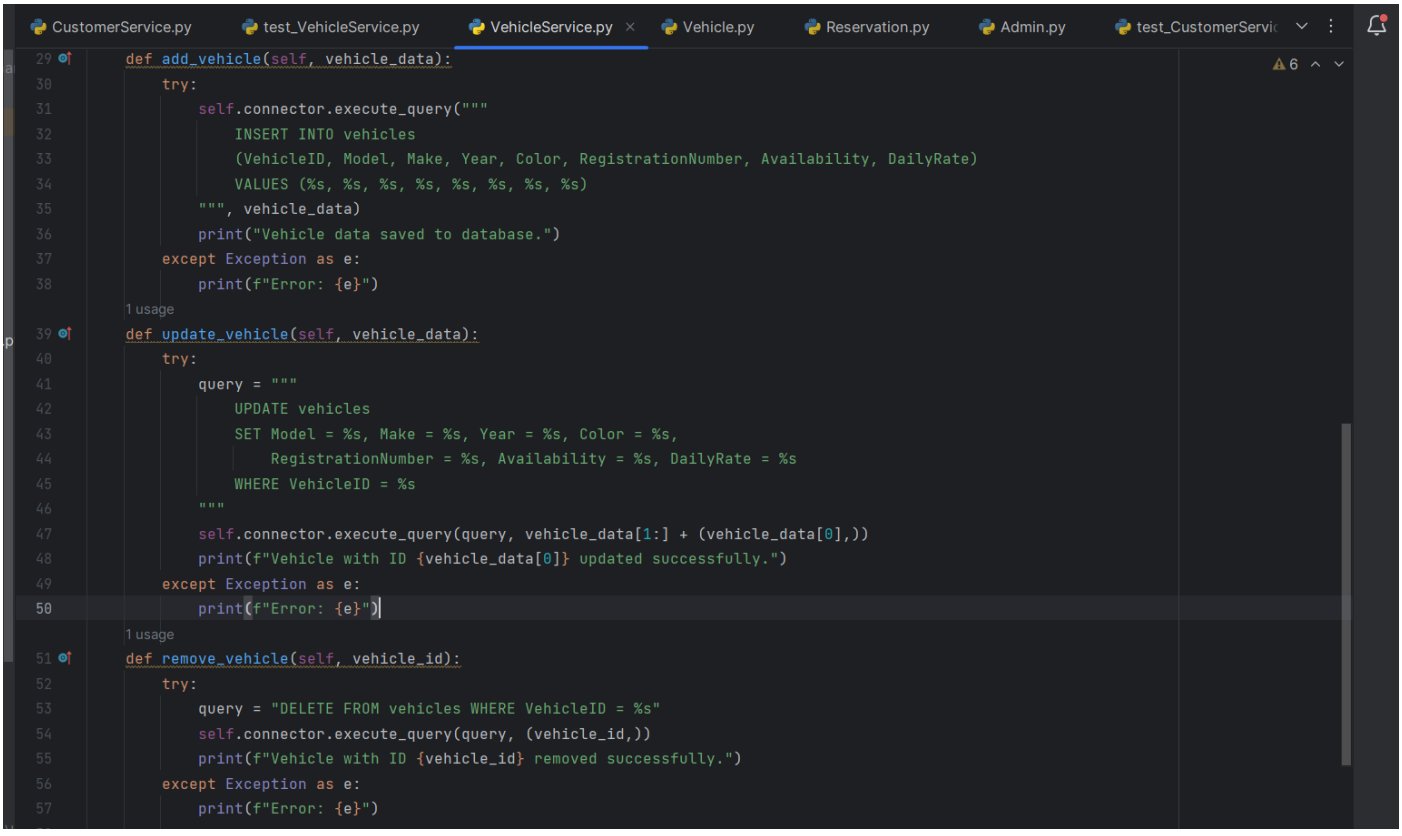
```python
                updated_customer["FirstName"],
                updated_customer["LastName"],
                updated_customer.get("Email", ""),
                updated_customer.get("PhoneNumber", ""),
                updated_customer.get("Address", ""),
                updated_customer["Password"],
                updated_customer.get("RegistrationDate", str(existing_customer.RegistrationDate)),
                customer_id,
            )
            self.db_context.execute_query(query, params)
        except InvalidInputException as iie:
            raise iie
        except Exception as e:
            raise DatabaseConnectionException(str(e))
    1 usage
    def delete_customer(self, customer_id):
        try:
            reservations = self.connector.execute_query("SELECT * FROM reservations WHERE CustomerID = %s",(customer_id,), fetch_all=True)
            if reservations:
                for reservation in reservations:
                    reservation_id = reservation[0]
                    self.connector.execute_query("DELETE FROM reservations WHERE ReservationID = %s", (reservation_id,))
            self.connector.execute_query("DELETE FROM customers WHERE CustomerID = %s", (customer_id,))
            print(f"Customer with ID {customer_id} deleted successfully.")
        except Exception as e:
            print(f"Error: {e}")
```

VehicleService (implements IVehicleService):

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



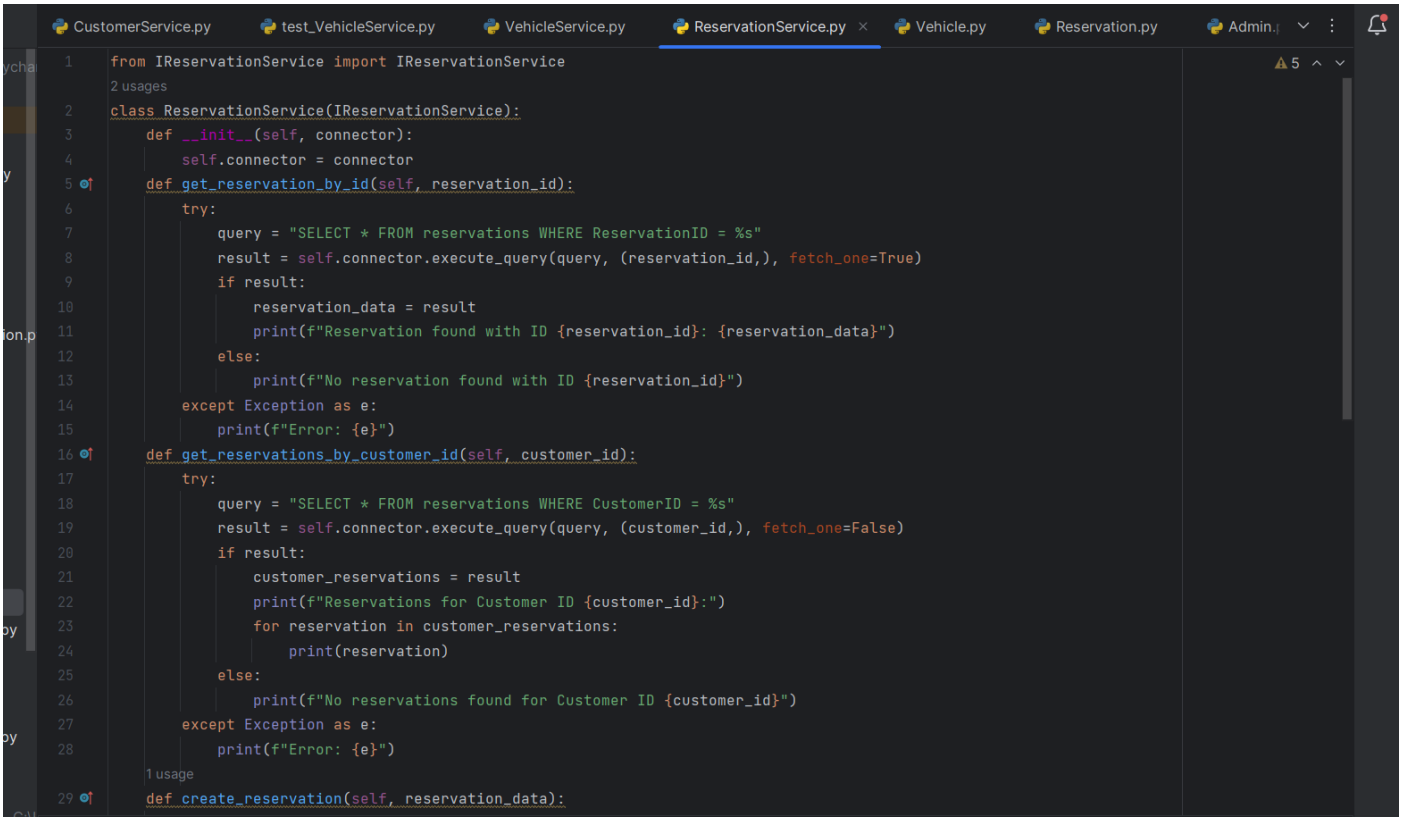
```python
from IVehicleService import IVehicleService
6 usages
class VehicleService(IVehicleService):
    def __init__(self, connector):
        self.connector = connector
    def get_vehicle_by_id(self, vehicle_id):
        try:
            query = "SELECT * FROM vehicles WHERE VehicleID = %s"
            result = self.connector.execute_query(query, (vehicle_id,), fetch_one=True)
            if result:
                vehicle_data = result
                print(f"Vehicle found with ID {vehicle_id}: {vehicle_data}")
            else:
                print(f"No vehicle found with ID {vehicle_id}")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def get_available_vehicles(self):
        try:
            query = "SELECT * FROM vehicles WHERE Availability = 'Available'"
            result = self.connector.execute_query(query, fetch_one=False)
            if result:
                available_vehicles = result
                print("Available Vehicles:")
                for vehicle in available_vehicles:
                    print(vehicle)
            else:
                print("No available vehicles.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
```



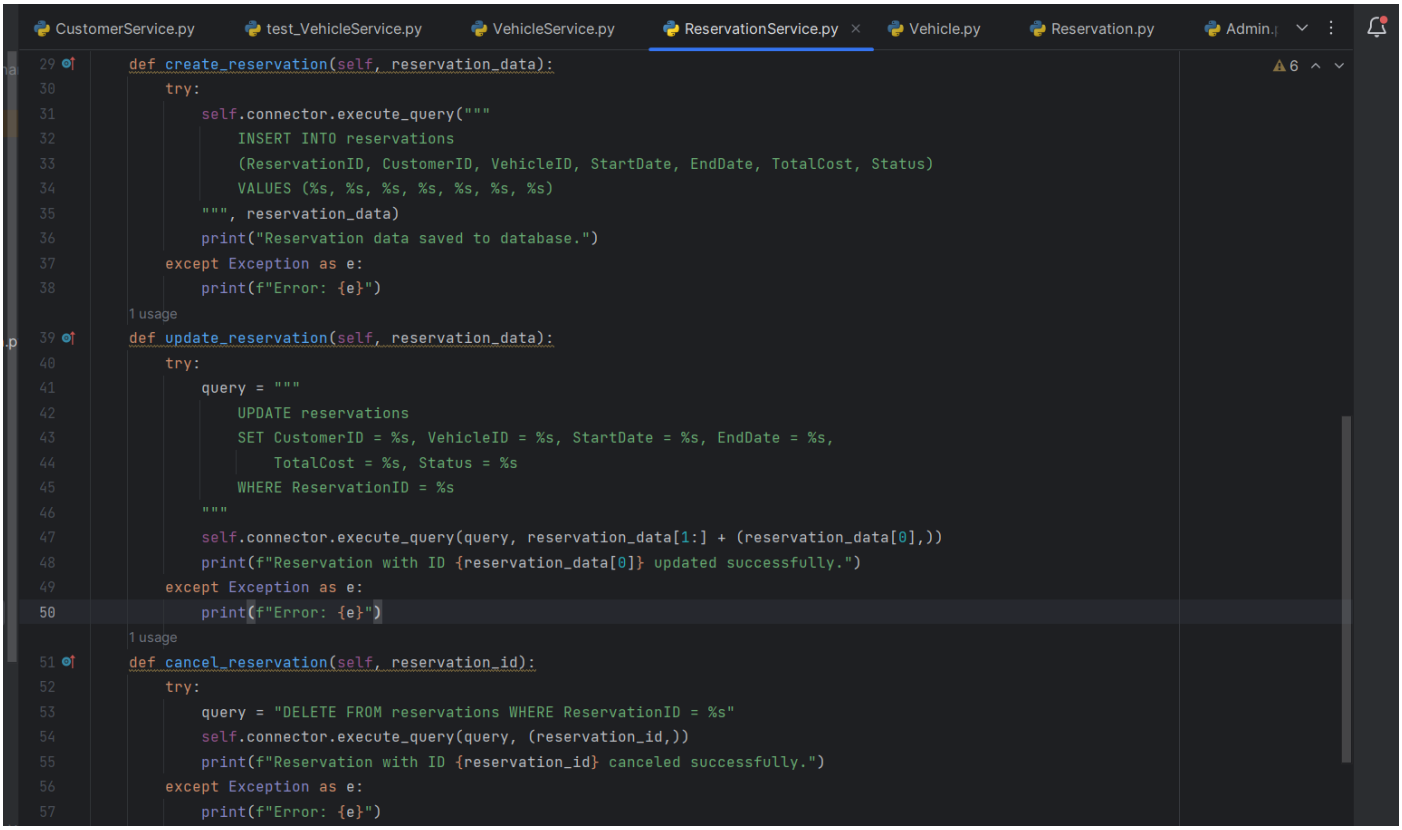
```python
    def add_vehicle(self, vehicle_data):
        try:
            self.connector.execute_query("""
                INSERT INTO vehicles
                (VehicleID, Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, vehicle_data)
            print("Vehicle data saved to database.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def update_vehicle(self, vehicle_data):
        try:
            query = """
                UPDATE vehicles
                SET Model = %s, Make = %s, Year = %s, Color = %s,
                    RegistrationNumber = %s, Availability = %s, DailyRate = %s
                WHERE VehicleID = %s
            """
            self.connector.execute_query(query, vehicle_data[1:] + (vehicle_data[0],))
            print(f"Vehicle with ID {vehicle_data[0]} updated successfully.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def remove_vehicle(self, vehicle_id):
        try:
            query = "DELETE FROM vehicles WHERE VehicleID = %s"
            self.connector.execute_query(query, (vehicle_id,))
            print(f"Vehicle with ID {vehicle_id} removed successfully.")
        except Exception as e:
            print(f"Error: {e}")
```

ReservationService (implements IReservationService):

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



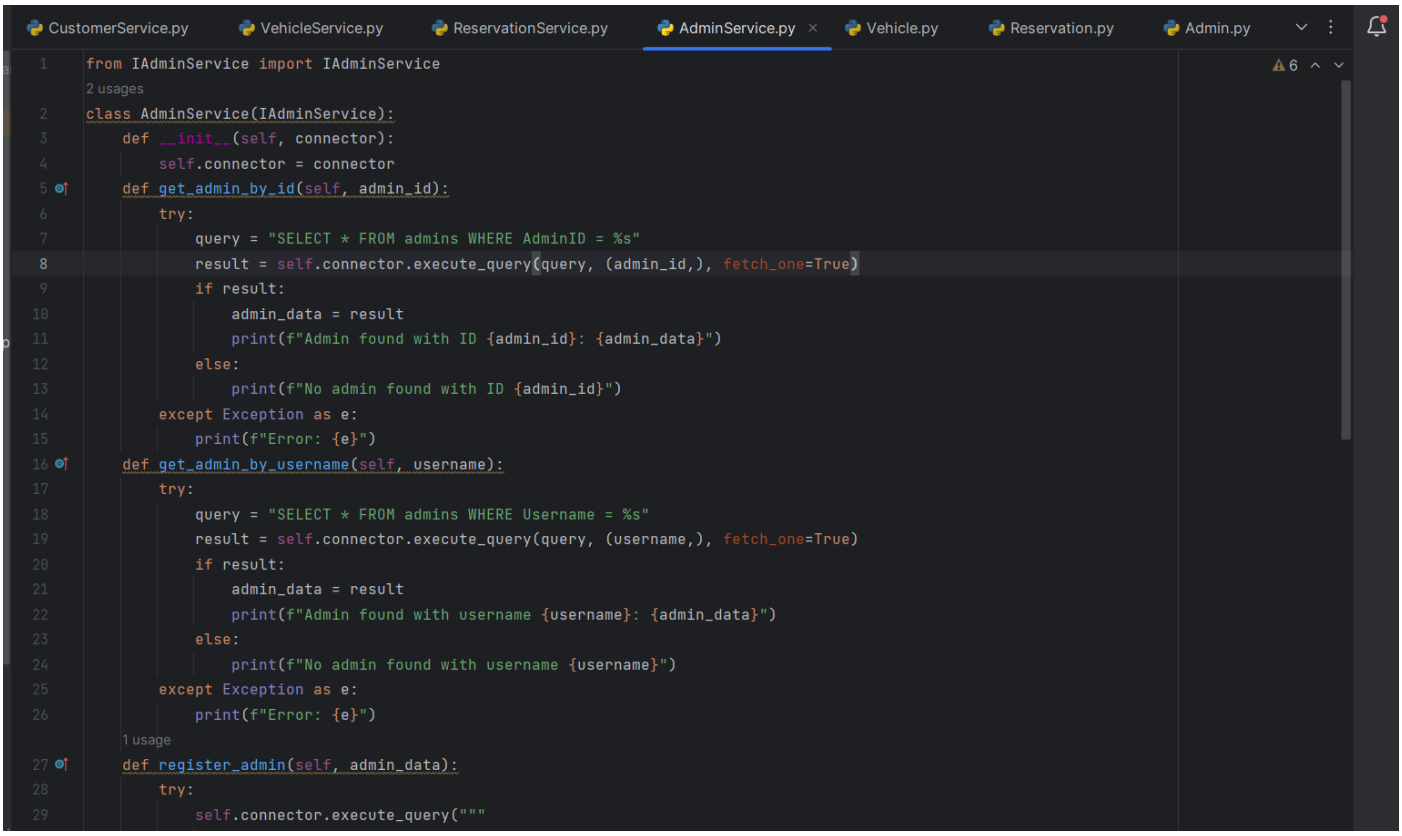
```python
from IReservationService import IReservationService
2 usages
class ReservationService(IReservationService):
    def __init__(self, connector):
        self.connector = connector
    def get_reservation_by_id(self, reservation_id):
        try:
            query = "SELECT * FROM reservations WHERE ReservationID = %s"
            result = self.connector.execute_query(query, (reservation_id,), fetch_one=True)
            if result:
                reservation_data = result
                print(f"Reservation found with ID {reservation_id}: {reservation_data}")
            else:
                print(f"No reservation found with ID {reservation_id}")
        except Exception as e:
            print(f"Error: {e}")
    def get_reservations_by_customer_id(self, customer_id):
        try:
            query = "SELECT * FROM reservations WHERE CustomerID = %s"
            result = self.connector.execute_query(query, (customer_id,), fetch_one=False)
            if result:
                customer_reservations = result
                print(f"Reservations for Customer ID {customer_id}:")
                for reservation in customer_reservations:
                    print(reservation)
            else:
                print(f"No reservations found for Customer ID {customer_id}")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def create_reservation(self, reservation_data):
```



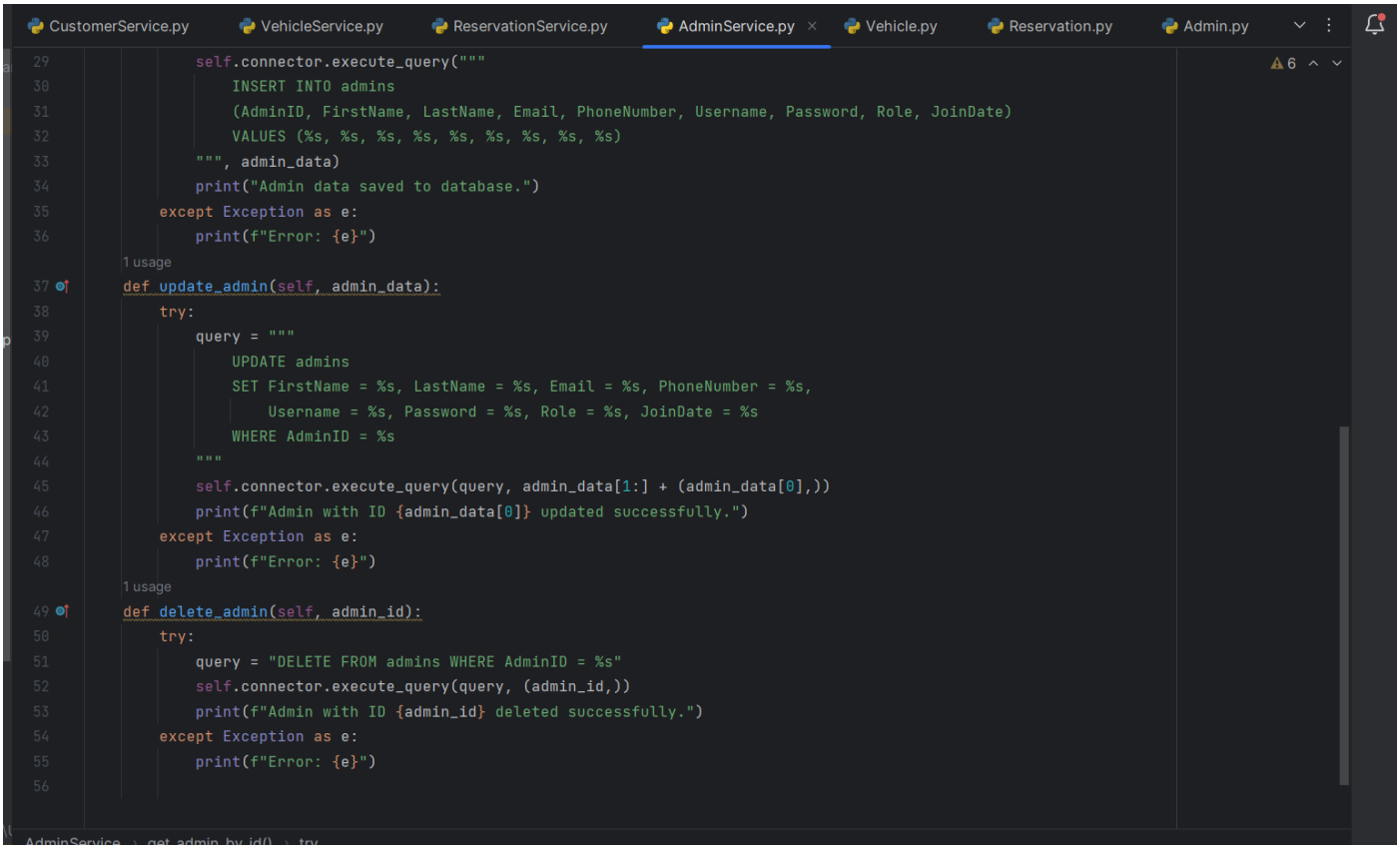
```python
    def create_reservation(self, reservation_data):
        try:
            self.connector.execute_query("""
                INSERT INTO reservations
                (ReservationID, CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, reservation_data)
            print("Reservation data saved to database.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def update_reservation(self, reservation_data):
        try:
            query = """
                UPDATE reservations
                SET CustomerID = %s, VehicleID = %s, StartDate = %s, EndDate = %s,
                    TotalCost = %s, Status = %s
                WHERE ReservationID = %s
            """
            self.connector.execute_query(query, reservation_data[1:] + (reservation_data[0],))
            print(f"Reservation with ID {reservation_data[0]} updated successfully.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def cancel_reservation(self, reservation_id):
        try:
            query = "DELETE FROM reservations WHERE ReservationID = %s"
            self.connector.execute_query(query, (reservation_id,))
            print(f"Reservation with ID {reservation_id} canceled successfully.")
        except Exception as e:
            print(f"Error: {e}")
```

AdminService (implements IAdminService):

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



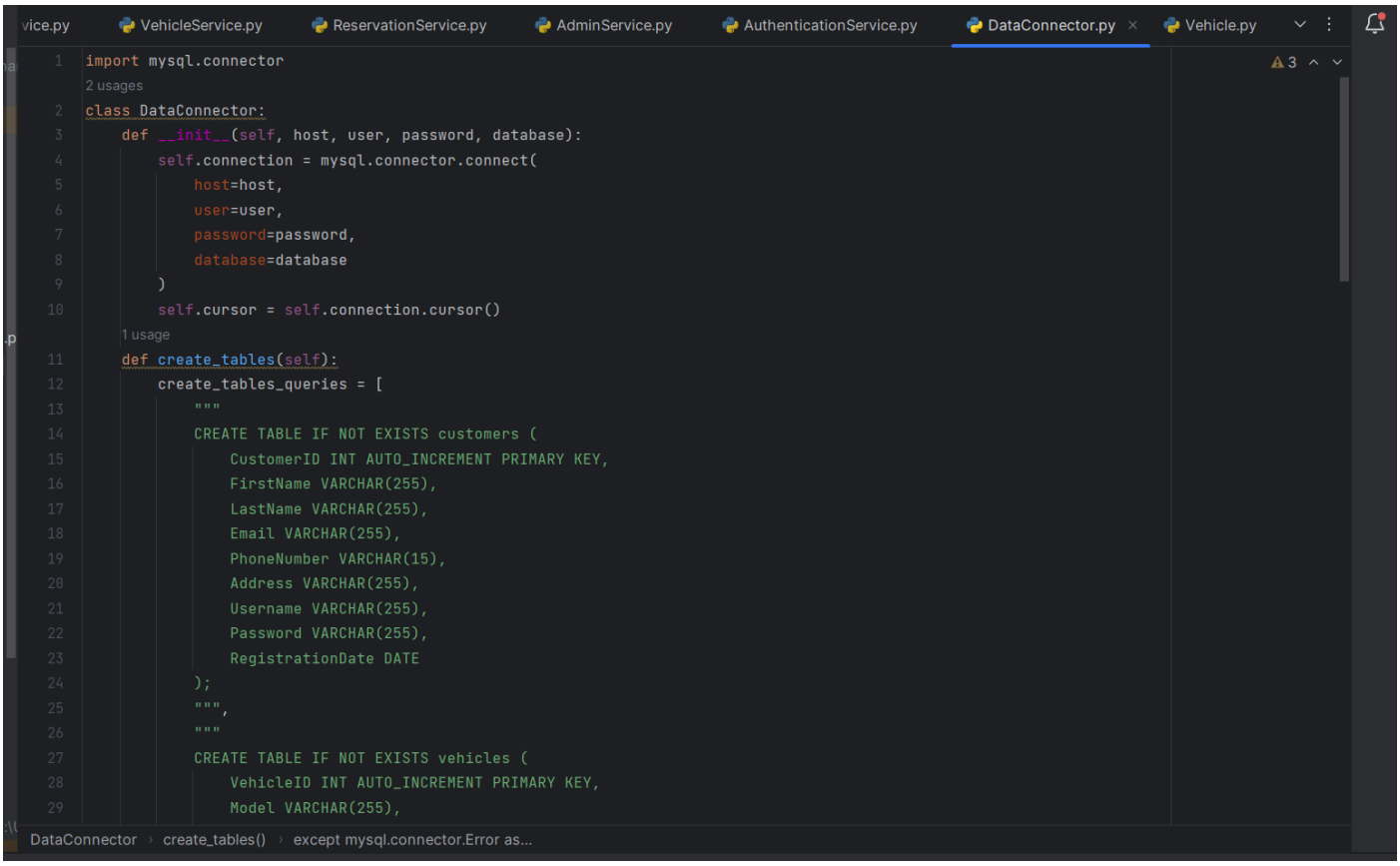
```python
from IAdminService import IAdminService
2 usages
class AdminService(IAdminService):
    def __init__(self, connector):
        self.connector = connector
    def get_admin_by_id(self, admin_id):
        try:
            query = "SELECT * FROM admins WHERE AdminID = %s"
            result = self.connector.execute_query(query, (admin_id,), fetch_one=True)
            if result:
                admin_data = result
                print(f"Admin found with ID {admin_id}: {admin_data}")
            else:
                print(f"No admin found with ID {admin_id}")
        except Exception as e:
            print(f"Error: {e}")
    def get_admin_by_username(self, username):
        try:
            query = "SELECT * FROM admins WHERE Username = %s"
            result = self.connector.execute_query(query, (username,), fetch_one=True)
            if result:
                admin_data = result
                print(f"Admin found with username {username}: {admin_data}")
            else:
                print(f"No admin found with username {username}")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def register_admin(self, admin_data):
        try:
            self.connector.execute_query("""
```



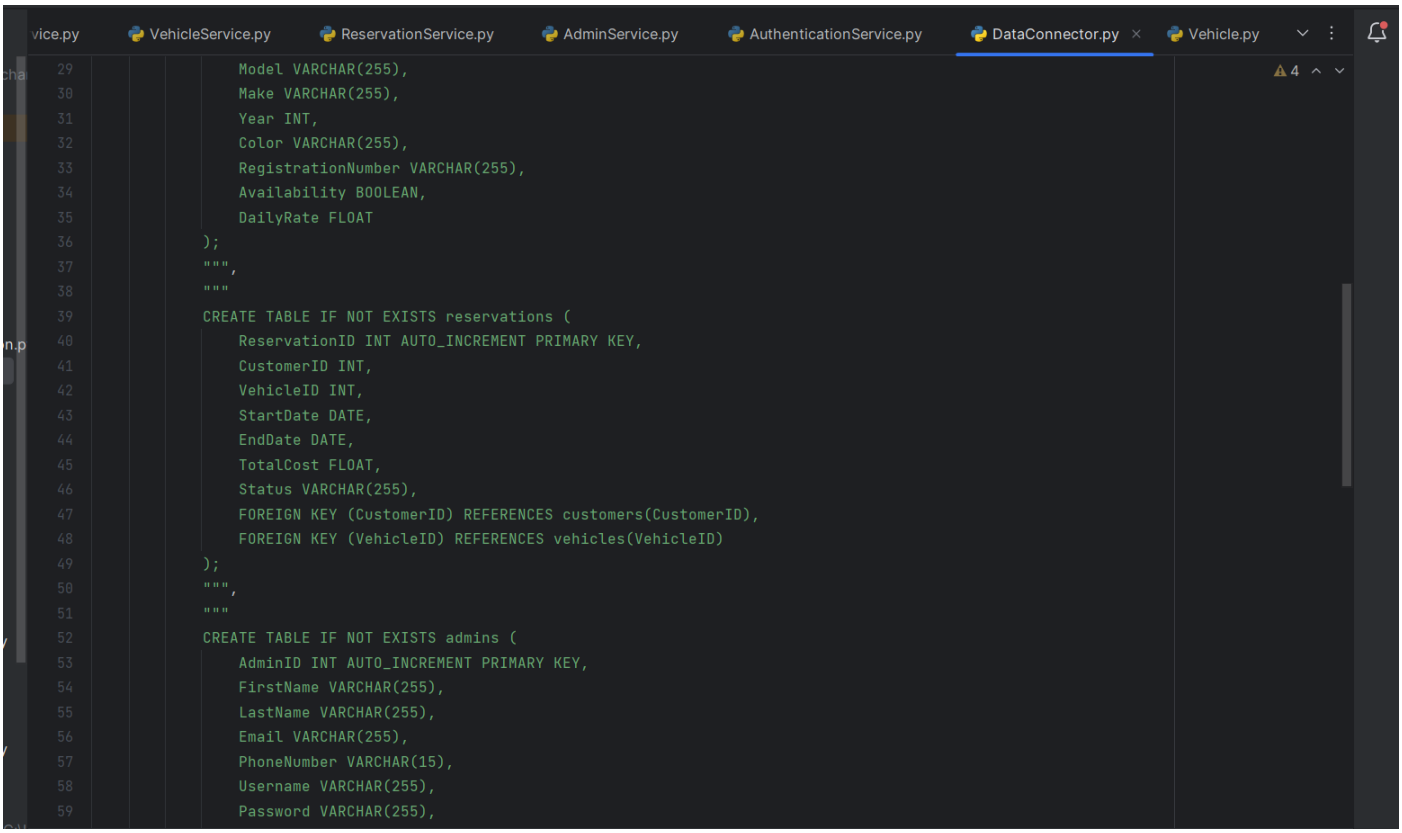
```python
                self.connector.execute_query("""
                    INSERT INTO admins
                    (AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                """, admin_data)
                print("Admin data saved to database.")
            except Exception as e:
                print(f"Error: {e}")
    1 usage
    def update_admin(self, admin_data):
        try:
            query = """
                UPDATE admins
                SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber = %s,
                    Username = %s, Password = %s, Role = %s, JoinDate = %s
                WHERE AdminID = %s
            """
            self.connector.execute_query(query, admin_data[1:] + (admin_data[0],))
            print(f"Admin with ID {admin_data[0]} updated successfully.")
        except Exception as e:
            print(f"Error: {e}")
    1 usage
    def delete_admin(self, admin_id):
        try:
            query = "DELETE FROM admins WHERE AdminID = %s"
            self.connector.execute_query(query, (admin_id,))
            print(f"Admin with ID {admin_id} deleted successfully.")
        except Exception as e:
            print(f"Error: {e}")
```

DatabaseConnector:

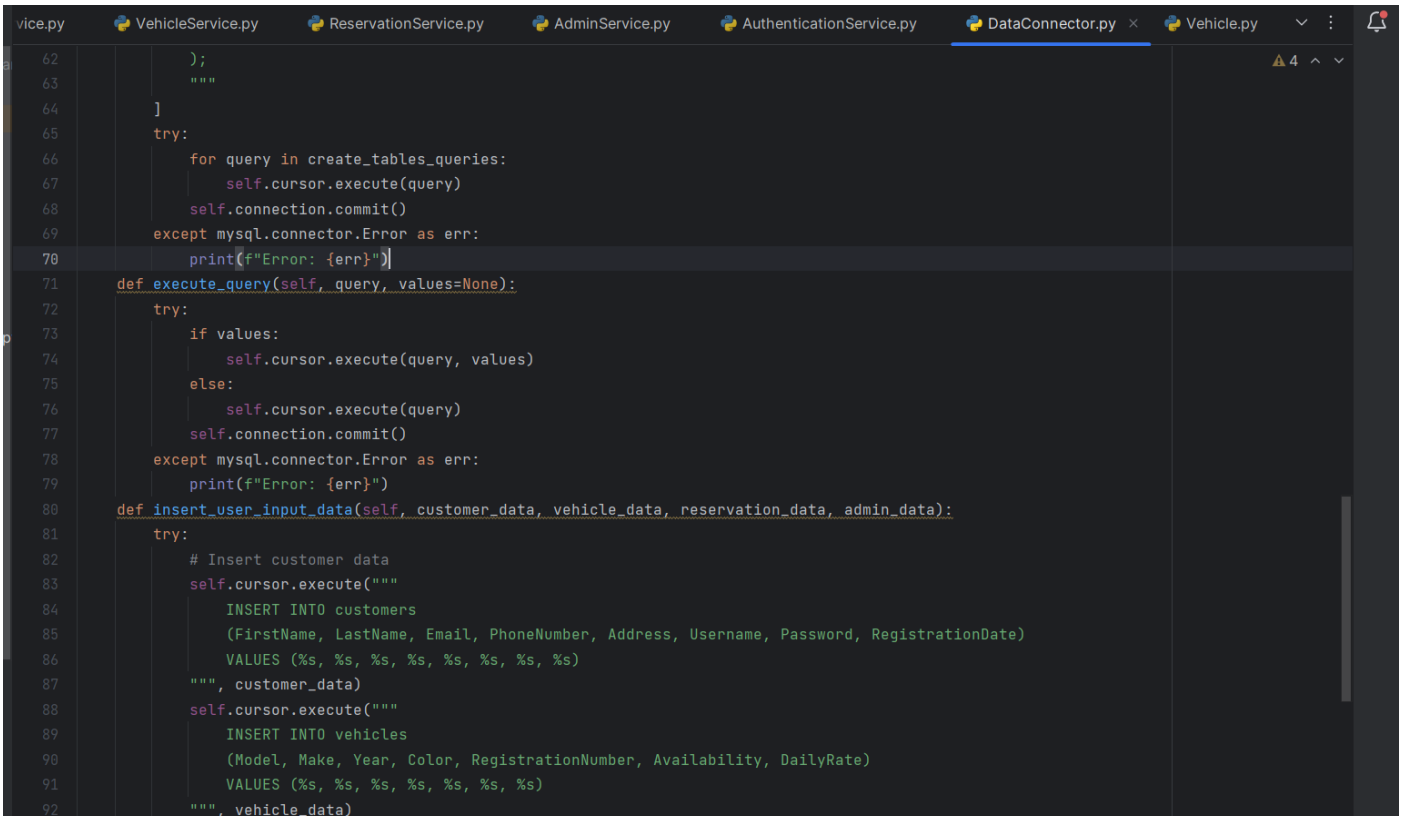
- A class responsible for handling database connections and interactions.



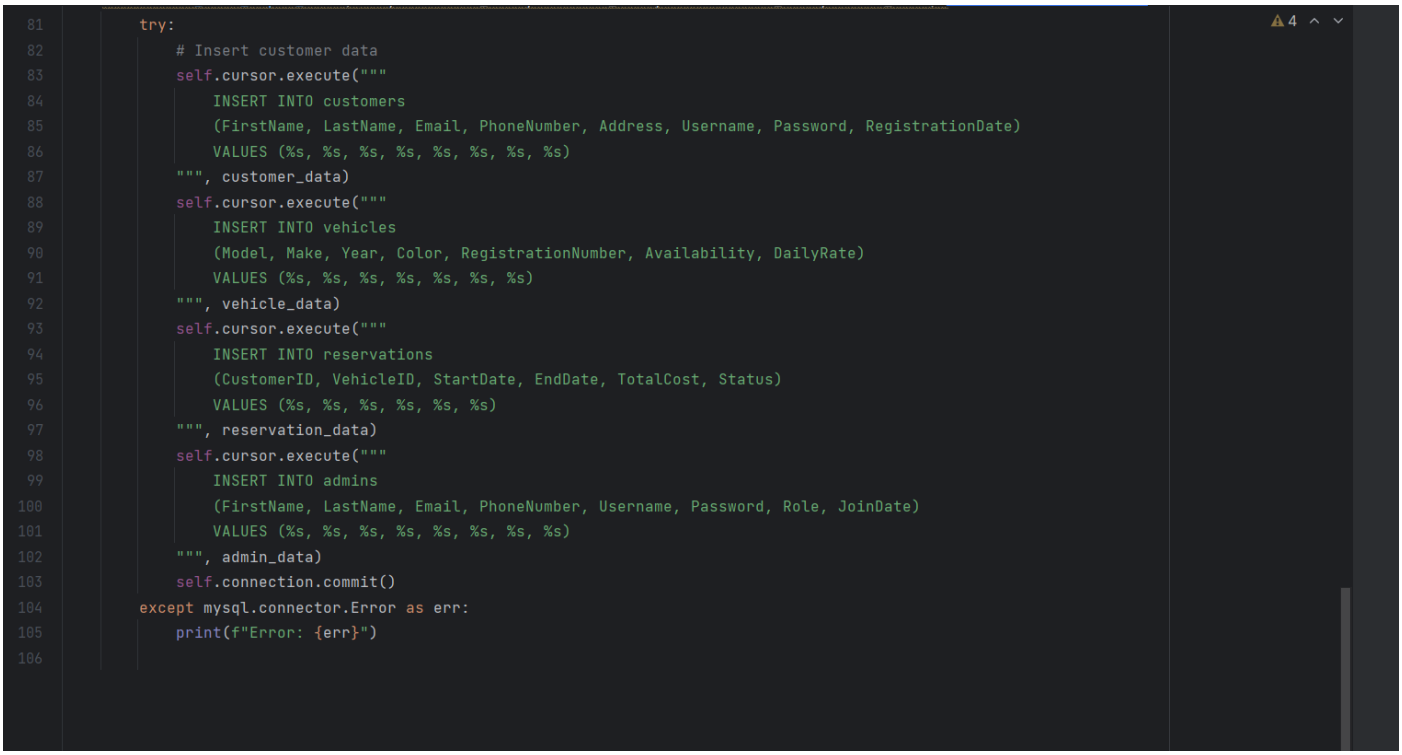
```python
import mysql.connector
2 usages
class DataConnector:
    def __init__(self, host, user, password, database):
        self.connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            database=database
        )
        self.cursor = self.connection.cursor()
    1 usage
    def create_tables(self):
        create_tables_queries = [
            """
            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
            );
            """,
            """
            CREATE TABLE IF NOT EXISTS vehicles (
                VehicleID INT AUTO_INCREMENT PRIMARY KEY,
                Model VARCHAR(255),
```



```python
                Model VARCHAR(255),
                Make VARCHAR(255),
                Year INT,
                Color VARCHAR(255),
                RegistrationNumber VARCHAR(255),
                Availability BOOLEAN,
                DailyRate FLOAT
            );
            """,
            """
            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)
            );
            """,
            """
            CREATE TABLE IF NOT EXISTS admins (
                AdminID INT AUTO_INCREMENT PRIMARY KEY,
                FirstName VARCHAR(255),
                LastName VARCHAR(255),
                Email VARCHAR(255),
                PhoneNumber VARCHAR(15),
                Username VARCHAR(255),
                Password VARCHAR(255),
```



```
vice.py  VehicleService.py  ReservationService.py  AdminService.py  AuthenticationService.py  DataConnector.py  Vehicle.py
                );
                """
            ]
            try:
                for query in create_tables_queries:
                    self.cursor.execute(query)
                self.connection.commit()
            except mysql.connector.Error as err:
                print(f"Error: {err}")
    def execute_query(self, query, values=None):
            try:
                if values:
                    self.cursor.execute(query, values)
                else:
                    self.cursor.execute(query)
                self.connection.commit()
            except mysql.connector.Error as err:
                print(f"Error: {err}")
    def insert_user_input_data(self, customer_data, vehicle_data, reservation_data, admin_data):
            try:
                # Insert customer data
                self.cursor.execute("""
                    INSERT INTO customers
                    (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """, customer_data)
                self.cursor.execute("""
                    INSERT INTO vehicles
                    (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, vehicle_data)
```



```
        try:
            # Insert customer data
            self.cursor.execute("""
                INSERT INTO customers
                (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, customer_data)
            self.cursor.execute("""
                INSERT INTO vehicles
                (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, vehicle_data)
            self.cursor.execute("""
                INSERT INTO reservations
                (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, reservation_data)
            self.cursor.execute("""
                INSERT INTO admins
                (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, admin_data)
            self.connection.commit()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
```

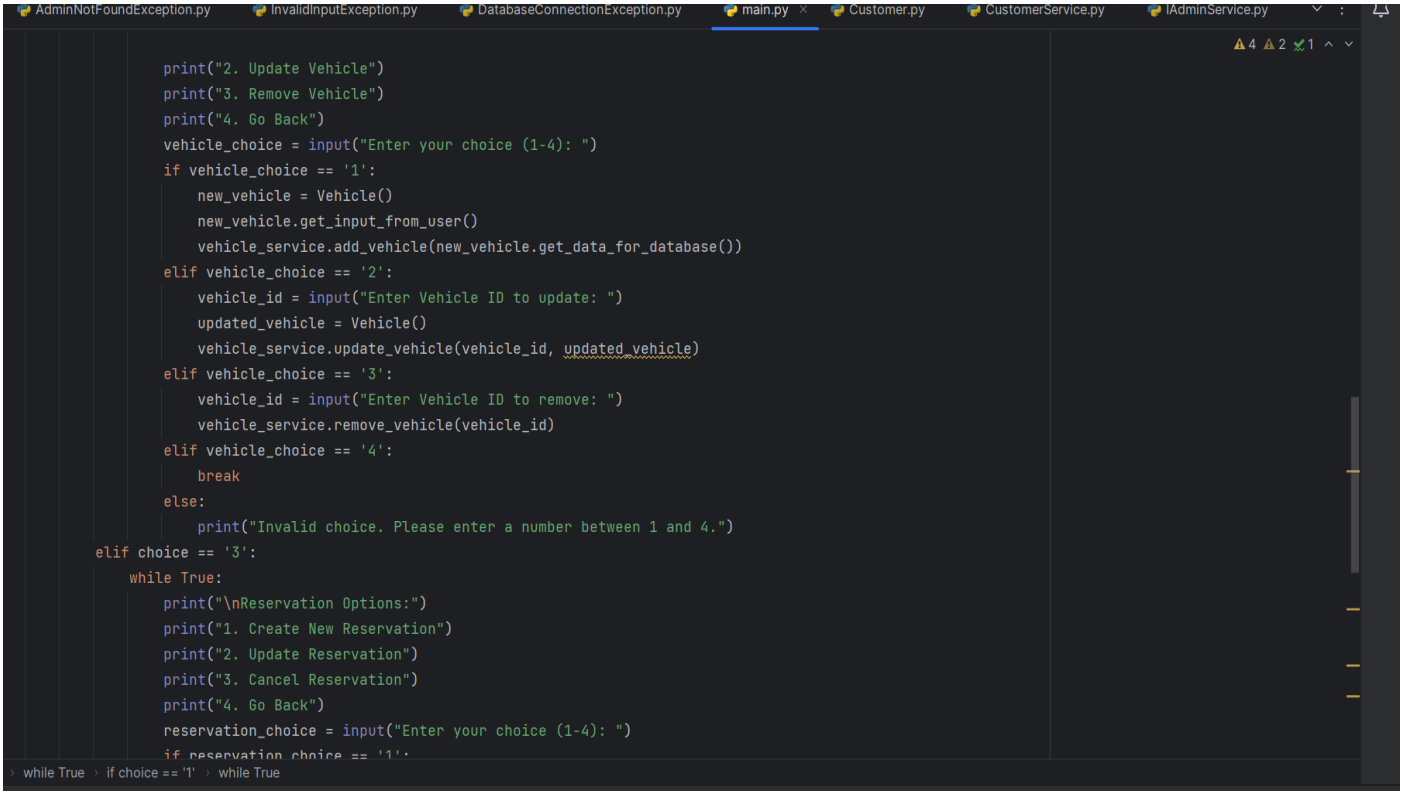
Main.py Class

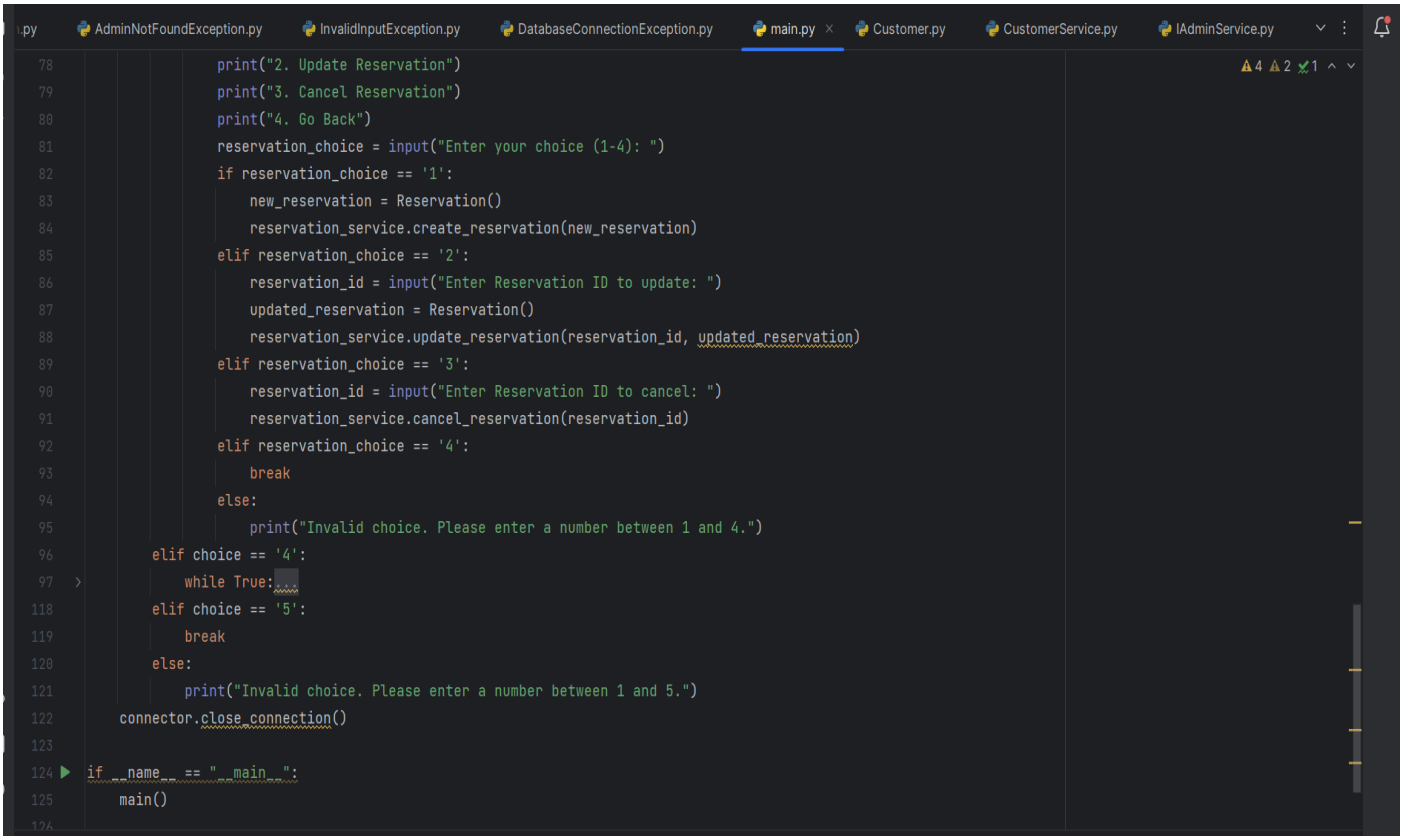
AdminNotFoundException.py | InvalidInputException.py | DatabaseConnectionException.py | main.py | Customer.py | CustomerService.py | IAdminService.py

```python
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':
```

.py | AdminNotFoundException.py | InvalidInputException.py | DatabaseConnectionException.py | main.py | Customer.py | CustomerService.py | IAdminService.py

```python
        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")
```

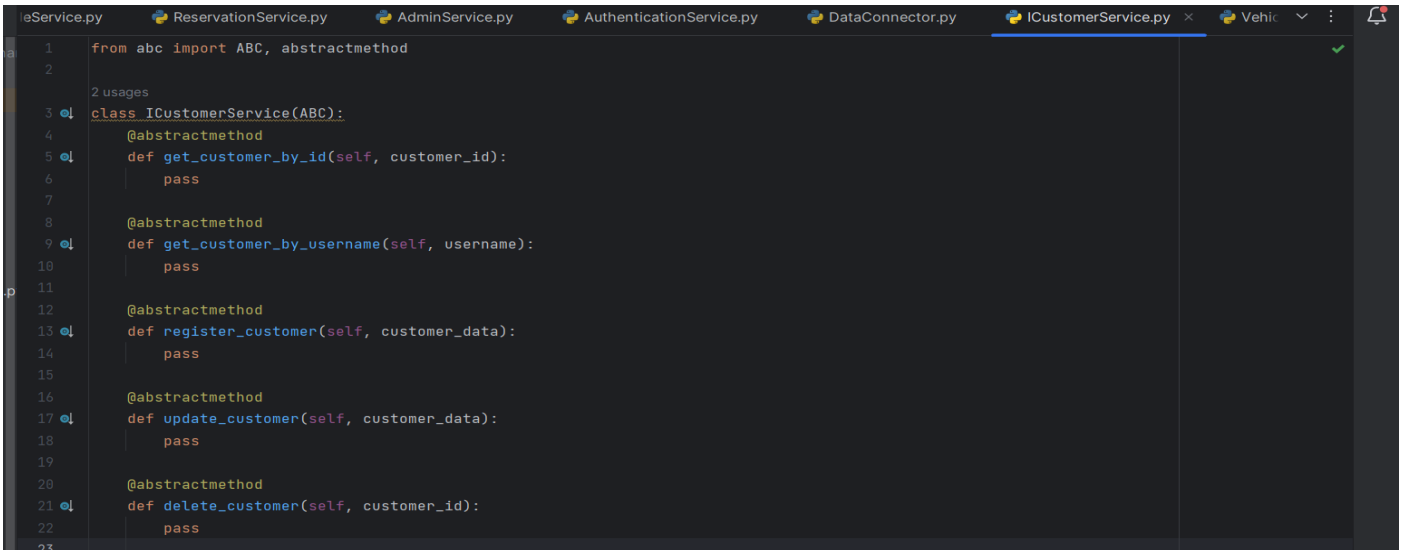

AdminNotFoundException.py
InvalidInputException.py
DatabaseConnectionException.py
main.py
Customer.py
CustomerService.py
IAdminService.py
print("2. Update Vehicle")
print("3. Remove Vehicle")
print("4. Go Back")
vehicle_choice = input("Enter your choice (1-4): ")
if vehicle_choice == '1':
new_vehicle = 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): ")
while True
if choice == '1'
while True


AdminNotFoundException.py
InvalidInputException.py
DatabaseConnectionException.py
main.py
Customer.py
CustomerService.py
IAdminService.py
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()

Interfaces:

ICustomerService:

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

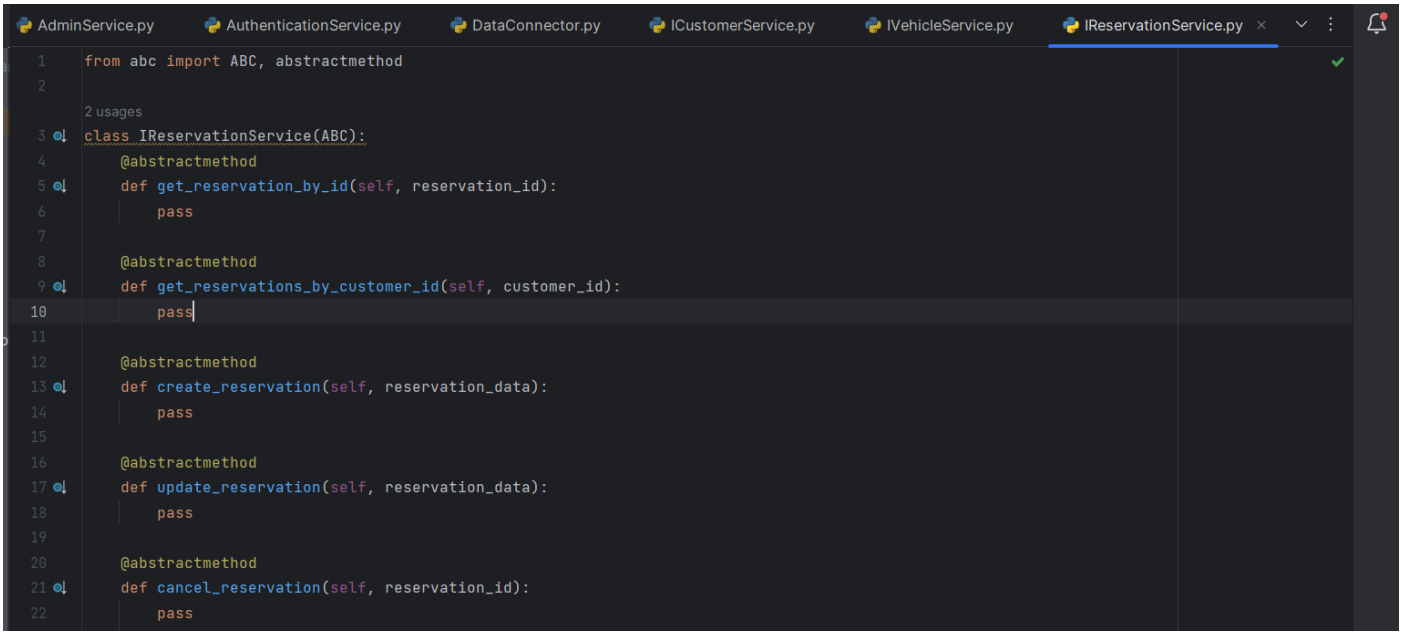


IVehicleService:

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

IReservationService:

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



IAdminService:

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

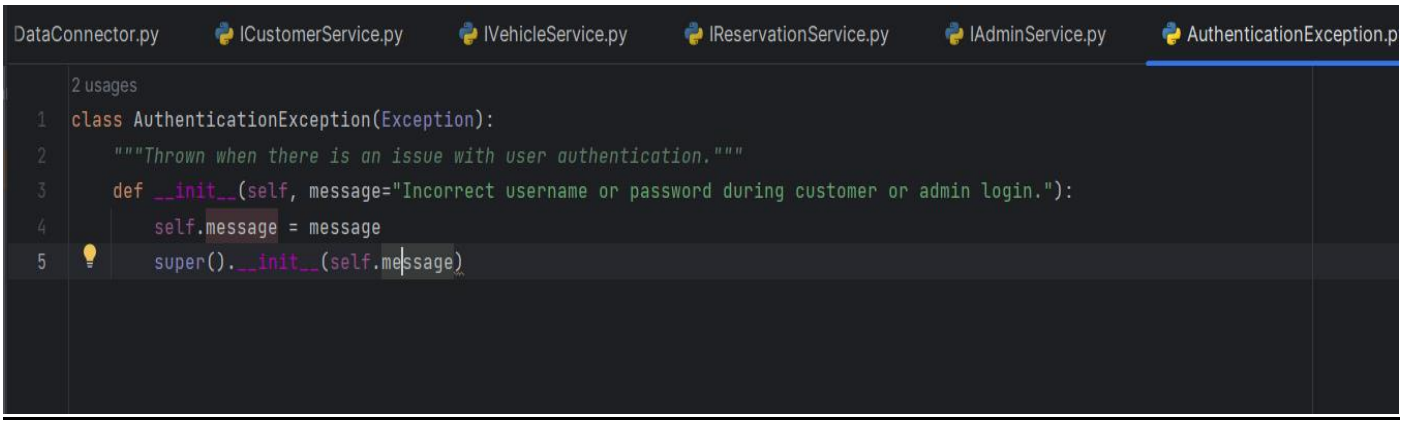
Custom Exceptions:

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

AuthenticationException:

• Thrown when there is an issue with user authentication.

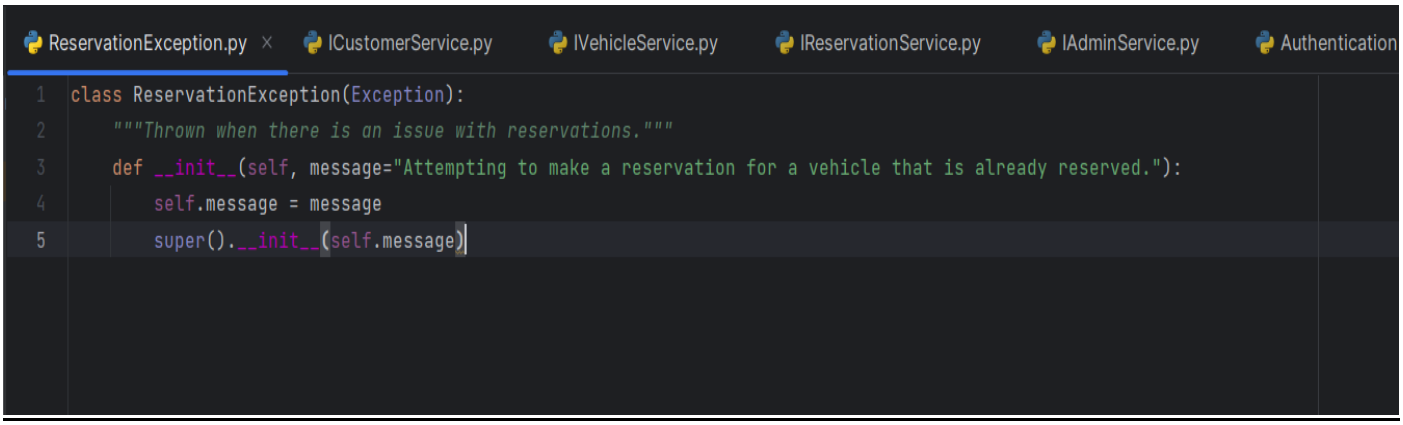
• Example Usage: Incorrect username or password during customer or admin login.



ReservationException:

• Thrown when there is an issue with reservations.

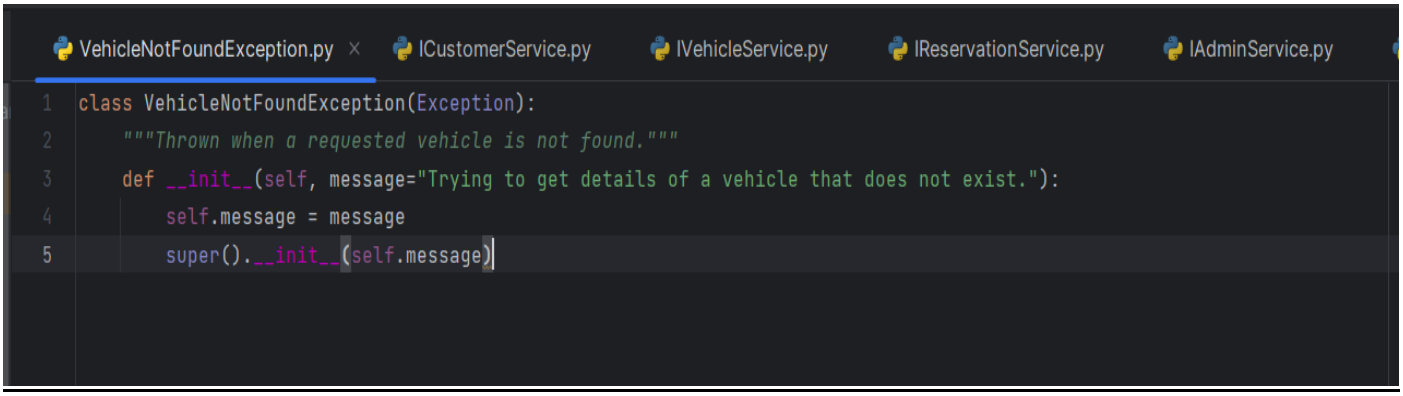
• Example Usage: Attempting to make a reservation for a vehicle that is already reserved.



VehicleNotFoundException:

• Thrown when a requested vehicle is not found.

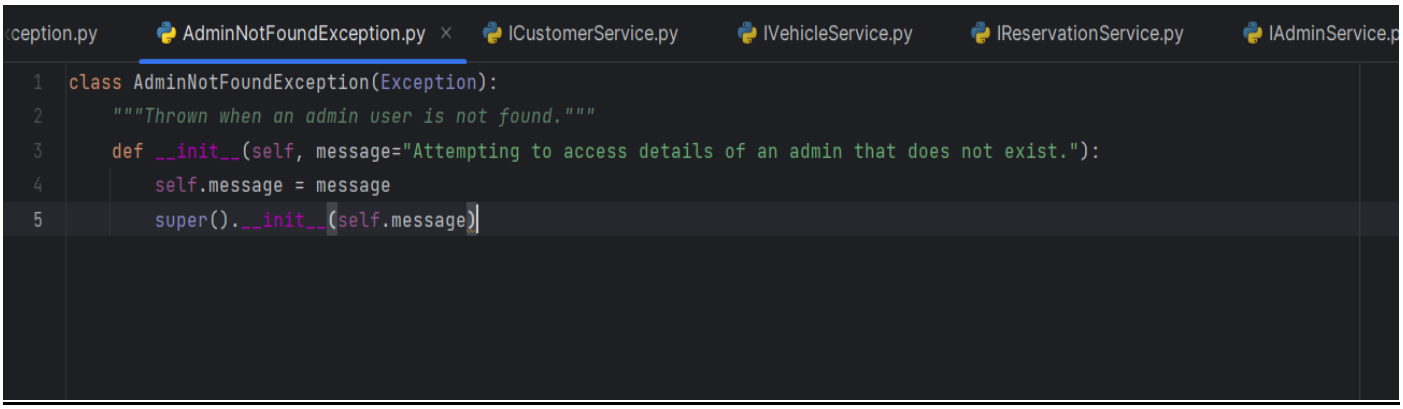
• Example Usage: Trying to get details of a vehicle that does not exist.



AdminNotFoundException:

• Thrown when an admin user is not found.

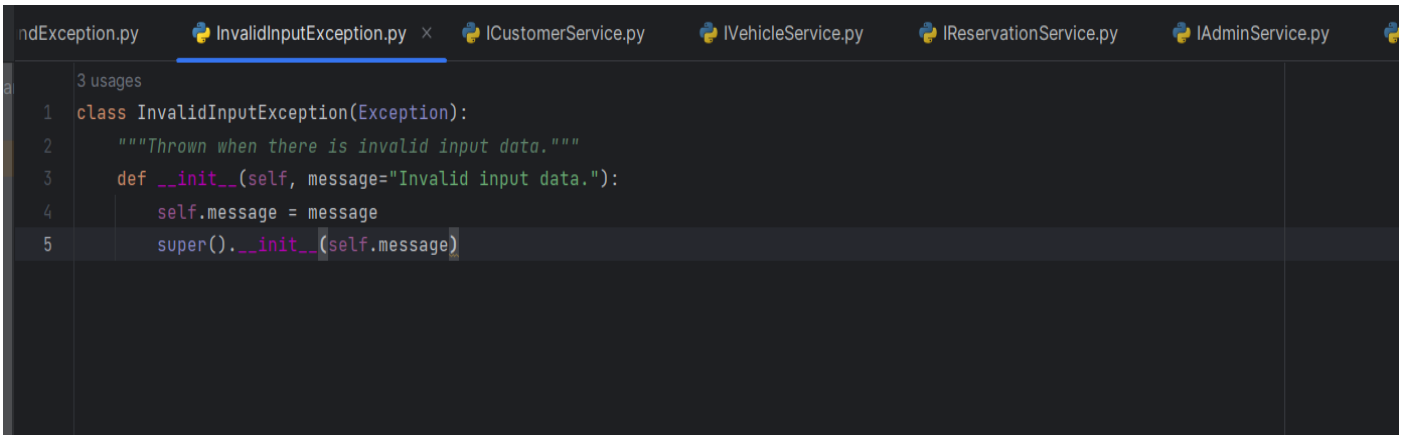
• Example Usage: Attempting to access details of an admin that does not exist.



InvalidInputException:

• Thrown when there is invalid input data.

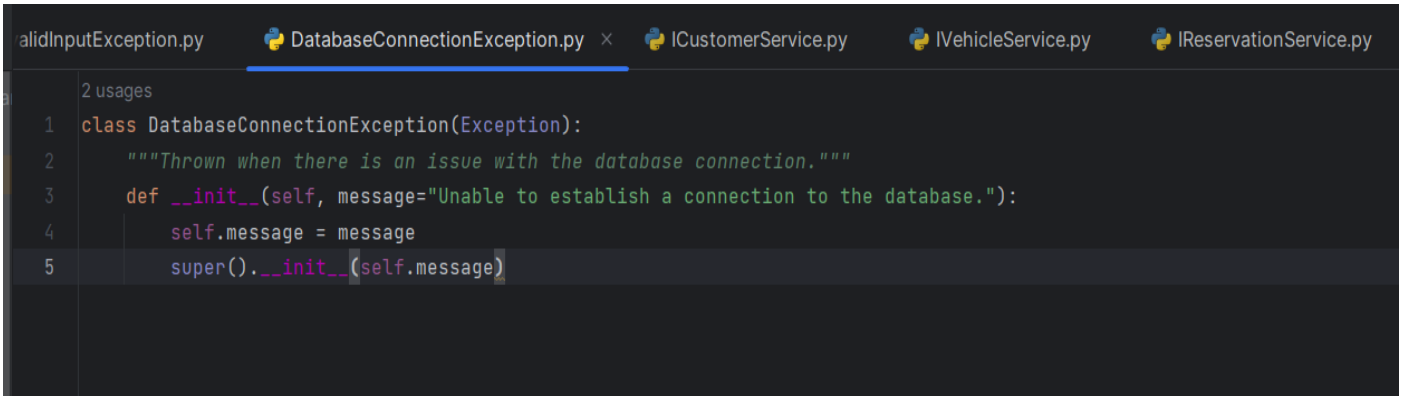
• Example Usage: When a required field is missing or has an incorrect format.



DatabaseConnectionException:

• Thrown when there is an issue with the database connection.

• Example Usage: Unable to establish a connection to the database.



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

```python
import unittest
from AuthenticationService import AuthenticationService

class TestAuthenticationService(unittest.TestCase):
    def setUp(self):

        self.auth_service = AuthenticationService()

    def test_authentication_with_invalid_credentials(self):

        invalid_username = "invalidUsername"
        invalid_password = "invalidPassword"

        result = self.auth_service.authenticate(invalid_username, invalid_password)

        self.assertFalse(result, "Authentication should fail with invalid credentials")

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

2. Test updating customer information.

main.py | test_authentication_service.py | test_CustomerService.py | test_VehicleService.py | Admin.py | AdminService.py | Res

```python
import unittest
from unittest.mock import MagicMock
from CustomerService import CustomerService, InvalidInputException, DatabaseConnectionException
class TestCustomerService(unittest.TestCase):
    def setUp(self):
        self.connector = MagicMock()
        self.customer_service = CustomerService(self.connector)

    def test_update_customer_with_valid_input(self):
        # Arrange
        customer_id = 1
        updated_customer_data = {
            "FirstName": "Sushant",
            "LastName": "Kumar",
            "Email": "sushant@gmail.com",
            "PhoneNumber": "9876543210",
            "Address": "Delhi",
            "Password": "sushant737",
            "RegistrationDate": "2024-01-11",
        }
        self.customer_service.get_customer_by_id = MagicMock(return_value={"CustomerID": customer_id})
        self.customer_service.update_customer(customer_id, updated_customer_data)

        expected_query = """
            UPDATE customers
            SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Password=?, RegistrationDate=?
            WHERE CustomerID=?
        """
        expected_params = (
            updated_customer_data["Anu"],
            updated_customer_data["Singh"],
```

```
main.py  test_authentication_service.py  test_CustomerService.py  test_VehicleService.py  Admin.py  AdminService.py  Res
                updated_customer_data["Singh"],
                updated_customer_data.get("anu@gmail.co", ""),
                updated_customer_data.get("9523456", ""),
                updated_customer_data.get("Mumbai", ""),
                updated_customer_data["anu9398"],
                updated_customer_data.get("2024-01-11", ""),
                customer_id,
            )
            self.connector.execute_query.assert_called_once_with(expected_query, expected_params)
        def test_update_customer_with_invalid_customer_id(self):
            invalid_customer_id = 999
            updated_customer_data = {
                "FirstName": "UpdatedFirstName",
                "LastName": "UpdatedLastName",
            }
            self.customer_service.get_customer_by_id = MagicMock(return_value=None)
            with self.assertRaises(InvalidInputException):
                self.customer_service.update_customer(invalid_customer_id, updated_customer_data)
        def test_update_customer_with_database_connection_error(self):
            customer_id = 1
            updated_customer_data = {
                "FirstName": "sushant",
                "LastName": "kumar",
            }
```

```
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 o
r password
========================== 3 failed, 2 passed in 0.36s ==========================
(venv) PS C:\Users\ssush\PycharmProjects\carconnect>
```

3. Test adding a new vehicle.

```
main.py  test_authentication_service.py  test_CustomerService.py  test_VehicleService.py  Admin.py  AdminService.py  ReservationService.py  VehicleService.
import unittest
from unittest.mock import MagicMock
from VehicleService import VehicleService

class TestVehicleService(unittest.TestCase):
    def setUp(self):
        self.connector = MagicMock()
        self.vehicle_service = VehicleService(self.connector)

    def test_get_available_vehicles(self):
        self.connector.execute_query.return_value = [
            (1, 'Sedan', 'Toyota', 2022, 'Blue', 'ABC123', 'Available', 50.0),
            (2, 'SUV', 'Ford', 2023, 'Red', 'XYZ789', 'Available', 60.0)
        ]
        available_vehicles = self.vehicle_service.get_available_vehicles()
        expected_query = "SELECT * FROM vehicles WHERE Availability = 'Available'"
        self.connector.execute_query.assert_called_once_with(expected_query, fetch_one=False)

        self.assertEqual(len(available_vehicles), 2, "Expected 2 available vehicles")
        self.assertEqual(available_vehicles[0]["Model"], 'Sedan', "Incorrect vehicle model")
        self.assertEqual(available_vehicles[1]["DailyRate"], 60.0, "Incorrect daily rate")

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

```
DatabaseConnectionException.p   TestVehicleService › test_add_new_vehicle_with_datab...
Terminal   Local

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 o
r 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.

main.py | test_authentication_service.py | test_CustomerService.py | test_VehicleService.py | test_update_vehicle_details.py | Admin.py

```python
import unittest
from unittest.mock import MagicMock
from VehicleService import VehicleService
from InvalidInputException import InvalidInputException
from DatabaseConnectionException import DatabaseConnectionException
class TestVehicleService(unittest.TestCase):
    def setUp(self):
        self.connector = MagicMock()
        self.vehicle_service = VehicleService(self.connector)

    def test_update_vehicle_with_valid_input(self):
        # Arrange
        vehicle_id = 1
        updated_vehicle_data = {
            "Model": "UpdatedModel",
            "Make": "UpdatedMake",
            "Year": 2023,
            "Color": "UpdatedColor",
            "RegistrationNumber": "UpdatedRegNumber",
            "Availability": "UpdatedAvailability",
            "DailyRate": 60.0,
        }
        self.vehicle_service.get_vehicle_by_id = MagicMock(return_value={"VehicleID": vehicle_id})
        self.vehicle_service.update_vehicle(vehicle_id, updated_vehicle_data)
        expected_query = """
            UPDATE vehicles
            SET Model=?, Make=?, Year=?, Color=?, RegistrationNumber=?, Availability=?, DailyRate=?
            WHERE VehicleID=?
        """
        expected_params = (
            updated_vehicle_data["SUV"],
```

TestVehicleService

.py | CRLF | UTF-8 | 4 spaces | Python 3.9 (carconnect)

main.py | test_authentication_service.py | test_CustomerService.py | test_VehicleService.py | test_update_vehicle_details.py | Admin.py

```python
            updated_vehicle_data["SUV"],
            updated_vehicle_data["RangeRover"],
            updated_vehicle_data["2023"],
            updated_vehicle_data["Black"],
            updated_vehicle_data["DL0125"],
            updated_vehicle_data["Available"],
            updated_vehicle_data["2000"],
            vehicle_id["2"],
        )
        self.connector.execute_query.assert_called_once_with(expected_query, expected_params)
    def test_update_vehicle_with_invalid_vehicle_id(self):
        invalid_vehicle_id = 999
        updated_vehicle_data = {
            "Model": "SUV",
        }
        self.vehicle_service.get_vehicle_by_id = MagicMock(return_value=None)
        with self.assertRaises(InvalidInputException):
            self.vehicle_service.update_vehicle(invalid_vehicle_id, updated_vehicle_data)

    def test_update_vehicle_with_database_connection_error(self):
        vehicle_id = 1
        updated_vehicle_data = {
            "Model": "Sedan",
        }
        self.vehicle_service.get_vehicle_by_id = MagicMock(return_value={"VehicleID": vehicle_id})
        self.connector.execute_query.side_effect = DatabaseConnectionException("Database connection error")
        with self.assertRaises(DatabaseConnectionException):
            self.vehicle_service.update_vehicle(vehicle_id, updated_vehicle_data)

if __name__ == "__main__":
    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

```
ysql> 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   Admin
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**********