# CASE STUDY :-

## CarConnect, a Car Rental Platform

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

1. ***Customer Table***:

• CustomerID (Primary Key): Unique identifier for each customer.

• FirstName: First name of the customer.

• LastName: Last name of the customer.

• Email: Email address of the customer for communication.

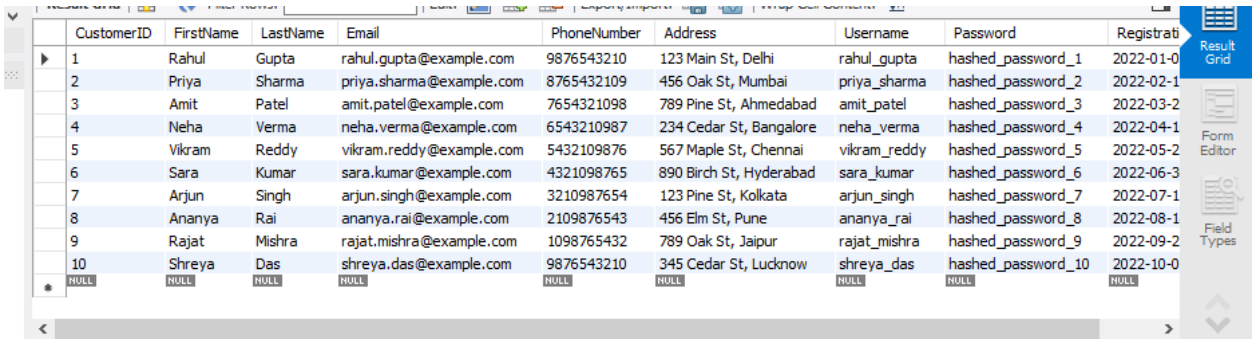
• PhoneNumber: Contact number of the customer.

• Address: Customer's residential address.

• Username: Unique username for customer login.

• Password: Securely hashed password for customer authentication.

• RegistrationDate: Date when the customer registered.



| CustomerID | FirstName | LastName | Email | PhoneNumber | Address | Username | Password | Registrati |
|---|---|---|---|---|---|---|---|---|
| 1 | Rahul | Gupta | rahul.gupta@example.com | 9876543210 | 123 Main St, Delhi | rahul_gupta | hashed_password_1 | 2022-01-0 |
| 2 | Priya | Sharma | priya.sharma@example.com | 8765432109 | 456 Oak St, Mumbai | priya_sharma | hashed_password_2 | 2022-02-1 |
| 3 | Amit | Patel | amit.patel@example.com | 7654321098 | 789 Pine St, Ahmedabad | amit_patel | hashed_password_3 | 2022-03-2 |
| 4 | Neha | Verma | neha.verma@example.com | 6543210987 | 234 Cedar St, Bangalore | neha_verma | hashed_password_4 | 2022-04-1 |
| 5 | Vikram | Reddy | vikram.reddy@example.com | 5432109876 | 567 Maple St, Chennai | vikram_reddy | hashed_password_5 | 2022-05-2 |
| 6 | Sara | Kumar | sara.kumar@example.com | 4321098765 | 890 Birch St, Hyderabad | sara_kumar | hashed_password_6 | 2022-06-3 |
| 7 | Arjun | Singh | arjun.singh@example.com | 3210987654 | 123 Pine St, Kolkata | arjun_singh | hashed_password_7 | 2022-07-1 |
| 8 | Ananya | Rai | ananya.rai@example.com | 2109876543 | 456 Elm St, Pune | ananya_rai | hashed_password_8 | 2022-08-1 |
| 9 | Rajat | Mishra | rajat.mishra@example.com | 1098765432 | 789 Oak St, Jaipur | rajat_mishra | hashed_password_9 | 2022-09-2 |
| 10 | Shreya | Das | shreya.das@example.com | 9876543210 | 345 Cedar St, Lucknow | shreya_das | hashed_password_10 | 2022-10-0 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |



***2. Vehicle Table***:

• VehicleID (Primary Key): Unique identifier for each vehicle.

• Model: Model of the vehicle.

• Make: Manufacturer or brand of the vehicle.

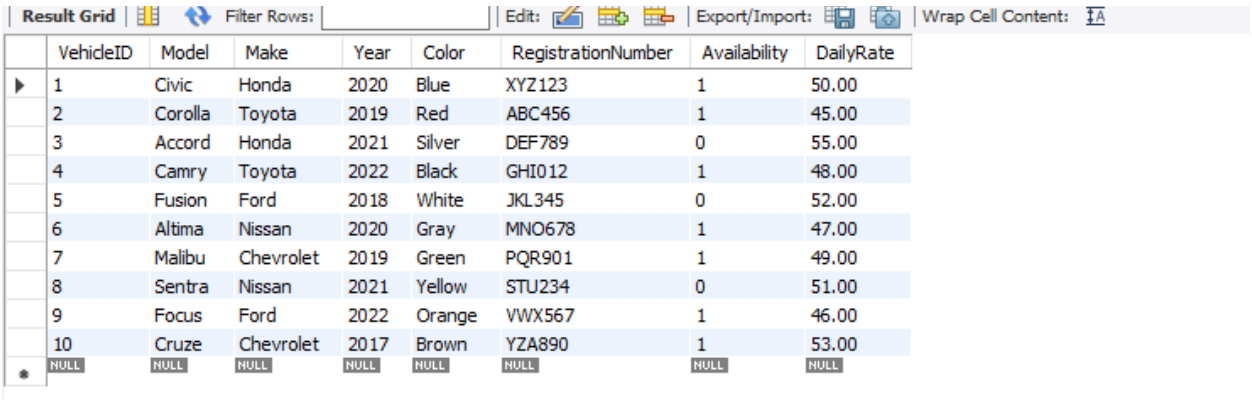
• Year: Manufacturing year of the vehicle.

• 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



| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability | DailyRate |
|---|---|---|---|---|---|---|---|
| 1 | Civic | Honda | 2020 | Blue | XYZ123 | 1 | 50.00 |
| 2 | Corolla | Toyota | 2019 | Red | ABC456 | 1 | 45.00 |
| 3 | Accord | Honda | 2021 | Silver | DEF789 | 0 | 55.00 |
| 4 | Camry | Toyota | 2022 | Black | GHI012 | 1 | 48.00 |
| 5 | Fusion | Ford | 2018 | White | JKL345 | 0 | 52.00 |
| 6 | Altima | Nissan | 2020 | Gray | MNO678 | 1 | 47.00 |
| 7 | Malibu | Chevrolet | 2019 | Green | PQR901 | 1 | 49.00 |
| 8 | Sentra | Nissan | 2021 | Yellow | STU234 | 0 | 51.00 |
| 9 | Focus | Ford | 2022 | Orange | VWX567 | 1 | 46.00 |
| 10 | Cruze | Chevrolet | 2017 | Brown | YZA890 | 1 | 53.00 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

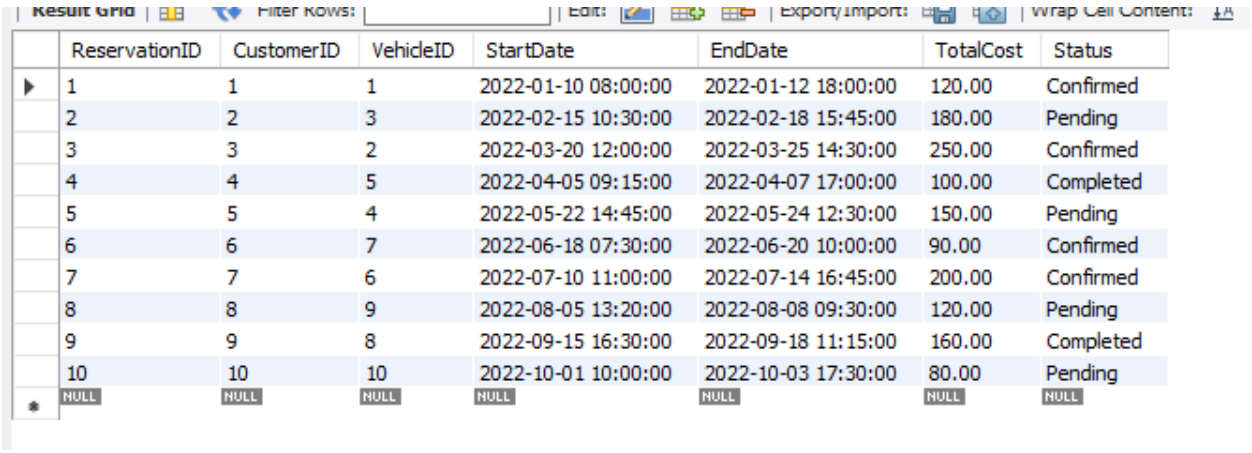
***3. Reservation Table***:

• ReservationID (Primary Key): Unique identifier for each reservation.

• CustomerID (Foreign Key): Foreign key referencing the Customer table.

• VehicleID (Foreign Key): Foreign key referencing the Vehicle table.

• StartDate: Date and time of the reservation start.



| ReservationID | CustomerID | VehicleID | StartDate | EndDate | TotalCost | Status |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2022-01-10 08:00:00 | 2022-01-12 18:00:00 | 120.00 | Confirmed |
| 2 | 2 | 3 | 2022-02-15 10:30:00 | 2022-02-18 15:45:00 | 180.00 | Pending |
| 3 | 3 | 2 | 2022-03-20 12:00:00 | 2022-03-25 14:30:00 | 250.00 | Confirmed |
| 4 | 4 | 5 | 2022-04-05 09:15:00 | 2022-04-07 17:00:00 | 100.00 | Completed |
| 5 | 5 | 4 | 2022-05-22 14:45:00 | 2022-05-24 12:30:00 | 150.00 | Pending |
| 6 | 6 | 7 | 2022-06-18 07:30:00 | 2022-06-20 10:00:00 | 90.00 | Confirmed |
| 7 | 7 | 6 | 2022-07-10 11:00:00 | 2022-07-14 16:45:00 | 200.00 | Confirmed |
| 8 | 8 | 9 | 2022-08-05 13:20:00 | 2022-08-08 09:30:00 | 120.00 | Pending |
| 9 | 9 | 8 | 2022-09-15 16:30:00 | 2022-09-18 11:15:00 | 160.00 | Completed |
| 10 | 10 | 10 | 2022-10-01 10:00:00 | 2022-10-03 17:30:00 | 80.00 | Pending |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

• EndDate: Date and time of the reservation end.

• TotalCost: Total cost of the reservation.

• Status: Current status of the reservation (e.g., pending, confirmed, completed)

***4. Admin Table:***

• AdminID (Primary Key): Unique identifier for each admin.

• FirstName: First name of the admin.

• LastName: Last name of the admin.

• Email: Email address of the admin for communication.

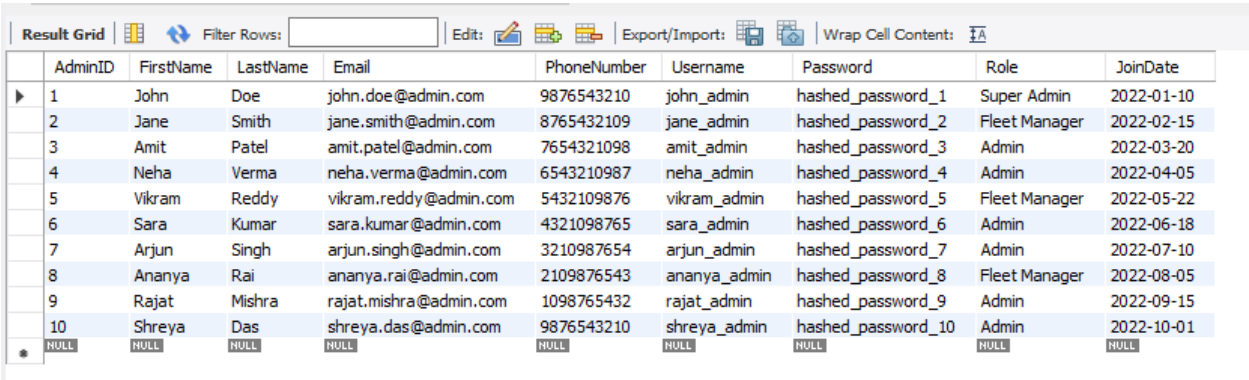
• PhoneNumber: Contact number of the admin.

• Username: Unique username for admin login.

• Password: Securely hashed password for admin authentication.

• Role: Role of the admin within the system (e.g., super admin, fleet manager).

• JoinDate: Date when the admin joined the system.




| AdminID | FirstName | LastName | Email | PhoneNumber | Username | Password | Role | JoinDate |
|---|---|---|---|---|---|---|---|---|
| 1 | John | Doe | john.doe@admin.com | 9876543210 | john_admin | hashed_password_1 | Super Admin | 2022-01-10 |
| 2 | Jane | Smith | jane.smith@admin.com | 8765432109 | jane_admin | hashed_password_2 | Fleet Manager | 2022-02-15 |
| 3 | Amit | Patel | amit.patel@admin.com | 7654321098 | amit_admin | hashed_password_3 | Admin | 2022-03-20 |
| 4 | Neha | Verma | neha.verma@admin.com | 6543210987 | neha_admin | hashed_password_4 | Admin | 2022-04-05 |
| 5 | Vikram | Reddy | vikram.reddy@admin.com | 5432109876 | vikram_admin | hashed_password_5 | Fleet Manager | 2022-05-22 |
| 6 | Sara | Kumar | sara.kumar@admin.com | 4321098765 | sara_admin | hashed_password_6 | Admin | 2022-06-18 |
| 7 | Arjun | Singh | arjun.singh@admin.com | 3210987654 | arjun_admin | hashed_password_7 | Admin | 2022-07-10 |
| 8 | Ananya | Rai | ananya.rai@admin.com | 2109876543 | ananya_admin | hashed_password_8 | Fleet Manager | 2022-08-05 |
| 9 | Rajat | Mishra | rajat.mishra@admin.com | 1098765432 | rajat_admin | hashed_password_9 | Admin | 2022-09-15 |
| 10 | Shreya | Das | shreya.das@admin.com | 9876543210 | shreya_admin | hashed_password_10 | Admin | 2022-10-01 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

-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

```python
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address, username, password,
                 registration_date):
        self._customer_id = customer_id
        self._first_name = first_name
        self._last_name = last_name
        self._email = email
        self._phone_number = phone_number
        self._address = address
        self._username = username
        self._password = password
        self._registration_date = registration_date

    def get_customer_id(self):
        return self._customer_id

    def get_first_name(self):
        return self._first_name

    def get_last_name(self):
        return self._last_name

    def get_email(self):
        return self._email

    def get_phone_number(self):
```

```python
    def get_password(self):
        return self._password

    def get_registration_date(self):
        return self._registration_date

    def set_customer_id(self, customer_id):
        if isinstance(customer_id, int) and customer_id > 0:
            self._customer_id = customer_id
        else:
            print("Invalid customer_id. It should be a positive integer.")

    def set_first_name(self, first_name):
        if first_name:
            self._first_name = first_name
        else:
            print("Invalid first_name. It should not be empty.")

    def set_last_name(self, last_name):
        if last_name:
            self._last_name = last_name
        else:
            print("Invalid last_name. It should not be empty.")

    def set_email(self, email):
```

```python
    def set_phone_number(self, phone_number):
        if phone_number and re.match( pattern: r"\d{10}", phone_number):
            self._phone_number = phone_number
        else:
            print("Invalid phone number format.")

    def set_address(self, address):
        if address:
            self._address = address
        else:
            print("Invalid address. It should not be empty.")

    def set_username(self, username):
        if username:
            self._username = username
        else:
            print("Invalid username. It should not be empty.")

    def set_password(self, password):
        if password:
            self._password = password
        else:
            print("Invalid password. It should not be empty.")

    def set_registration_date(self, registration_date):
```

• Methods: Authenticate(password)

```
    1 usage
    def authenticate(self, input_password):
        return input_password == self._password

    1 usage
    @classmethod
    def create_from_input(cls):
        customer_id = int(input("Enter CustomerID: "))
        first_name = input("Enter First Name: ")
        last_name = input("Enter Last Name: ")
        email = input("Enter Email: ")
        phone_number = input("Enter Phone Number: ")
        address = input("Enter Address: ")
        username = input("Enter Username: ")
        password = input("Enter Password: ")
        registration_date = input("Enter Registration Date (YYYY-MM-DD): ")
        print("Customer Created Succesfully ")
        return cls(customer_id, first_name, last_name, email, phone_number, address, username, password,
                   registration_date)


customer_instance = Customer.create_from_input()
print("Authentication method")
input_password = input("Enter your password: ")
```

***OUTPUT:-***

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\entity\Customer.py
Enter CustomerID: 101
Enter First Name: Aniket
Enter Last Name: Biyani
Enter Email: anixbiyani11@gmail.com
Enter Phone Number: 7020905391
Enter Address: Pune
Enter Username: aniket
Enter Password: Aniket@123
Enter Registration Date (YYYY-MM-DD): 2023-13-30
Customer Created Succesfully
Authentication method
Enter your password: Aniket@123
Authentication successful!

Process finished with exit code 0
```

• ***Vehicle***

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

```
class Vehicle:
    def __init__(self, vehicle_id, model, make, year, color, registration_number, availability, daily_rate):
        self._vehicle_id = vehicle_id
        self._model = model
        self._make = make
        self._year = year
        self._color = color
        self._registration_number = registration_number
        self._availability = availability
        self._daily_rate = daily_rate

    def get_vehicle_id(self):
        return self._vehicle_id

    def get_model(self):
        return self._model

    def get_make(self):
        return self._make

    def get_year(self):
        return self._year

    def get_color(self):
        return self._color
```

```
    def set_vehicle_id(self, vehicle_id):
        if isinstance(vehicle_id, int) and vehicle_id > 0:
            self._vehicle_id = vehicle_id
        else:
            print("Invalid vehicle_id. It should be a positive integer.")

    def set_model(self, model):
        if model:
            self._model = model
        else:
            print("Invalid model. It should not be empty.")

    def set_make(self, make):
        if make:
            self._make = make
        else:
            print("Invalid make. It should not be empty.")

    def set_year(self, year):
        if isinstance(year, int) and year > 0:
            self._year = year
        else:
            print("Invalid year. It should be a positive integer.")
```

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

```
class Reservation:
    def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status):
        self._reservation_id = reservation_id
        self._customer_id = customer_id
        self._vehicle_id = vehicle_id
        self._start_date = start_date
        self._end_date = end_date
        self._total_cost = total_cost
        self._status = status

    def get_reservation_id(self):
        return self._reservation_id

    def get_customer_id(self):
        return self._customer_id

    def get_vehicle_id(self):
        return self._vehicle_id

    def get_start_date(self):
        return self._start_date

    def get_end_date(self):
        return self._end_date

    def get_total_cost(self):
        return self._total_cost
```

```
    def set_reservation_id(self, reservation_id):
        if isinstance(reservation_id, int) and reservation_id > 0:
            self._reservation_id = reservation_id
        else:
            print("Invalid reservation_id. It should be a positive integer.")

    def set_customer_id(self, customer_id):
        if isinstance(customer_id, int) and customer_id > 0:
            self._customer_id = customer_id
        else:
            print("Invalid customer_id. It should be a positive integer.")

    def set_vehicle_id(self, vehicle_id):
        if isinstance(vehicle_id, int) and vehicle_id > 0:
            self._vehicle_id = vehicle_id
        else:
            print("Invalid vehicle_id. It should be a positive integer.")

    def set_start_date(self, start_date):
        if isinstance(start_date, str):
            self._start_date = start_date
        else:
            print("Invalid start_date. It should be a valid date format.")
```

- ***Methods: CalculateTotalCost()***

```
    1 usage
    def CalculateTotalCost(self):
        start_date = datetime.strptime(self._start_date, __format: "%Y-%m-%d")
        end_date = datetime.strptime(self._end_date, __format: "%Y-%m-%d")
        days_rented = (end_date - start_date).days
        daily_rate = 50
        self._total_cost = days_rented * daily_rate

    1 usage
    @classmethod
    def create_from_input(cls):
        reservation_id = int(input("Enter ReservationID: "))
        customer_id = int(input("Enter CustomerID: "))
        vehicle_id = int(input("Enter VehicleID: "))
        start_date = input("Enter Start Date (YYYY-MM-DD): ")
        end_date = input("Enter End Date (YYYY-MM-DD): ")
        total_cost = 0
        status = input("Enter Status: ")
        print("Reservation done succesfully ")
        return cls(reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status)


reservation_instance = Reservation.create_from_input()
reservation_instance.CalculateTotalCost()
print("Total Cost:", reservation_instance._total_cost)
```

***OUTPUT:-***

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\entity\Reservation.py
Enter ReservationID: 1
Enter CustomerID: 11
Enter VehicleID: 21
Enter Start Date (YYYY-MM-DD): 2023-12-21
Enter End Date (YYYY-MM-DD): 2023-12-31
Enter Status: Completed
Reservation done succesfully
Total Cost: 500

Process finished with exit code 0
```

- ***Admin***

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

```
class Admin:
    def __init__(self, admin_id, first_name, last_name, email, phone_number, username, password, role, join_date):
        self._admin_id = admin_id
        self._first_name = first_name
        self._last_name = last_name
        self._email = email
        self._phone_number = phone_number
        self._username = username
        self._password = password
        self._role = role
        self._join_date = join_date

    def get_admin_id(self):
        return self._admin_id

    def get_first_name(self):
        return self._first_name

    def get_last_name(self):
        return self._last_name

    def get_email(self):
        return self._email

    def get_phone_number(self):
        return self._phone_number
```

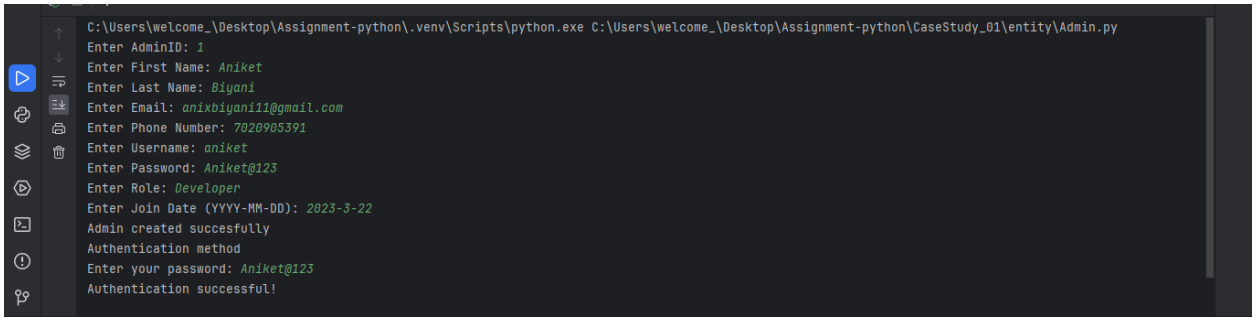
```python
    def get_password(self):
        return self._password

    def get_role(self):
        return self._role

    def get_join_date(self):
        return self._join_date

    def set_admin_id(self, admin_id):
        if isinstance(admin_id, int) and admin_id > 0:
            self._admin_id = admin_id
        else:
            print("Invalid admin_id. It should be a positive integer.")

    def set_first_name(self, first_name):
        if first_name:
            self._first_name = first_name
        else:
            print("Invalid first_name. It should not be empty.")

    def set_last_name(self, last_name):
        if last_name:
            self._last_name = last_name
        else:
            print("Invalid last_name. It should not be empty.")
```

```python
    def set_email(self, email):
        if email and re.match( pattern: r"[^@]+@[^@]+\.[^@]+", email):
            self._email = email
        else:
            print("Invalid email format.")

    def set_phone_number(self, phone_number):
        if phone_number and re.match( pattern: r"\d{10}", phone_number):
            self._phone_number = phone_number
        else:
            print("Invalid phone number format.")

    def set_username(self, username):
        if username:
            self._username = username
        else:
            print("Invalid username. It should not be empty.")

    def set_password(self, password):
        if password:
            self._password = password
        else:
            print("Invalid password. It should not be empty.")
```

- Methods: Authenticate(password)

```python
    1 usage
    def Authenticate(self, input_password):
            return input_password == self._password

    1 usage
    @classmethod
    def create_from_input(cls):
        admin_id = int(input("Enter AdminID: "))
        first_name = input("Enter First Name: ")
        last_name = input("Enter Last Name: ")
        email = input("Enter Email: ")
        phone_number = input("Enter Phone Number: ")
        username = input("Enter Username: ")
        password = input("Enter Password: ")
        role = input("Enter Role: ")
        join_date = input("Enter Join Date (YYYY-MM-DD): ")
        print("Admin created succesfully")
        return cls(admin_id, first_name, last_name, email, phone_number, username, password, role, join_date)

admin_instance = Admin.create_from_input()
print("Authentication method")
input_password = input("Enter your password: ")

if admin_instance.Authenticate(input_password):
    print("Authentication successful!")
else:
    print("Authentication failed. Incorrect password.")
```

***OUTPUT:-***




***Interfaces:***

- **ICustomerService:**

- GetCustomerById(customerId)

- GetCustomerByUsername(username)

- RegisterCustomer(customerData)

- UpdateCustomer(customerData)

- DeleteCustomer(customerId)

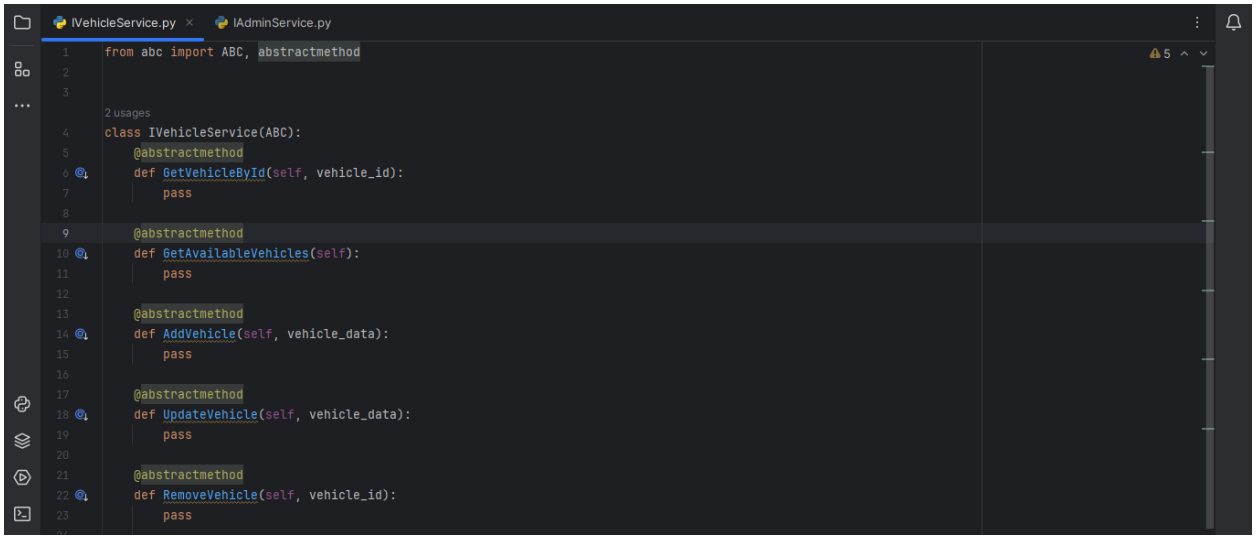

- **IVehicleService:**

- GetVehicleById(vehicleId)

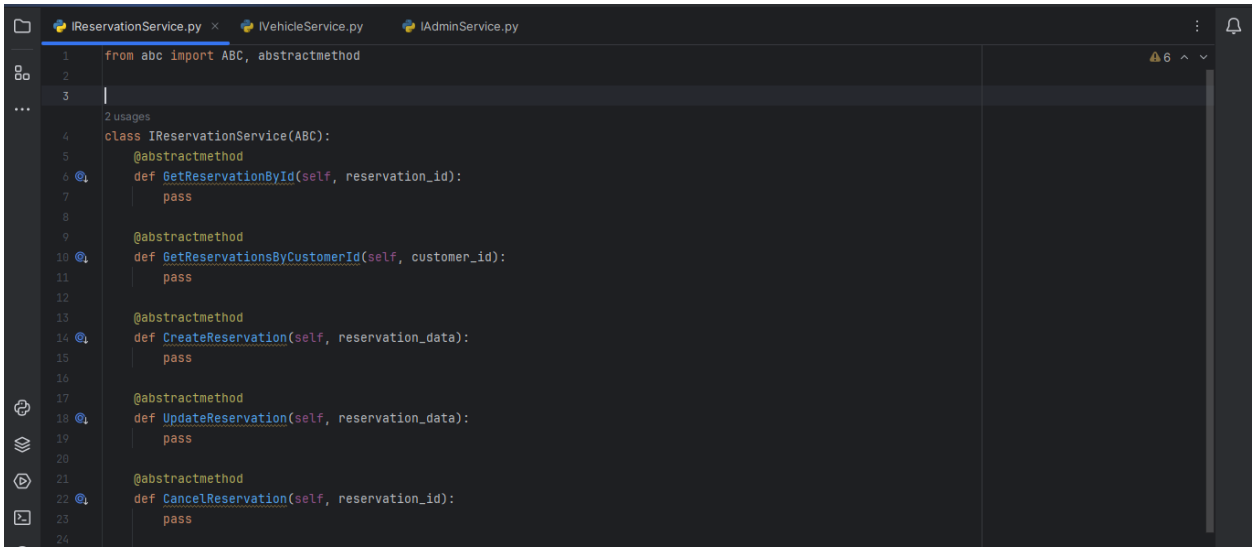
- GetAvailableVehicles()

- AddVehicle(vehicleData)

- UpdateVehicle(vehicleData)
- RemoveVehicle(vehicleId)




- **IReservationService:**
- GetReservationById(reservationId)
- GetReservationsByCustomerId(customerId)
- CreateReservation(reservationData)
- UpdateReservation(reservationData)
- CancelReservation(reservationId)



• **IAdminService:**

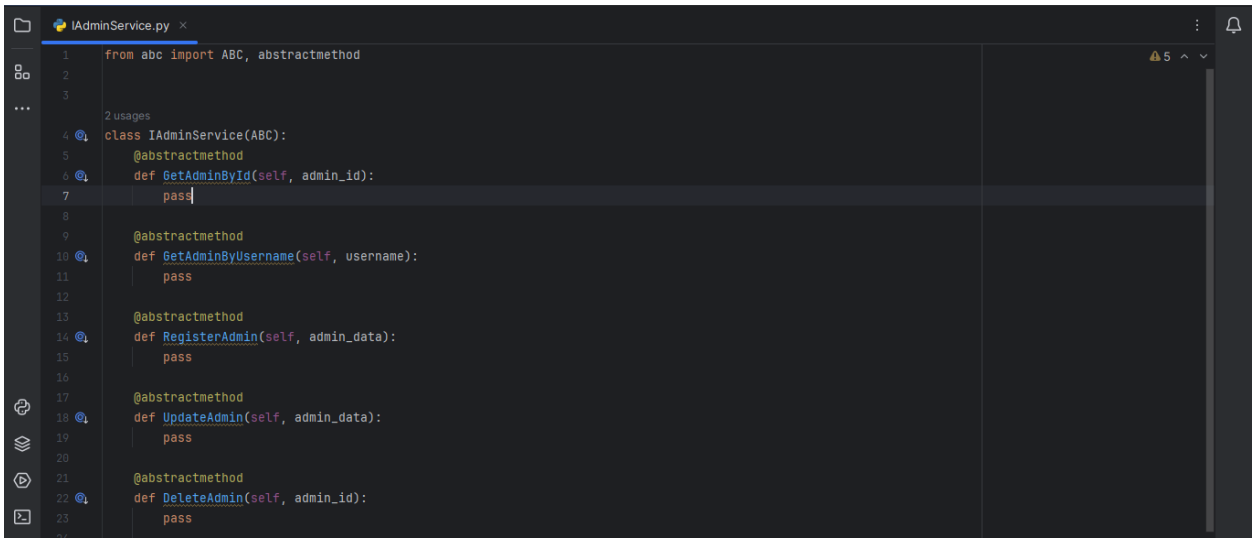
• GetAdminById(adminId)

• GetAdminByUsername(username)

• RegisterAdmin(adminData)

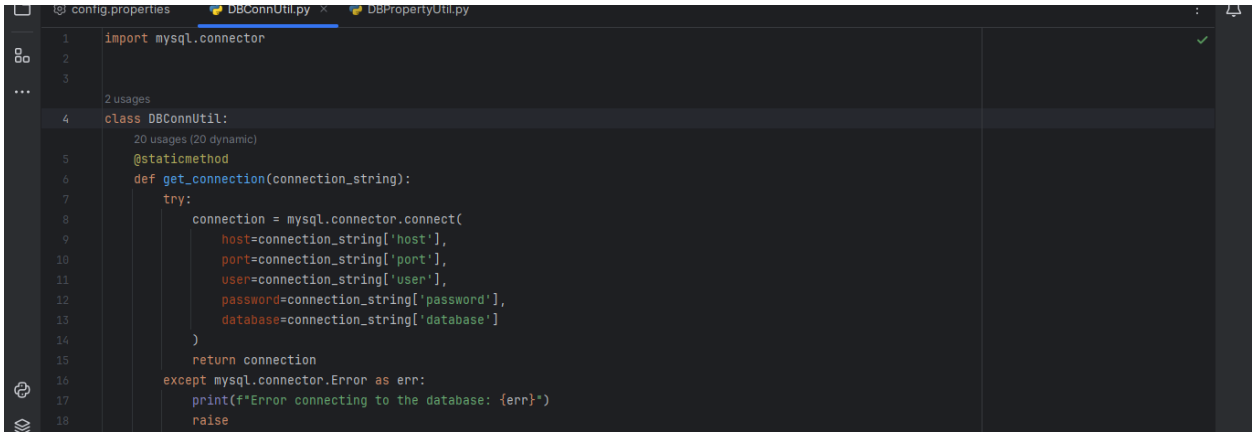
• UpdateAdmin(adminData)

• DeleteAdmin(adminId)




Connect your application to the SQL database:

• Create a connection string that includes the necessary information to connect to your SQL Server database. This includes the server name, database name, authentication credentials, and any other relevant settings.




• Use the SqlConnection class to establish a connection to the SQL Server database

```
[DatabaseConfig]
db_url=localhost
db_port=3306
db_name=carconnect
db_username=root
db_password=Aniket@123
```

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

```
import os
import configparser


2 usages
class DBPropertyUtil:
    1 usage
    @staticmethod
    def get_connection_string():
        script_dir = os.path.dirname(os.path.abspath(__file__))
        config_path = os.path.join(script_dir, 'config.properties')

        config = configparser.ConfigParser()
        config.read(config_path)

        db_config = config['DatabaseConfig']
        connection_string = {
            'host': db_config['db_url'],
            'port': db_config.getint('db_port'),
            'database': db_config['db_name'],
            'user': db_config['db_username'],
            'password': db_config['db_password']
        }

        return connection_string
```

***Custom Exceptions:***

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.

```
class AuthenticationException(Exception):
    def __init__(self, message="Authentication failed"):
        self.message = message
        super().__init__(self.message)


class ReservationException(Exception):
    def __init__(self, message="Reservation issue"):
        self.message = message
        super().__init__(self.message)


3 usages
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)


2 usages
class VehicleNotFoundException(Exception):
    def __init__(self, message="Vehicle not found"):
        self.message = message
        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.

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.

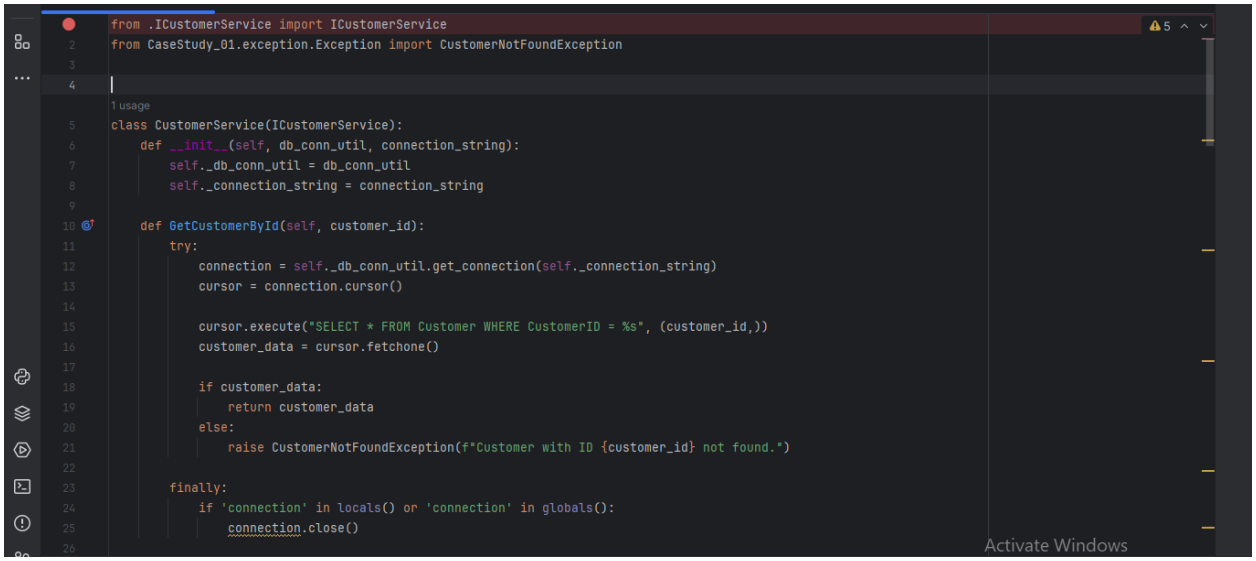
DatabaseConnectionException: • Thrown when there is an issue with the database connection. • Example Usage: Unable to establish a connection to the database.

```
class ReservationNotFoundException(Exception):
    def __init__(self, message="Reservation not found"):
        self.message = message
        super().__init__(self.message)


3 usages
class AdminNotFoundException(Exception):
    def __init__(self, message="Admin not found"):
        self.message = message
        super().__init__(self.message)


class InvalidInputException(Exception):
    def __init__(self, message="Invalid input"):
        self.message = message
        super().__init__(self.message)


class DatabaseConnectionException(Exception):
    def __init__(self, message="Database connection issue"):
        self.message = message
        super().__init__(self.message)
```

**• CustomerService (implements ICustomerService):**

• Methods:

***GetCustomerById,***



```
from .ICustomerService import ICustomerService
from CaseStudy_01.exception.Exception import CustomerNotFoundException


1 usage
class CustomerService(ICustomerService):
    def __init__(self, db_conn_util, connection_string):
        self._db_conn_util = db_conn_util
        self._connection_string = connection_string

    def GetCustomerById(self, customer_id):
        try:
            connection = self._db_conn_util.get_connection(self._connection_string)
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM Customer WHERE CustomerID = %s", (customer_id,))
            customer_data = cursor.fetchone()

            if customer_data:
                return customer_data
            else:
                raise CustomerNotFoundException(f"Customer with ID {customer_id} not found.")

        finally:
            if 'connection' in locals() or 'connection' in globals():
                connection.close()
```

***OUTPUT:-***


***GetCustomerByUsername,***

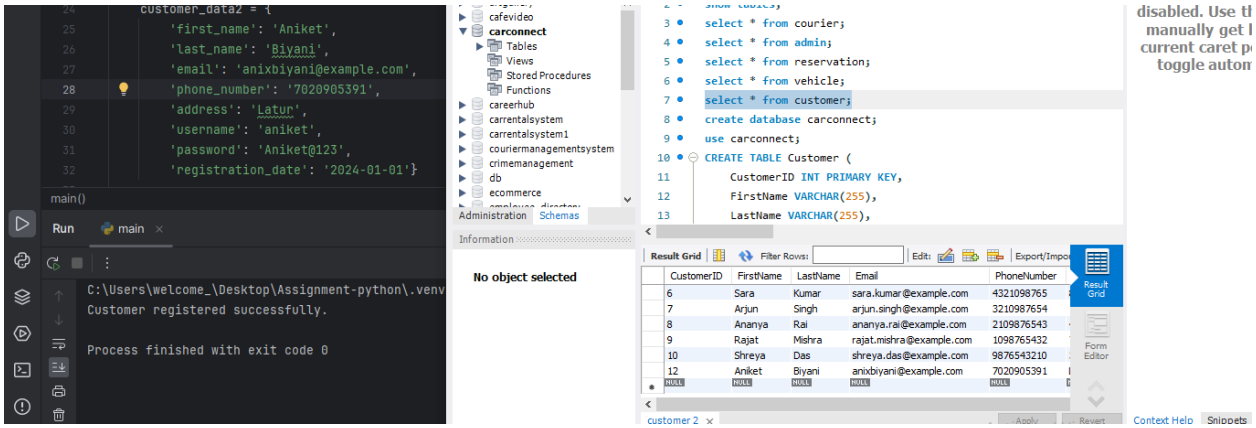

***OUTPUT:-***

## RegisterCustomer

```python
def RegisterCustomer(self, customer_data):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username,
             Password, RegistrationDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            customer_data['first_name'],
            customer_data['last_name'],
            customer_data['email'],
            customer_data['phone_number'],
            customer_data['address'],
            customer_data['username'],
            customer_data['password'],
            customer_data['registration_date']
        ))

        connection.commit()
        print("Customer registered successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
```



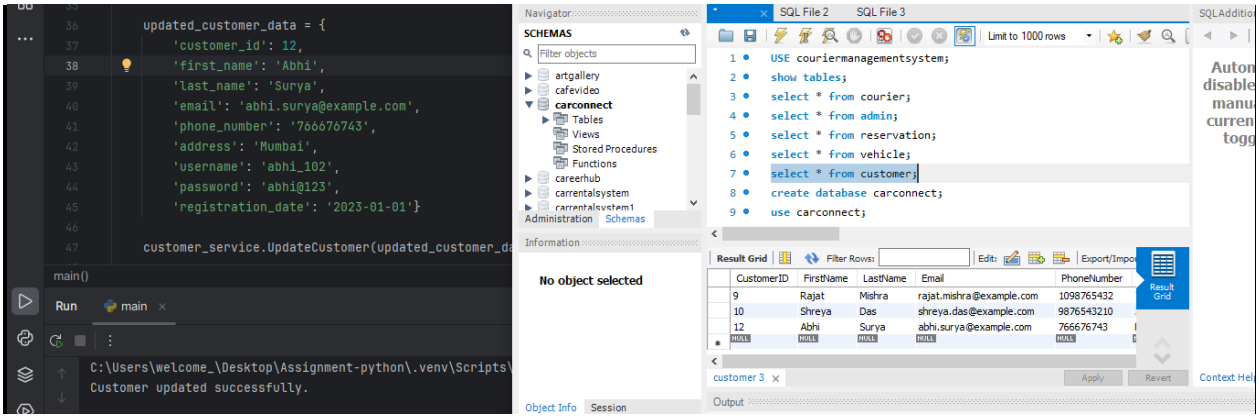
## *OUTPUT:-*




## UpdateCustomer,

```python
def UpdateCustomer(self, customer_data):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()
        cursor.execute("""
            UPDATE Customer
            SET FirstName = %s, LastName = %s, Email = %s, PhoneNumber = %s, Address = %s,
            Username = %s, Password = %s
            WHERE CustomerID = %s
        """, (
            customer_data['first_name'],
            customer_data['last_name'],
            customer_data['email'],
            customer_data['phone_number'],
            customer_data['address'],
            customer_data['username'],
            customer_data['password'],
            customer_data['customer_id']
        ))

        connection.commit()
        print("Customer updated successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
```

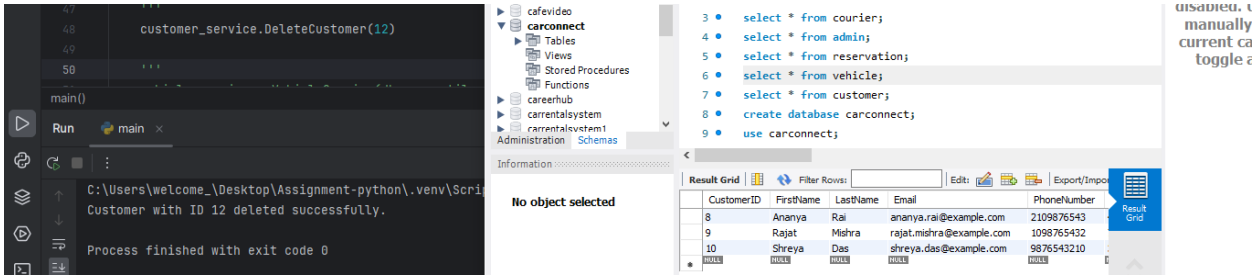
***OUTPUT:-***




## DeleteCustomer

```python
def DeleteCustomer(self, customer_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("DELETE FROM Customer WHERE CustomerID = %s", (customer_id,))

        connection.commit()
        print(f"Customer with ID {customer_id} deleted successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***




- ***VehicleService (implements IVehicleService)***

- Methods:

## GetVehicleById,

```python
def GetVehicleById(self, vehicle_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("SELECT * FROM Vehicle WHERE VehicleID = %s", (vehicle_id,))
        vehicle_data = cursor.fetchone()

        if vehicle_data:
            return vehicle_data
        else:
            raise VehicleNotFoundException(f"Vehicle with ID {vehicle_id} not found.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***

```
    vehicle_service = VehicleService(db_conn_util,connection_string)

    vehicle_id = 1
    vehicle_data = vehicle_service.GetVehicleById(vehicle_id)
    print("Vechicle Data:", vehicle_data)
    '''
main()
```

```
Run   main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Vechicle Data: (1, 'SUV', 'Ford', 2023, 'Red', 'XYZ789', 1, Decimal('60.00'))
```

**GetAvailableVehicles,**

```
    def GetAvailableVehicles(self):
        try:
            connection = self._db_conn_util.get_connection(self._connection_string)
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM Vehicle WHERE Availability>0")
            available_vehicles = cursor.fetchall()

            return available_vehicles

        finally:
            if 'connection' in locals() or 'connection' in globals():
                connection.close()
```

***OUTPUT:-***

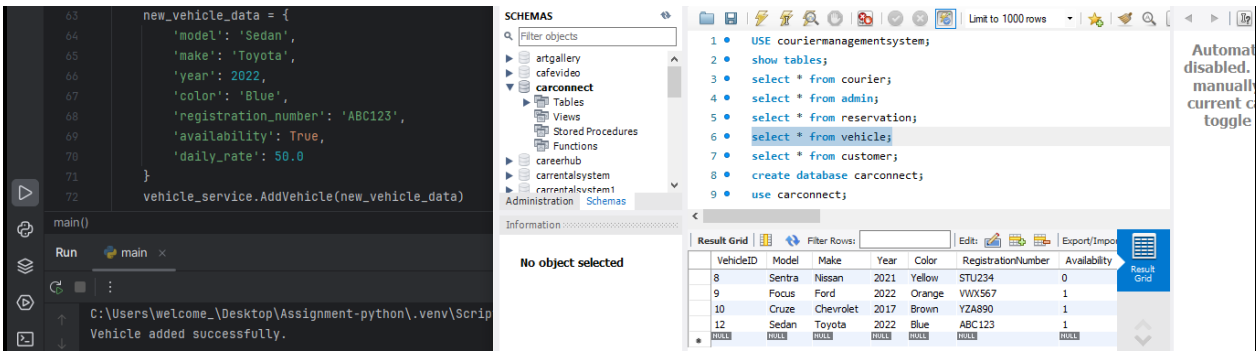
```
    available_vehicles=vehicle_service.GetAvailableVehicles()
    print("Available Vehicles:")
    for vehicle in available_vehicles:
        print(vehicle)
main()  ›  for vehicle in available_vehicl...
```

```
Run   main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Available Vehicles:
(1, 'SUV', 'Ford', 2023, 'Red', 'XYZ789', 1, Decimal('60.00'))
(2, 'Corolla', 'Toyota', 2019, 'Red', 'ABC456', 1, Decimal('45.00'))
(4, 'Camry', 'Toyota', 2022, 'Black', 'GHI012', 1, Decimal('48.00'))
(6, 'Altima', 'Nissan', 2020, 'Gray', 'MNO678', 1, Decimal('47.00'))
(7, 'Malibu', 'Chevrolet', 2019, 'Green', 'PQR901', 1, Decimal('49.00'))
(9, 'Focus', 'Ford', 2022, 'Orange', 'VWX567', 1, Decimal('46.00'))
(10, 'Cruze', 'Chevrolet', 2017, 'Brown', 'YZA890', 1, Decimal('53.00'))
```

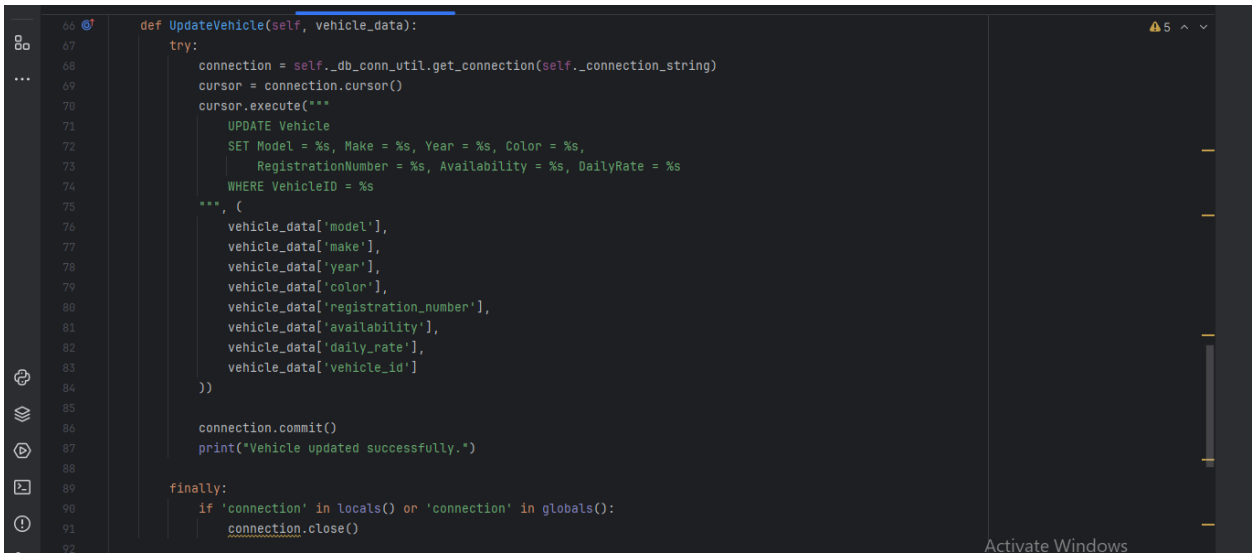
**AddVehicle,**

```
    def AddVehicle(self, vehicle_data):
        try:
            connection = self._db_conn_util.get_connection(self._connection_string)
            cursor = connection.cursor()

            cursor.execute("""
                INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (
                vehicle_data['model'],
                vehicle_data['make'],
                vehicle_data['year'],
                vehicle_data['color'],
                vehicle_data['registration_number'],
                vehicle_data['availability'],
                vehicle_data['daily_rate']
            ))

            connection.commit()
            print("Vehicle added successfully.")

        finally:
            if 'connection' in locals() or 'connection' in globals():
                connection.close()
```

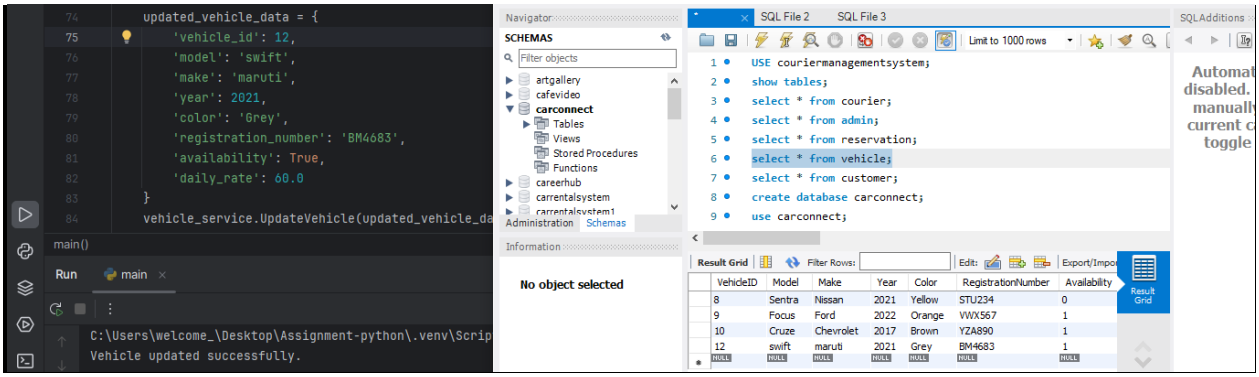
***OUTPUT:-***




**UpdateVehicle,**




***OUTPUT:-***



**RemoveVehicle**

```
1 usage
def RemoveVehicle(self, vehicle_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("DELETE FROM Vehicle WHERE VehicleID = %s", (vehicle_id,))

        connection.commit()
        print("Vehicle removed successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

```
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Vehicle removed successfully.
```

***OUTPUT:-***

```
vehicle_service.RemoveVehicle(12)
main()
Run  main
C:\Users\welcome_\Desktop\Assignment-python\
Vehicle removed successfully.

Process finished with exit code 0
```

careerhub
carrentalsystem
carrentalsystem1
Administration Schemas
Information
No object selected

```
select * from customer;
create database carconnect;
use carconnect;
```

Result Grid | Filter Rows: | Edit: | Export/Import

| VehicleID | Model | Make | Year | Color | RegistrationNumber | Availability |
|---|---|---|---|---|---|---|
| 7 | Malibu | Chevrolet | 2019 | Green | PQR901 | 1 |
| 8 | Sentra | Nissan | 2021 | Yellow | STU234 | 0 |
| 9 | Focus | Ford | 2022 | Orange | VWX567 | 1 |
| 10 | Cruze | Chevrolet | 2017 | Brown | YZA890 | 1 |

vehicle 7 | Apply | Revert | Context Help | Snippets

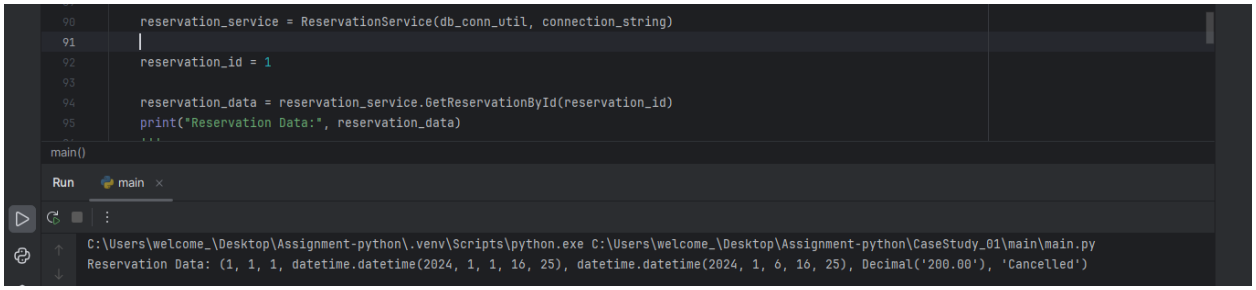
- ***ReservationService (implements IReservationService):***

- Methods:

**GetReservationById,**

```
from .IReservationService import IReservationService
from CaseStudy_01.exception.Exception import ReservationNotFoundException
from decimal import Decimal
2 usages
class ReservationService(IReservationService):
    def __init__(self, db_conn_util, connection_string):
        self._db_conn_util = db_conn_util
        self._connection_string = connection_string

    1 usage
    def GetReservationById(self, reservation_id):
        try:
            connection = self._db_conn_util.get_connection(self._connection_string)
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM Reservation WHERE ReservationID = %s", (reservation_id,))
            reservation_data = cursor.fetchone()

            if reservation_data:
                return reservation_data
            else:
                raise ReservationNotFoundException(f"Reservation with ID {reservation_id} not found.")

        finally:
            if 'connection' in locals() or 'connection' in globals():
                connection.close()
```

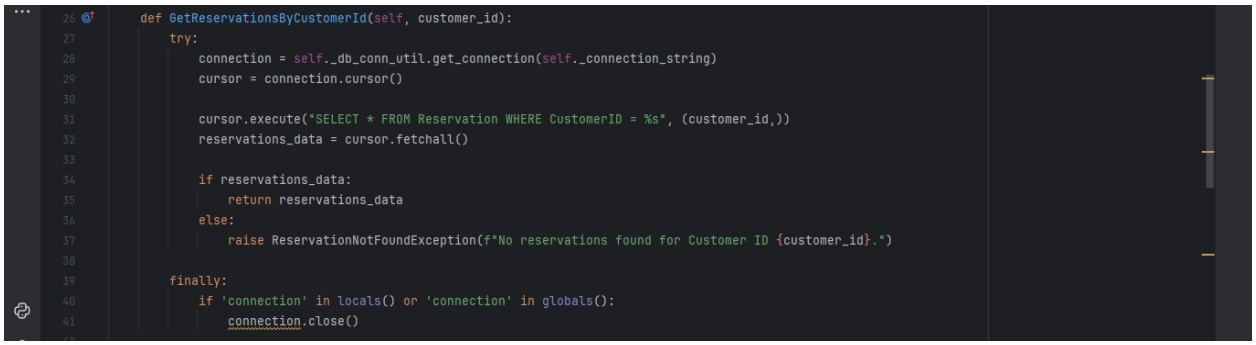
***OUTPUT:-***



```
reservation_service = ReservationService(db_conn_util, connection_string)

reservation_id = 1

reservation_data = reservation_service.GetReservationById(reservation_id)
print("Reservation Data:", reservation_data)
'''
main()
```

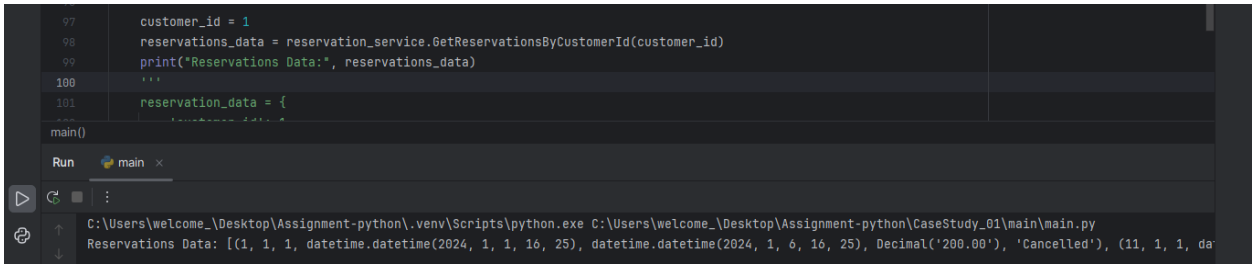
```
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Reservation Data: (1, 1, 1, datetime.datetime(2024, 1, 1, 16, 25), datetime.datetime(2024, 1, 6, 16, 25), Decimal('200.00'), 'Cancelled')
```

**GetReservationsByCustomerId,**



```
def GetReservationsByCustomerId(self, customer_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("SELECT * FROM Reservation WHERE CustomerID = %s", (customer_id,))
        reservations_data = cursor.fetchall()

        if reservations_data:
            return reservations_data
        else:
            raise ReservationNotFoundException(f"No reservations found for Customer ID {customer_id}.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***



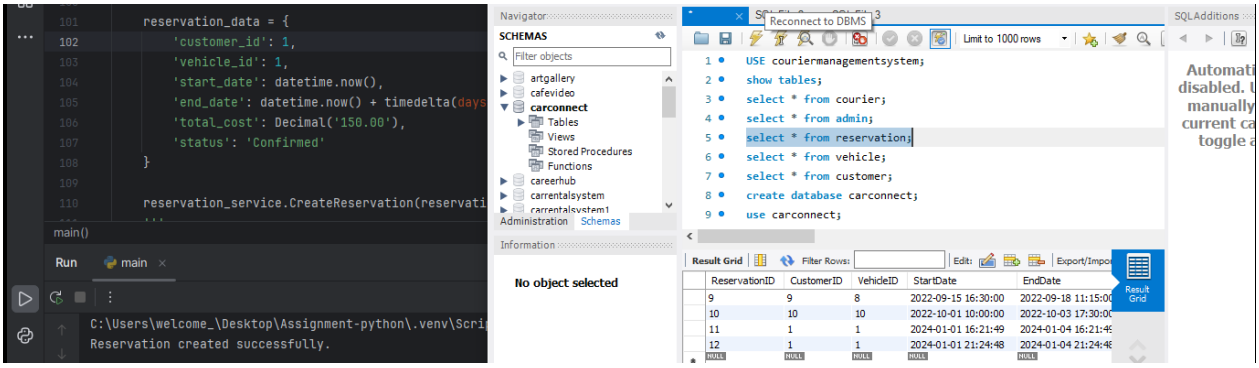
```
customer_id = 1
reservations_data = reservation_service.GetReservationsByCustomerId(customer_id)
print("Reservations Data:", reservations_data)
'''
reservation_data = {
main()
```

```
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Reservations Data: [(1, 1, 1, datetime.datetime(2024, 1, 1, 16, 25), datetime.datetime(2024, 1, 6, 16, 25), Decimal('200.00'), 'Cancelled'), (11, 1, 1, da
```

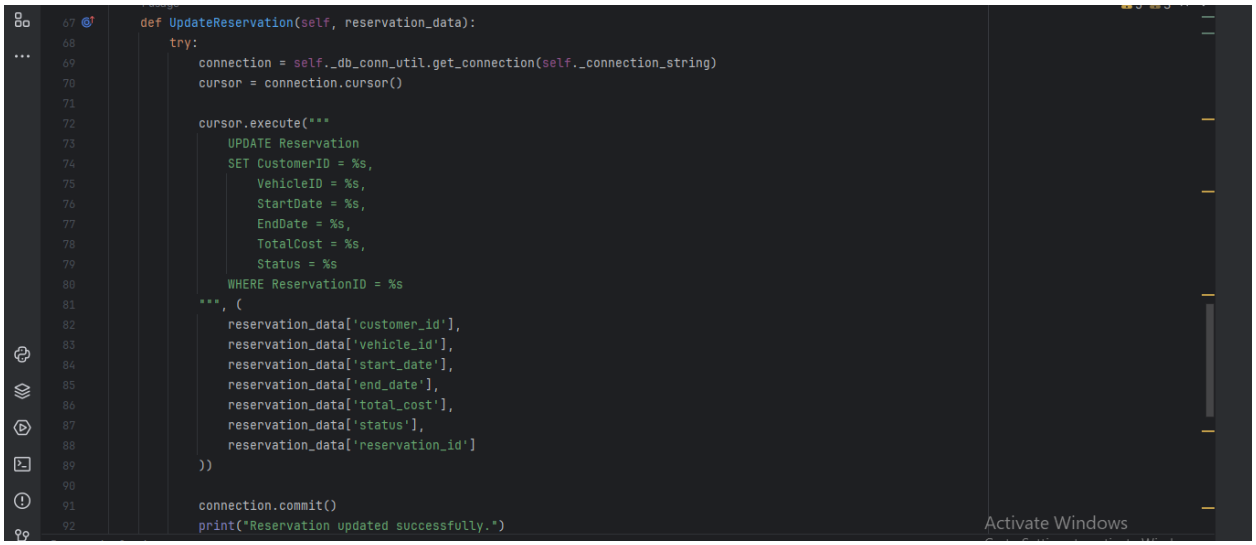
**CreateReservation,**

```
def CreateReservation(self, reservationData,):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
            VALUES (%s, %s, %s, %s, %s, %s)
        """, (
            reservationData['customer_id'],
            reservationData['vehicle_id'],
            reservationData['start_date'],
            reservationData['end_date'],
            Decimal(reservationData['total_cost']),
            reservationData['status']
        ))

        connection.commit()
        print("Reservation created successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

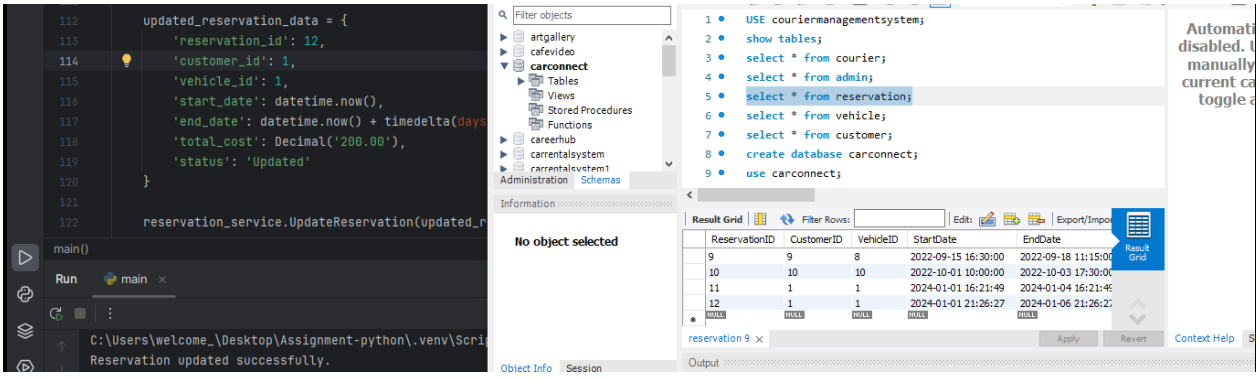
***OUTPUT:-***




**UpdateReservation,**




***OUTPUT:-***



**CancelReservation**

```
1 usage
def CancelReservation(self, reservation_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("""
            UPDATE Reservation
            SET Status = 'Cancelled'
            WHERE ReservationID = %s
        """, (reservation_id,))

        connection.commit()
        print(f"Reservation with ID {reservation_id} has been cancelled.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***

```
reservation_id_to_cancel = 12

reservation_service.CancelReservation(reservation_id_to_cancel)

main()

Run  main

C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Reservation with ID 12 has been cancelled.
```

- ***AdminService (implements IAdminService):***

- Methods:

**GetAdminById,**

```
from .IAdminService import IAdminService
from CaseStudy_01.exception.Exception import AdminNotFoundException


1 usage
class AdminService(IAdminService):
    def __init__(self, db_conn_util, connection_string):
        self._db_conn_util = db_conn_util
        self._connection_string = connection_string

    def GetAdminById(self, admin_id):
        try:
            connection = self._db_conn_util.get_connection(self._connection_string)
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM Admin WHERE AdminID = %s", (admin_id,))
            admin_data = cursor.fetchone()

            if admin_data:
                return admin_data
            else:
                raise AdminNotFoundException(f"Admin with ID {admin_id} not found.")

        finally:
            if 'connection' in locals() or 'connection' in globals():
                connection.close()
```

***OUTPUT:-***

```
admin_service = AdminService(db_conn_util,connection_string)
admin_id_to_fetch = 1

admin_data = admin_service.GetAdminById(admin_id_to_fetch)
print("Admin Data:", admin_data)
'''
```

```
main()
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Admin Data: (1, 'UpdatedAdmin', 'UpdatedUser', 'updated_admin@example.com', '9876543210', 'john_admin', 'new_admin_password', 'Super Admin', datetime.date
```

**GetAdminByUsername,**

```
1 usage
def GetAdminByUsername(self, username):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("SELECT * FROM Admin WHERE Username = %s", (username,))
        admin_data = cursor.fetchone()

        if admin_data:
            return admin_data
        else:
            raise AdminNotFoundException(f"Admin with username {username} not found.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***

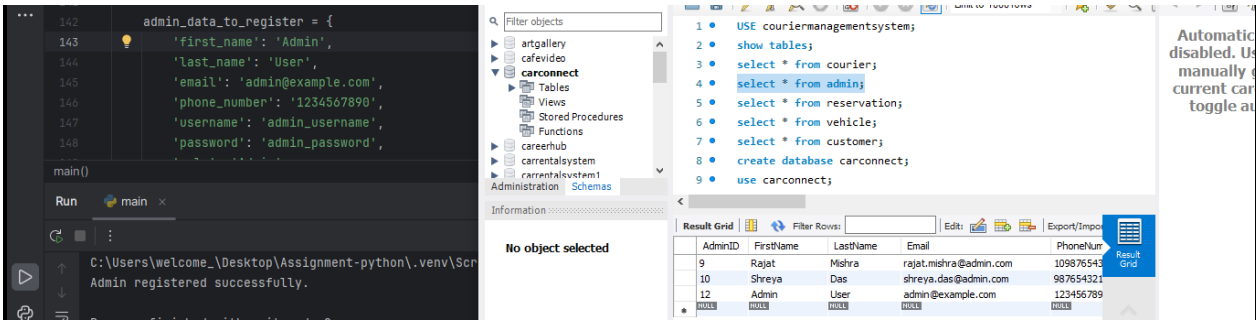
```
admin_username_to_fetch = "john_admin"

admin_data = admin_service.GetAdminByUsername(admin_username_to_fetch)
print("Admin Data:", admin_data)
```

```
main()
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Admin Data: (1, 'UpdatedAdmin', 'UpdatedUser', 'updated_admin@example.com', '9876543210', 'john_admin', 'new_admin_password', 'Super Admin', datetime.date
```

**RegisterAdmin,**

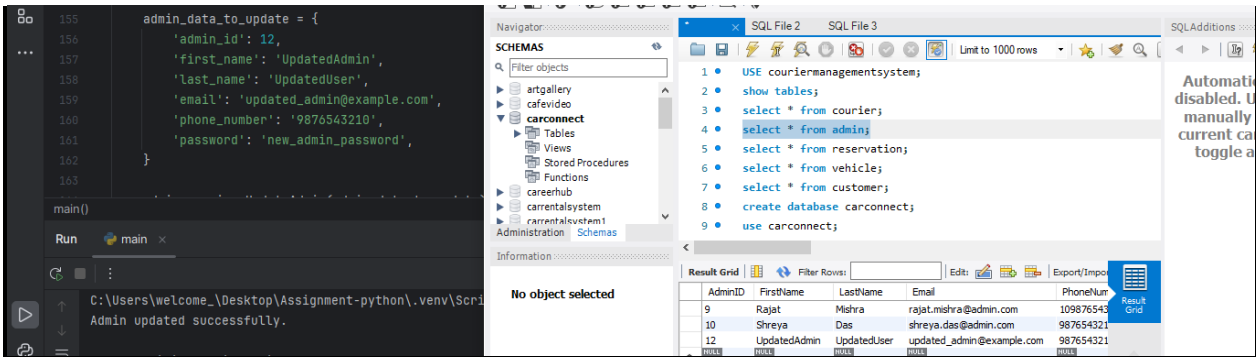
```
def RegisterAdmin(self, admin_data):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("""
              INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
              VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, (
            admin_data['first_name'],
            admin_data['last_name'],
            admin_data['email'],
            admin_data['phone_number'],
            admin_data['username'],
            admin_data['password'],
            admin_data['role'],
            admin_data['join_date']
        ))

        connection.commit()
        print("Admin registered successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***




**UpdateAdmin,**


***OUTPUT:-***



**DeleteAdmin**

```
1 usage
def DeleteAdmin(self, admin_id):
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        cursor.execute("DELETE FROM Admin WHERE AdminID = %s", (admin_id,))

        connection.commit()
        print("Admin deleted successfully.")

    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.close()
```

***OUTPUT:-***

```
admin_id_to_delete = 12

admin_service.DeleteAdmin(admin_id_to_delete)
```

```
main()
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scrip
Admin deleted successfully.

Process finished with exit code 0
```

carconnect
- Tables
- Views
- Stored Procedures
- Functions

careerhub
carrentalsystem
carrentalsystem1

Administration | Schemas

Information

No object selected

```
select * from admin;
select * from reservation;
select * from vehicle;
select * from customer;
create database carconnect;
use carconnect;
```

Result Grid | Filter Rows: | Edit: | Export/Impo

| AdminID | FirstName | LastName | Email | PhoneNum |
|---|---|---|---|---|
| 8 | Ananya | Rai | ananya.rai@admin.com | 210987654 |
| 9 | Rajat | Mishra | rajat.mishra@admin.com | 109876543 |
| 10 | Shreya | Das | shreya.das@admin.com | 987654321 |
| NULL | NULL | NULL | NULL | NULL |

Result Grid

- AuthenticationService:

- A class responsible for handling user authentication.

```
from CaseStudy_01.exception.Exception import AuthenticationException
from CaseStudy_01.dao.CustomerService import CustomerService
from CaseStudy_01.dao.AdminService import AdminService
from CaseStudy_01.util.DBConnUtil import DBConnUtil

2 usages
class AuthenticationService:
    @staticmethod
    def authenticate_customer(username, password, connection_string):
        try:

            customer_service = CustomerService(DBConnUtil(),connection_string)

            customer_data = customer_service.GetCustomerByUsername(username)

            if customer_data and customer_data['Password'] == password:
                print("Authentication successful.")
                return customer_data
            else:
                raise AuthenticationException("Invalid customer credentials")

        except AuthenticationException as e:
            print(f"Error during customer authentication: {str(e)}")
            raise
```

***OUTPUT:-***

```
connection_string = DBPropertyUtil.get_connection_string()

try:
    customer = AuthenticationService.authenticate_customer( username: "john_doe", password: "new_password",
                                                            connection_string)
    print("Customer authenticated:", customer)

except AuthenticationException as e:
main() › try
Run  main
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Authentication successful.
Customer authenticated: {'CustomerID': 1, 'Password': 'new_password'}

Process finished with exit code 0
```

```
1 usage
@staticmethod
def authenticate_admin(username, password, connection_string):
    try:
        admin_service = AdminService(DBConnUtil(), connection_string)

        admin_data = admin_service.GetAdminByUsername(username)

        if admin_data and admin_data['Password'] == password:
            print("Authentication successful.")
            return admin_data
        else:
            raise AuthenticationException("Invalid admin credentials")

    except AuthenticationException as e:
        print(f"Error during admin authentication: {str(e)}")
        raise
```

***OUTPUT:-***

```
connection_string = DBPropertyUtil.get_connection_string()

try:
    admin = AuthenticationService.authenticate_admin( username: "shreya_admin",  password: "hashed_password_10",
                                                      connection_string)
    print("Admin authenticated:", admin)

except AuthenticationException as e:
```

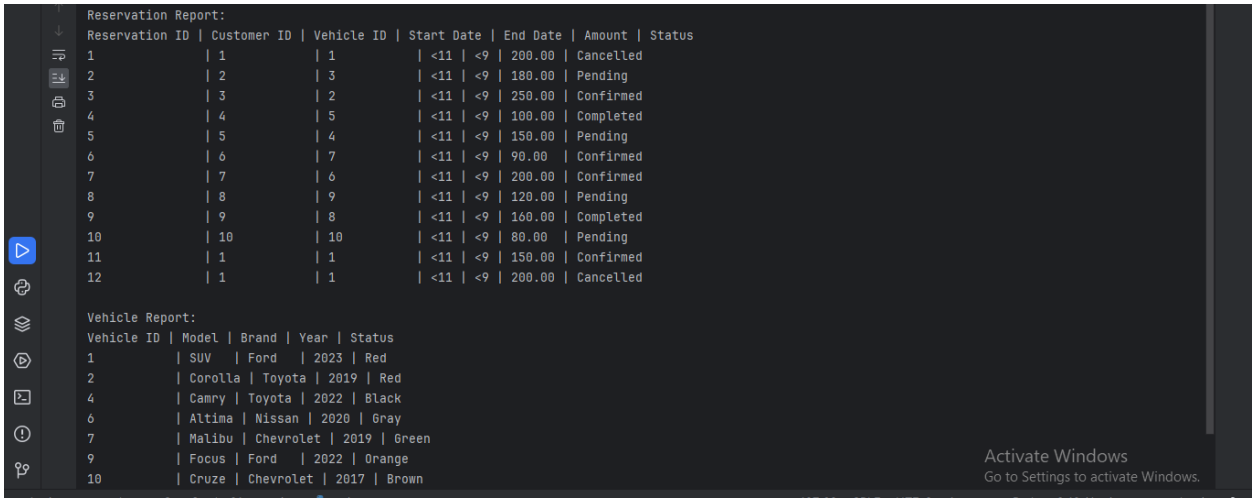
```
main() › try

Run   main ×

C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
Authentication successful.
Admin authenticated: {'AdminID': 10, 'Password': 'hashed_password_10'}
```

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

```
3 usages
class ReportGenerator:
    1 usage
    @staticmethod
    def generate_reservation_report(reservation_data):
        print("Reservation Report:")
        print("Reservation ID | Customer ID | Vehicle ID | Start Date | End Date | Amount | Status")
        for reservation in reservation_data:
            print(
                f"{reservation[0]:<15} | {reservation[1]:<12} | {reservation[2]:<11} | "
                f"{reservation[3]:<11} | {reservation[4]:<9} | {reservation[5]:<6} | {reservation[6]}")

    1 usage
    @staticmethod
    def generate_vehicle_report(vehicle_data):
        print("\nVehicle Report:")
        print("Vehicle ID | Model | Brand | Year | Status")
        for vehicle in vehicle_data:
            print(f"{vehicle[0]:<11} | {vehicle[1]:<5} | {vehicle[2]:<6} | {vehicle[3]:<4} | {vehicle[4]}")
```

```
reservation_service = ReservationService(db_conn_util, connection_string)
vehicle_service = VehicleService(db_conn_util,connection_string)


reservation_data = reservation_service.GetReservationsList()
vehicle_data = vehicle_service.GetAvailableVehicles()

ReportGenerator.generate_reservation_report(reservation_data)
ReportGenerator.generate_vehicle_report(vehicle_data)
```

***OUTPUT:-***



```
Reservation Report:
Reservation ID | Customer ID | Vehicle ID | Start Date | End Date | Amount | Status
1              | 1           | 1          | <11 | <9 | 200.00 | Cancelled
2              | 2           | 3          | <11 | <9 | 180.00 | Pending
3              | 3           | 2          | <11 | <9 | 250.00 | Confirmed
4              | 4           | 5          | <11 | <9 | 100.00 | Completed
5              | 5           | 4          | <11 | <9 | 150.00 | Pending
6              | 6           | 7          | <11 | <9 | 90.00  | Confirmed
7              | 7           | 6          | <11 | <9 | 200.00 | Confirmed
8              | 8           | 9          | <11 | <9 | 120.00 | Pending
9              | 9           | 8          | <11 | <9 | 160.00 | Completed
10             | 10          | 10         | <11 | <9 | 80.00  | Pending
11             | 1           | 1          | <11 | <9 | 150.00 | Confirmed
12             | 1           | 1          | <11 | <9 | 200.00 | Cancelled

Vehicle Report:
Vehicle ID | Model | Brand | Year | Status
1          | SUV   | Ford  | 2023 | Red
2          | Corolla | Toyota | 2019 | Red
4          | Camry | Toyota | 2022 | Black
6          | Altima | Nissan | 2020 | Gray
7          | Malibu | Chevrolet | 2019 | Green
9          | Focus | Ford  | 2022 | Orange
10         | Cruze | Chevrolet | 2017 | Brown
```

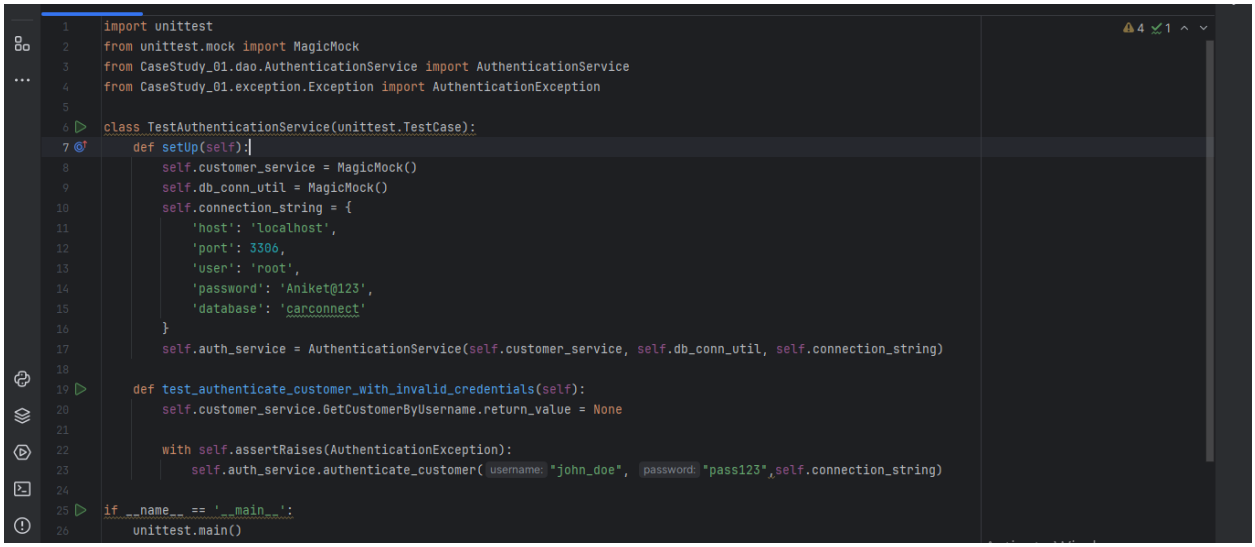


***Unit Testing:***

Create NUnit test cases for car rental System are essential to ensure the correctness and reliability of your system.

Below are some example questions to guide the creation of NUnit test cases for various components of the system:

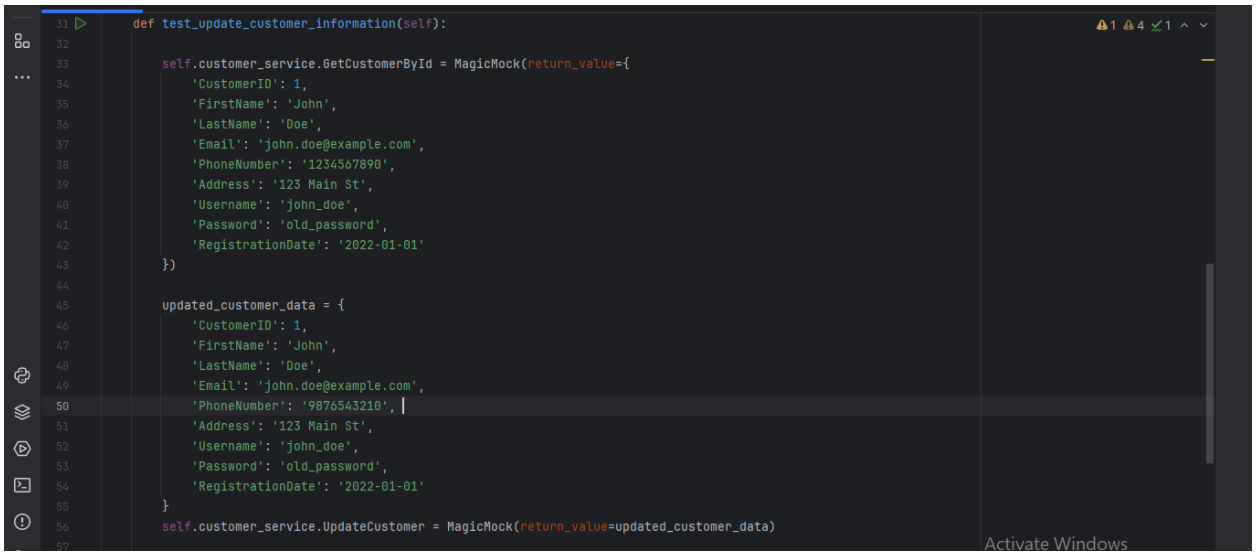
1. ***Test customer authentication with invalid credentials.***



```python
import unittest
from unittest.mock import MagicMock
from CaseStudy_01.dao.AuthenticationService import AuthenticationService
from CaseStudy_01.exception.Exception import AuthenticationException

class TestAuthenticationService(unittest.TestCase):
    def setUp(self):
        self.customer_service = MagicMock()
        self.db_conn_util = MagicMock()
        self.connection_string = {
            'host': 'localhost',
            'port': 3306,
            'user': 'root',
            'password': 'Aniket@123',
            'database': 'carconnect'
        }
        self.auth_service = AuthenticationService(self.customer_service, self.db_conn_util, self.connection_string)

    def test_authenticate_customer_with_invalid_credentials(self):
        self.customer_service.GetCustomerByUsername.return_value = None

        with self.assertRaises(AuthenticationException):
            self.auth_service.authenticate_customer( username: "john_doe",  password: "pass123",self.connection_string)

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

***OUTPUT:-***

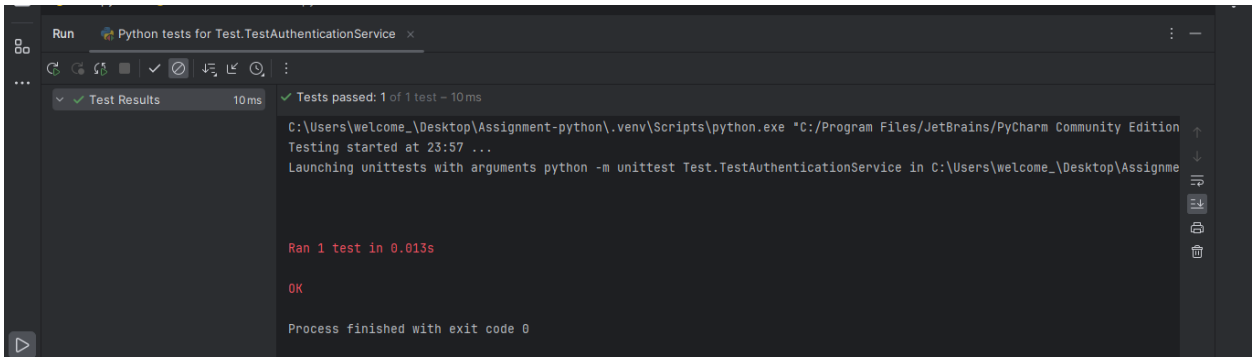

***Test updating customer information.***



```
def test_update_customer_information(self):

    self.customer_service.GetCustomerById = MagicMock(return_value={
        'CustomerID': 1,
        'FirstName': 'John',
        'LastName': 'Doe',
        'Email': 'john.doe@example.com',
        'PhoneNumber': '1234567890',
        'Address': '123 Main St',
        'Username': 'john_doe',
        'Password': 'old_password',
        'RegistrationDate': '2022-01-01'
    })

    updated_customer_data = {
        'CustomerID': 1,
        'FirstName': 'John',
        'LastName': 'Doe',
        'Email': 'john.doe@example.com',
        'PhoneNumber': '9876543210',
        'Address': '123 Main St',
        'Username': 'john_doe',
        'Password': 'old_password',
        'RegistrationDate': '2022-01-01'
    }
    self.customer_service.UpdateCustomer = MagicMock(return_value=updated_customer_data)
```



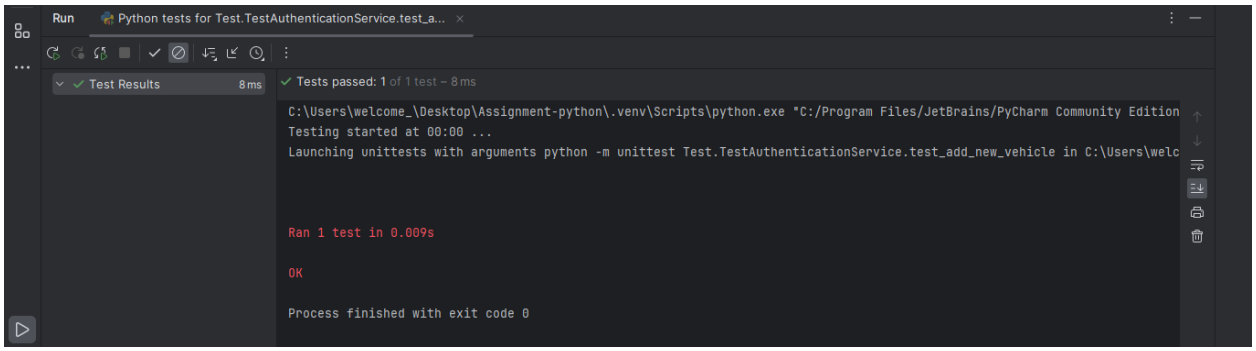
***OUTPUT:-***



### *3. Test adding a new vehicle*

```
    def test_add_new_vehicle(self):

        new_vehicle_data = {
            'VehicleID': 1,
            'Brand': 'Toyota',
            'Model': 'Camry',
            'Year': 2022,
            'RegistrationPlate': 'ABC123',
            'Color': 'Blue',
            'Type': 'Sedan',
            'Availability': True
        }
        self.vehicle_service.AddVehicle = MagicMock(return_value=new_vehicle_data)

        result = self.vehicle_service.AddVehicle(new_vehicle_data)
        self.assertEqual(result, new_vehicle_data)
if __name__ == '__main__':
    unittest.main()
```

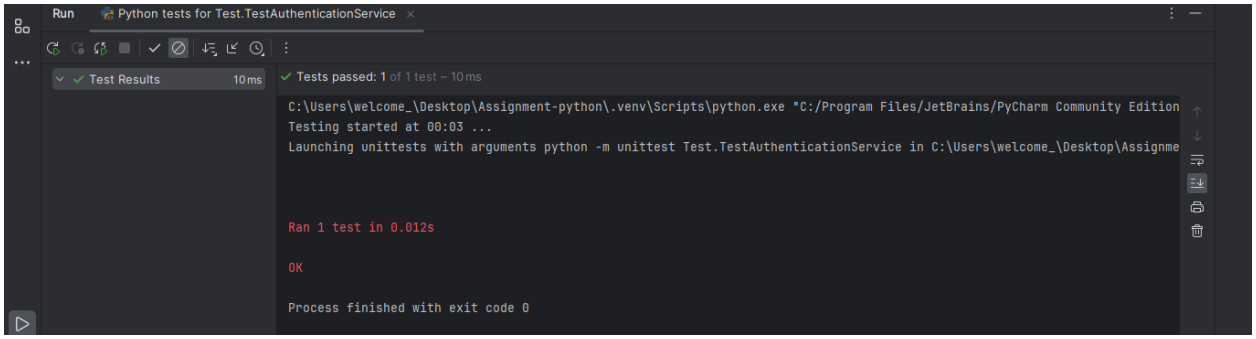
***OUTPUT:-***




### *4. Test updating vehicle details.*

```
    def test_update_vehicle_details(self):
        updated_vehicle_data = {
            'VehicleID': 1,
            'Brand': 'Updated Brand',
            'Model': 'Updated Model',
            'Year': 2023,
            'RegistrationPlate': 'XYZ789',
            'Color': 'Red',
            'Type': 'SUV',
            'Availability': True
        }
        self.vehicle_service.UpdateVehicle = MagicMock(return_value=updated_vehicle_data)

        result = self.vehicle_service.UpdateVehicle( *args: updated_vehicle_data['VehicleID'], updated_vehicle_data)
        self.assertEqual(result, updated_vehicle_data)
if __name__ == '__main__':
    unittest.main()
```

***OUTPUT:-***

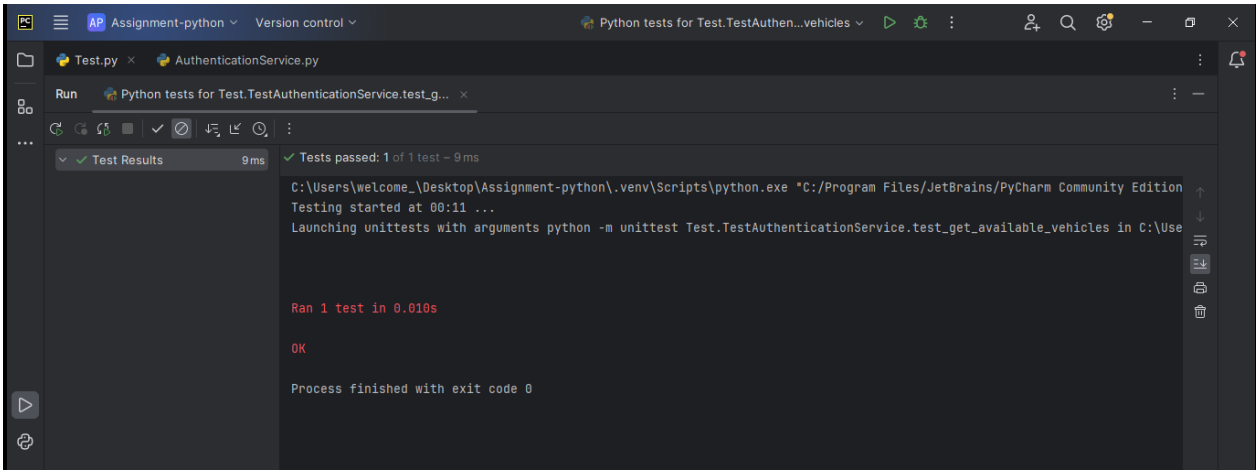


## *5. Test getting a list of available vehicles.*

```python
def test_get_available_vehicles(self):
    available_vehicles = [
        {
            'VehicleID': 1,
            'Brand': 'Toyota',
            'Model': 'Camry',
            'Year': 2022,
            'RegistrationPlate': 'ABC123',
            'Color': 'Blue',
            'Type': 'Sedan',
            'Availability': True
        },
        {
            'VehicleID': 2,
            'Brand': 'Honda',
            'Model': 'Civic',
            'Year': 2021,
            'RegistrationPlate': 'XYZ789',
            'Color': 'Silver',
            'Type': 'Sedan',
            'Availability': True
        }
    ]
    self.vehicle_service.GetAvailableVehicles = MagicMock(return_value=available_vehicles)
    result = self.vehicle_service.GetAvailableVehicles()
```



***OUTPUT:-***




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

```python
def test_get_all_vehicles(self):
    all_vehicles = [
        {
            'VehicleID': 1,
            'Brand': 'Toyota',
            'Model': 'Camry',
            'Year': 2022,
            'RegistrationPlate': 'ABC123',
            'Color': 'Blue',
            'Type': 'Sedan',
            'Availability': True
        },
        {
            'VehicleID': 2,
            'Brand': 'Honda',
            'Model': 'Civic',
            'Year': 2021,
            'RegistrationPlate': 'XYZ789',
            'Color': 'Silver',
            'Type': 'Sedan',
            'Availability': True
        }
    ]
    self.vehicle_service.GetAllVehicles = MagicMock(return_value=all_vehicles)
    result = self.vehicle_service.GetAllVehicles()
    self.assertEqual(result, all_vehicles)
```

***OUTPUT:-***


Run
Python tests for Test.TestAuthenticationService.test_g...
Test Results 10 ms
Tests passed: 1 of 1 test – 10 ms
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition
Testing started at 00:16 ...
Launching unittests with arguments python -m unittest Test.TestAuthenticationService.test_get_all_vehicles in C:\Users\wel
Ran 1 test in 0.011s
OK
Process finished with exit code 0