

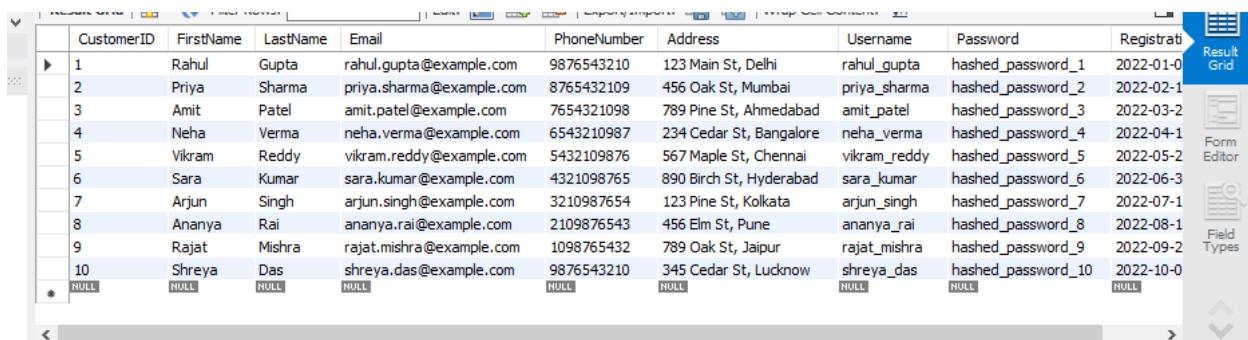
CASE STUDY :-

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

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

	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.

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

	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.

Result Grid									
	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

```
5     class Customer:
6         def __init__(self, customer_id, first_name, last_name, email, phone_number, address, username, password,
...             registration_date):
7             self._customer_id = customer_id
8             self._first_name = first_name
9             self._last_name = last_name
10            self._email = email
11            self._phone_number = phone_number
12            self._address = address
13            self._username = username
14            self._password = password
15            self._registration_date = registration_date
16
17        def get_customer_id(self):
18            return self._customer_id
19
20    def get_first_name(self):
21        return self._first_name
22
23    def get_last_name(self):
24        return self._last_name
25
26    def get_email(self):
27        return self._email
28
29    def get_phone_number(self):
30
```

```
9        def get_password(self):
10            return self._password
11
12        def get_registration_date(self):
13            return self._registration_date
14
15        def set_customer_id(self, customer_id):
16            if isinstance(customer_id, int) and customer_id > 0:
17                self._customer_id = customer_id
18            else:
19                print("Invalid customer_id. It should be a positive integer.")
20
21        def set_first_name(self, first_name):
22            if first_name:
23                self._first_name = first_name
24            else:
25                print("Invalid first_name. It should not be empty.")
26
27        def set_last_name(self, last_name):
28            if last_name:
29                self._last_name = last_name
30            else:
31                print("Invalid last_name. It should not be empty.")
32
33        def set_email(self, email):
34
```

```
69
70        def set_phone_number(self, phone_number):
71            if phone_number and re.match(r"\d{10}", phone_number):
72                self._phone_number = phone_number
73            else:
74                print("Invalid phone number format.")
75
76        def set_address(self, address):
77            if address:
78                self._address = address
79            else:
80                print("Invalid address. It should not be empty.")
81
82        def set_username(self, username):
83            if username:
84                self._username = username
85            else:
86                print("Invalid username. It should not be empty.")
87
88        def set_password(self, password):
89            if password:
90                self._password = password
91            else:
92                print("Invalid password. It should not be empty.")
93
94        def set_registration_date(self, registration_date):
95
```

- Methods: Authenticate(password)

```

  100     def authenticate(self, input_password):
  101         return input_password == self._password
  102
  103     @classmethod
  104     def create_from_input(cls):
  105         customer_id = int(input("Enter CustomerID: "))
  106         first_name = input("Enter First Name: ")
  107         last_name = input("Enter Last Name: ")
  108         email = input("Enter Email: ")
  109         phone_number = input("Enter Phone Number: ")
  110         address = input("Enter Address: ")
  111         username = input("Enter Username: ")
  112         password = input("Enter Password: ")
  113         registration_date = input("Enter Registration Date (YYYY-MM-DD): ")
  114         print("Customer Created Successfully")
  115         return cls(customer_id, first_name, last_name, email, phone_number, address, username, password,
  116                     registration_date)
  117
  118     customer_instance = Customer.create_from_input()
  119     print("Authentication method")
  120     input_password = input("Enter your password: ")
  121
  122

```

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: Bigani
Enter Email: anixbiganii@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 Successfully
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

```

  1  class Vehicle:
  2      def __init__(self, vehicle_id, model, make, year, color, registration_number, availability, daily_rate):
  3          self._vehicle_id = vehicle_id
  4          self._model = model
  5          self._make = make
  6          self._year = year
  7          self._color = color
  8          self._registration_number = registration_number
  9          self._availability = availability
 10          self._daily_rate = daily_rate
 11
 12      def get_vehicle_id(self):
 13          return self._vehicle_id
 14
 15      def get_model(self):
 16          return self._model
 17
 18      def get_make(self):
 19          return self._make
 20
 21      def get_year(self):
 22          return self._year
 23
 24      def get_color(self):
 25          return self._color
 26

```

```

50  35
51  def set_vehicle_id(self, vehicle_id):
52      if isinstance(vehicle_id, int) and vehicle_id > 0:
53          self._vehicle_id = vehicle_id
54      else:
55          print("Invalid vehicle_id. It should be a positive integer.")
56
57  def set_model(self, model):
58      if model:
59          self._model = model
60      else:
61          print("Invalid model. It should not be empty.")
62
63  def set_make(self, make):
64      if make:
65          self._make = make
66      else:
67          print("Invalid make. It should not be empty.")
68
69  def set_year(self, year):
70      if isinstance(year, int) and year > 0:
71          self._year = year
72      else:
73          print("Invalid year. It should be a positive integer.")
74
75  def set_color(self, color):

```

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

```

1  class Reservation:
2      def __init__(self, reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status):
3          self._reservation_id = reservation_id
4          self._customer_id = customer_id
5          self._vehicle_id = vehicle_id
6          self._start_date = start_date
7          self._end_date = end_date
8          self._total_cost = total_cost
9          self._status = status
10
11     def get_reservation_id(self):
12         return self._reservation_id
13
14     def get_customer_id(self):
15         return self._customer_id
16
17     def get_vehicle_id(self):
18         return self._vehicle_id
19
20     def get_start_date(self):
21         return self._start_date
22
23     def get_end_date(self):
24         return self._end_date
25
26     def get_total_cost(self):
27         return self._total_cost

```

```

50  31
51  def set_reservation_id(self, reservation_id):
52      if isinstance(reservation_id, int) and reservation_id > 0:
53          self._reservation_id = reservation_id
54      else:
55          print("Invalid reservation_id. It should be a positive integer.")
56
57  def set_customer_id(self, customer_id):
58      if isinstance(customer_id, int) and customer_id > 0:
59          self._customer_id = customer_id
60      else:
61          print("Invalid customer_id. It should be a positive integer.")
62
63  def set_vehicle_id(self, vehicle_id):
64      if isinstance(vehicle_id, int) and vehicle_id > 0:
65          self._vehicle_id = vehicle_id
66      else:
67          print("Invalid vehicle_id. It should be a positive integer.")
68
69  def set_start_date(self, start_date):
70      if isinstance(start_date, str):
71          self._start_date = start_date
72      else:
73          print("Invalid start_date. It should be a valid date format.")
74

```

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

```

 1 usage
 2 def calculate_total_cost(self):
 3     start_date = datetime.strptime(self._start_date, "%Y-%m-%d")
 4     end_date = datetime.strptime(self._end_date, "%Y-%m-%d")
 5     days_rented = (end_date - start_date).days
 6     daily_rate = 50
 7     self._total_cost = days_rented * daily_rate
 8
 9
10 usage
11 @classmethod
12 def create_from_input(cls):
13     reservation_id = int(input("Enter ReservationID: "))
14     customer_id = int(input("Enter CustomerID: "))
15     vehicle_id = int(input("Enter VehicleID: "))
16     start_date = input("Enter Start Date (YYYY-MM-DD): ")
17     end_date = input("Enter End Date (YYYY-MM-DD): ")
18     total_cost = 0
19     status = input("Enter Status: ")
20     print("Reservation done successfully")
21     return cls(reservation_id, customer_id, vehicle_id, start_date, end_date, total_cost, status)
22
23 reservation_instance = Reservation.create_from_input()
24 reservation_instance.calculate_total_cost()
25 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 successfully
Total Cost: 500
Process finished with exit code 0

```

- **Admin**

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

```

 4 class Admin:
 5     def __init__(self, admin_id, first_name, last_name, email, phone_number, username, password, role, join_date):
 6         self._admin_id = admin_id
 7         self._first_name = first_name
 8         self._last_name = last_name
 9         self._email = email
10         self._phone_number = phone_number
11         self._username = username
12         self._password = password
13         self._role = role
14         self._join_date = join_date
15
16     def get_admin_id(self):
17         return self._admin_id
18
19     def get_first_name(self):
20         return self._first_name
21
22     def get_last_name(self):
23         return self._last_name
24
25     def get_email(self):
26         return self._email
27
28     def get_phone_number(self):
29         return self._phone_number

```

```

34     def get_password(self):
35         return self._password
...
37     def get_role(self):
38         return self._role
39
40     def get_join_date(self):
41         return self._join_date
42
43     def set_admin_id(self, admin_id):
44         if isinstance(admin_id, int) and admin_id > 0:
45             self._admin_id = admin_id
46         else:
47             print("Invalid admin_id. It should be a positive integer.")
48
49     def set_first_name(self, first_name):
50         if first_name:
51             self._first_name = first_name
52         else:
53             print("Invalid first_name. It should not be empty.")
54
55     def set_last_name(self, last_name):
56         if last_name:
57             self._last_name = last_name
58         else:
59             print("Invalid last_name. It should not be empty.")
60

```

```

61     def set_email(self, email):
62         if email and re.match(r"^[^@]+@[^@]+\.[^@]+", email):
63             self._email = email
64         else:
65             print("Invalid email format.")
66
67     def set_phone_number(self, phone_number):
68         if phone_number and re.match(r"\d{10}", phone_number):
69             self._phone_number = phone_number
70         else:
71             print("Invalid phone number format.")
72
73     def set_username(self, username):
74         if username:
75             self._username = username
76         else:
77             print("Invalid username. It should not be empty.")
78
79     def set_password(self, password):
80         if password:
81             self._password = password
82         else:
83             print("Invalid password. It should not be empty.")
84

```

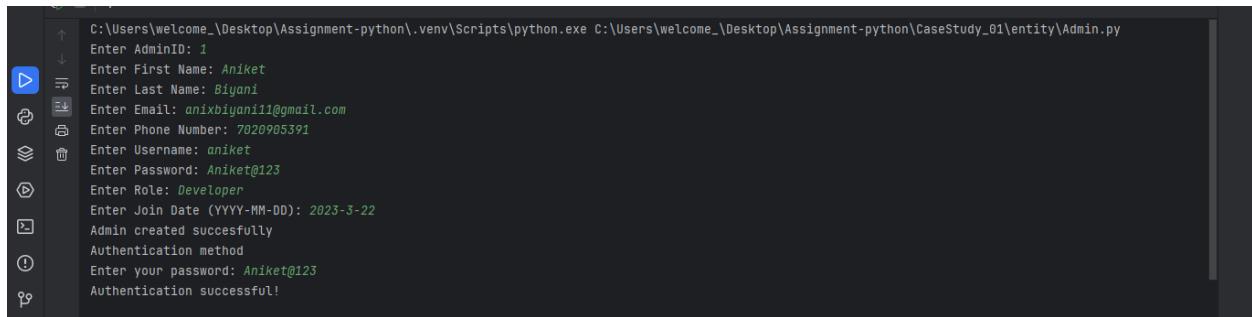
- Methods: Authenticate(password)

```

95     def Authenticate(self, input_password):
96         return input_password == self._password
...
97
98     1 usage
99     @classmethod
100    def create_from_input(cls):
101        admin_id = int(input("Enter AdminID: "))
102        first_name = input("Enter First Name: ")
103        last_name = input("Enter Last Name: ")
104        email = input("Enter Email: ")
105        phone_number = input("Enter Phone Number: ")
106        username = input("Enter Username: ")
107        password = input("Enter Password: ")
108        role = input("Enter Role: ")
109        join_date = input("Enter Join Date (YYYY-MM-DD): ")
110        print("Admin created successfully")
111        return cls(admin_id, first_name, last_name, email, phone_number, username, password, role, join_date)
112
113    admin_instance = Admin.create_from_input()
114    print("Authentication method")
115
116    input_password = input("Enter your password: ")
117
118    if admin_instance.Authenticate(input_password):
119        print("Authentication successful!")
120    else:
121        print("Authentication failed. Incorrect password.")

```

OUTPUT:-



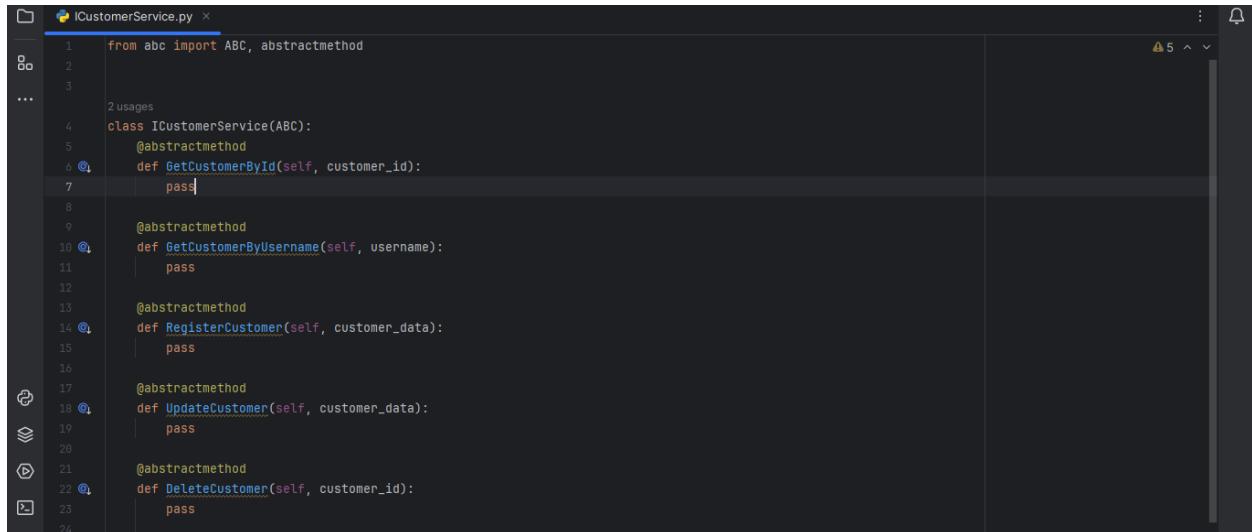
```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\entity\Admin.py
Enter AdminID: 1
Enter First Name: Aniket
Enter Last Name: Bigan
Enter Email: anixbigan11@gmail.com
Enter Phone Number: 7020905391
Enter Username: aniket
Enter Password: Aniket@123
Enter Role: Developer
Enter Join Date (YYYY-MM-DD): 2023-3-22
Admin created successfully
Authentication method
Enter your password: Aniket@123
Authentication successful!
```

Interfaces:

- **ICustomerService:**

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

- DeleteCustomer(customerId)



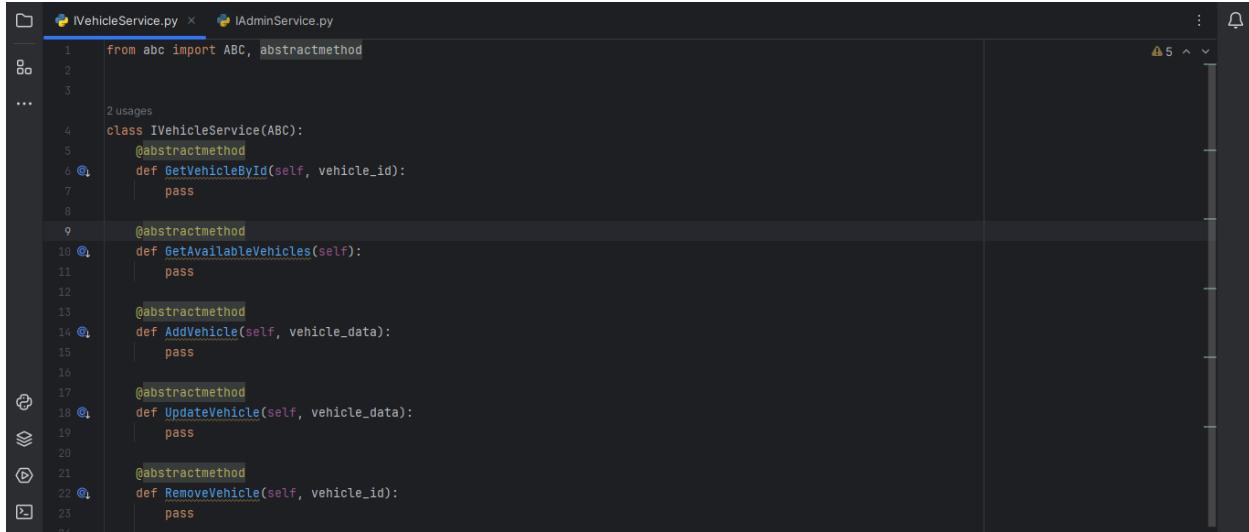
```
ICustomerService.py
1 from abc import ABC, abstractmethod
2
3 ...
4 class ICustomerService(ABC):
5     @abstractmethod
6     def GetCustomerById(self, customer_id):
7         pass
8
9     @abstractmethod
10    def GetCustomerByUsername(self, username):
11        pass
12
13    @abstractmethod
14    def RegisterCustomer(self, customer_data):
15        pass
16
17    @abstractmethod
18    def UpdateCustomer(self, customer_data):
19        pass
20
21    @abstractmethod
22    def DeleteCustomer(self, customer_id):
23        pass
24
```

- **IVehicleService:**

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

- UpdateVehicle(vehicleData)

- RemoveVehicle(vehicleId)



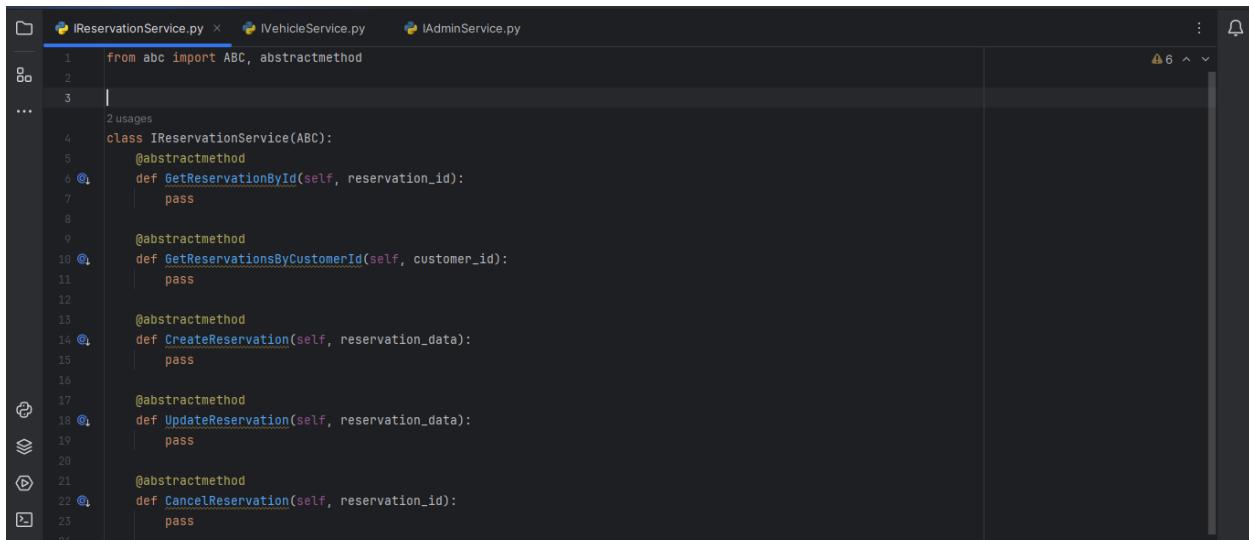
```

1 from abc import ABC, abstractmethod
2
3 ...
4 class IVehicleService(ABC):
5     @abstractmethod
6     def GetVehicleById(self, vehicle_id):
7         pass
8
9     @abstractmethod
10    def GetAvailableVehicles(self):
11        pass
12
13    @abstractmethod
14    def AddVehicle(self, vehicle_data):
15        pass
16
17    @abstractmethod
18    def UpdateVehicle(self, vehicle_data):
19        pass
20
21    @abstractmethod
22    def RemoveVehicle(self, vehicle_id):
23        pass

```

• IReservationService:

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



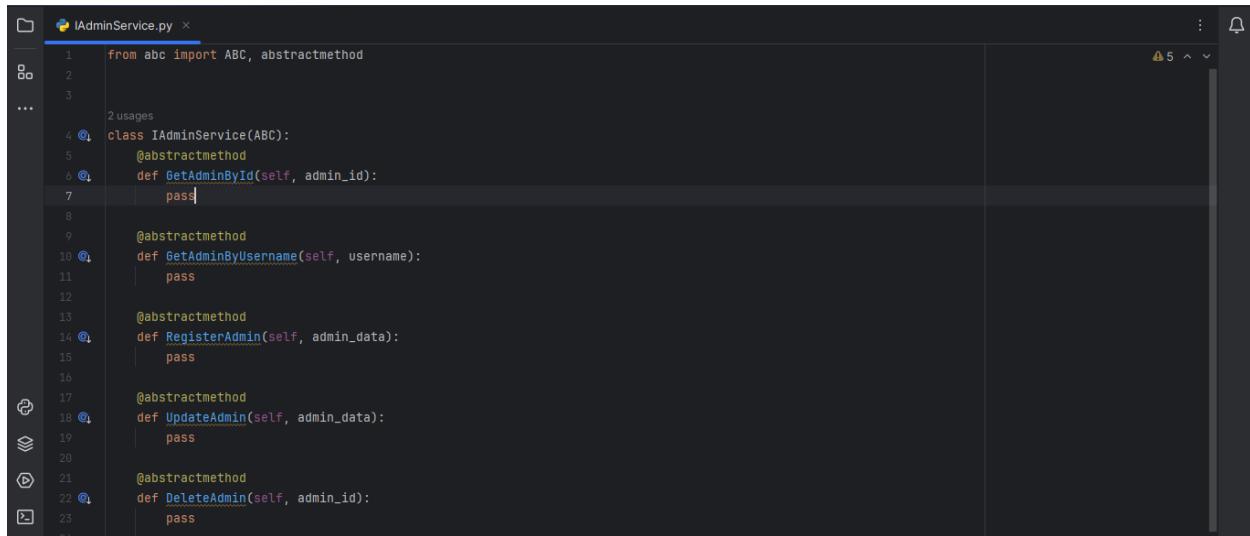
```

1 from abc import ABC, abstractmethod
2
3 ...
4 class IReservationService(ABC):
5     @abstractmethod
6     def GetReservationById(self, reservation_id):
7         pass
8
9     @abstractmethod
10    def GetReservationsByCustomerId(self, customer_id):
11        pass
12
13    @abstractmethod
14    def CreateReservation(self, reservation_data):
15        pass
16
17    @abstractmethod
18    def UpdateReservation(self, reservation_data):
19        pass
20
21    @abstractmethod
22    def CancelReservation(self, reservation_id):
23        pass

```

- **IAdminService:**

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



```
from abc import ABC, abstractmethod

...
2 usages
class IAdminService(ABC):
    @abstractmethod
    def GetAdminById(self, admin_id):
        pass

    @abstractmethod
    def GetAdminByUsername(self, username):
        pass

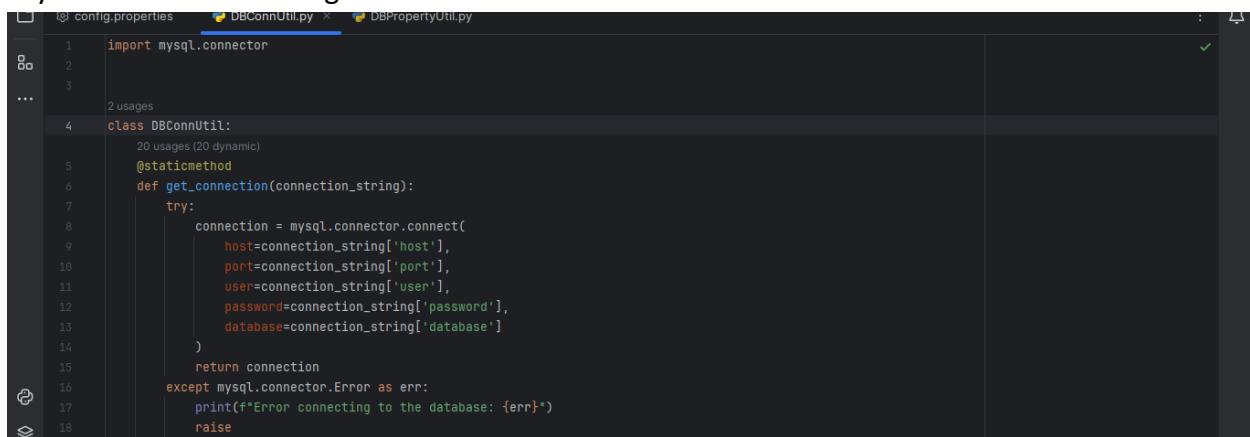
    @abstractmethod
    def RegisterAdmin(self, admin_data):
        pass

    @abstractmethod
    def UpdateAdmin(self, admin_data):
        pass

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

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.



```
import mysql.connector

...
2 usages
class DBConnUtil:
    @staticmethod
    def get_connection(connection_string):
        try:
            connection = mysql.connector.connect(
                host=connection_string['host'],
                port=connection_string['port'],
                user=connection_string['user'],
                password=connection_string['password'],
                database=connection_string['database']
            )
            return connection
        except mysql.connector.Error as err:
            print(f"Error connecting to the database: {err}")
            raise
```

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

The screenshot shows a code editor interface with two files open: config.properties and DBConnUtil.py. The config.properties file contains a single section named [DatabaseConfig] with the following key-value pairs:

```
[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

The screenshot shows the DBPropertyUtil.py file. It imports os and configparser, defines a class DBPropertyUtil with a static method get_connection_string(), and returns a dictionary representing the database connection parameters.

```
import os
import configparser

class DBPropertyUtil:
    @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.
Usage: Incorrect username or password during customer or admin login.

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

The screenshot shows four exception classes defined in a single file:

- AuthenticationException: Thrown when authentication fails.
- ReservationException: Thrown when there is a reservation issue.
- CustomerNotFoundException: Thrown when a customer is not found.
- VehicleNotFoundException: Thrown when a vehicle is not found.

```
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)

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

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

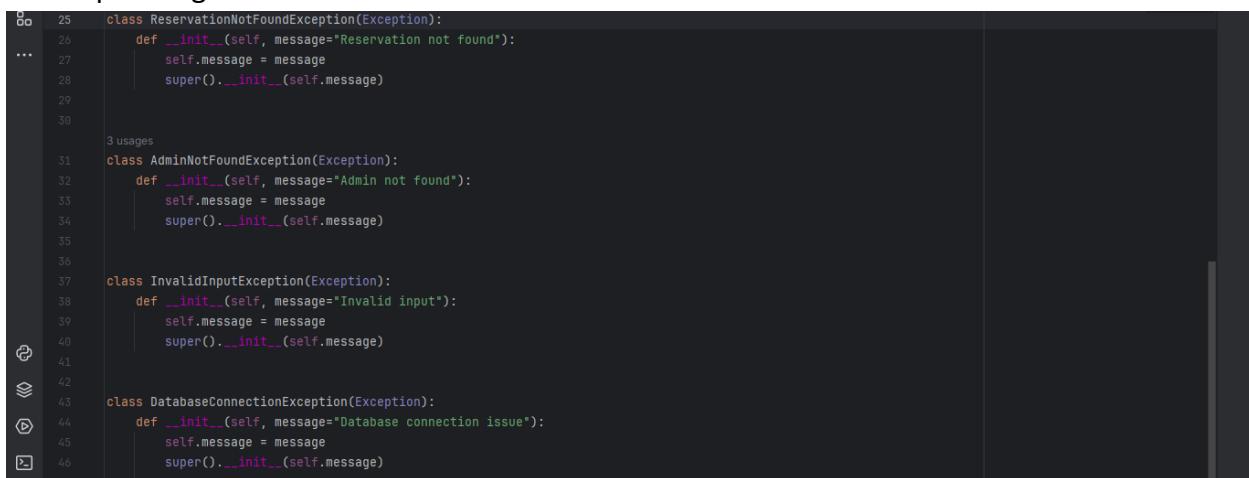
VehicleNotFoundException: • Thrown when a requested vehicle is not found. • Example Usage: Trying to get details of a vehicle that does not exist.

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.

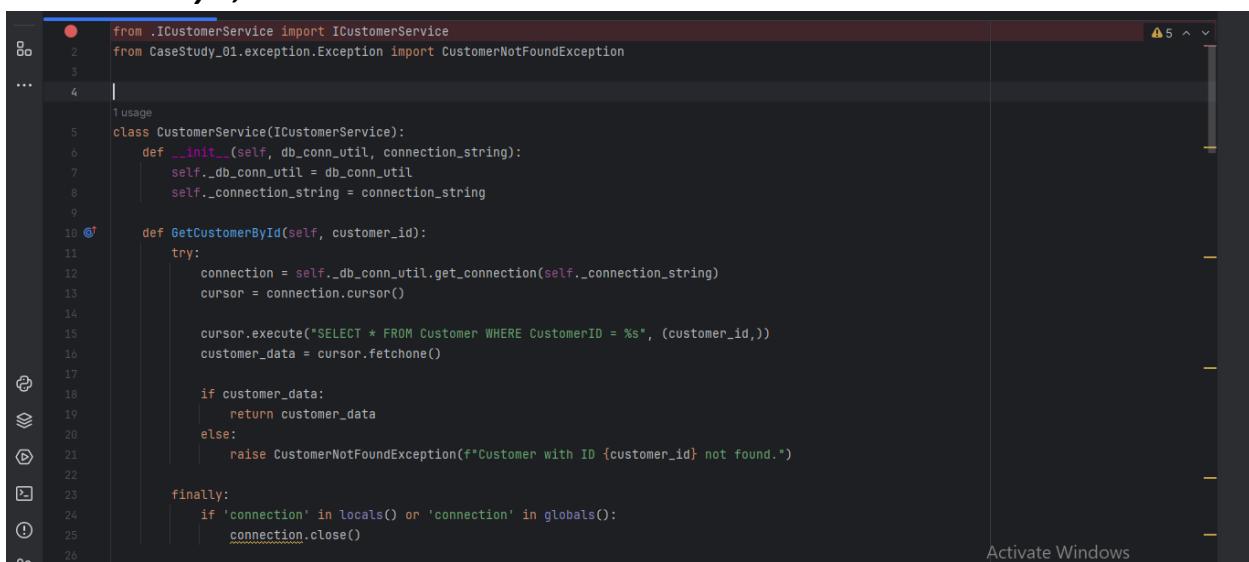


```
00 25 class ReservationNotFoundException(Exception):
01 26     def __init__(self, message="Reservation not found"):
02 ...
03     self.message = message
04     super().__init__(self.message)
05
06
07 3 usages
08 31 class AdminNotFoundException(Exception):
09 32     def __init__(self, message="Admin not found"):
10 ...
11     self.message = message
12     super().__init__(self.message)
13
14
15 37 class InvalidInputException(Exception):
16 38     def __init__(self, message="Invalid input"):
17 ...
18     self.message = message
19     super().__init__(self.message)
20
21
22 43 class DatabaseConnectionException(Exception):
23 44     def __init__(self, message="Database connection issue"):
24 ...
25     self.message = message
26     super().__init__(self.message)
```

- **CustomerService (implements ICustomerService):**

- Methods:

GetCustomerId,



```
00 1 from .ICustomerService import ICustomerService
01 2 from CaseStudy_01.exception.Exception import CustomerNotFoundException
02 ...
03
04 |
05 1 usage
06 5 class CustomerService(ICustomerService):
07 6     def __init__(self, db_conn_util, connection_string):
08 ...
09     self._db_conn_util = db_conn_util
10     self._connection_string = connection_string
11
12 @
13 10 def GetCustomerId(self, customer_id):
14     try:
15         connection = self._db_conn_util.get_connection(self._connection_string)
16         cursor = connection.cursor()
17
18         cursor.execute("SELECT * FROM Customer WHERE CustomerID = %s", (customer_id,))
19         customer_data = cursor.fetchone()
20
21         if customer_data:
22             return customer_data
23         else:
24             raise CustomerNotFoundException(f"Customer with ID {customer_id} not found.")
25
26     finally:
27         if 'connection' in locals() or 'connection' in globals():
28             connection.close()
```

Activate Windows

OUTPUT:-

```
11     connection_string = DBPropertyUtil.getConnectionString()
12     db_conn_util = DBConnUtil()
13
14     customer_service = CustomerService(db_conn_util,connection_string)
15
16     customer_id = 1
17     customer_data = customer_service.GetCustomerById(customer_id)
18     print("Customer Data:", customer_data)
19     """
20     username='Rahul'
21
22 main()
```

Run main x

C:\Users\welcome\Desktop\Assignment-python\venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\CaseStudy_01\main\main.py

Customer Data: (1, 'John', 'Doe', 'john.doe@example.com', '1234567890', '123 Main St', 'john_doe', 'new_password', datetime.date(2022, 1, 1))

Process finished with exit code 0

GetCustomerByUsername,

```
27     def GetCustomerByUsername(self, username):
28         try:
29             connection = self._db_conn_util.get_connection(self._connection_string)
30             cursor = connection.cursor()
31
32             cursor.execute("SELECT * FROM Customer WHERE Username = %s", (username,))
33             customer_data1 = cursor.fetchone()
34             if customer_data1:
35                 customer_dict = {
36                     'customer_id': customer_data1[0],
37                     'first_name': customer_data1[1],
38                     'last_name': customer_data1[2],
39                     'email': customer_data1[3],
40                     'phone_number': customer_data1[4],
41                     'address': customer_data1[5],
42                     'username': customer_data1[6],
43                     'password': customer_data1[7],
44                     'registration_date': customer_data1[8]
45                 }
46                 return customer_dict
47             else:
48                 raise CustomerNotFoundException(f"Customer with FirstName {username} not found.")
49
50         finally:
51             if 'connection' in locals() or 'connection' in globals():
52                 connection.close()
```

Activate Windows

OUTPUT:-

```
20     username='priya_sharma'
21     customer_data1 = customer_service.GetCustomerByUsername(username)
22     print("Customer Data:", customer_data)
23     """
24
25 main()
```

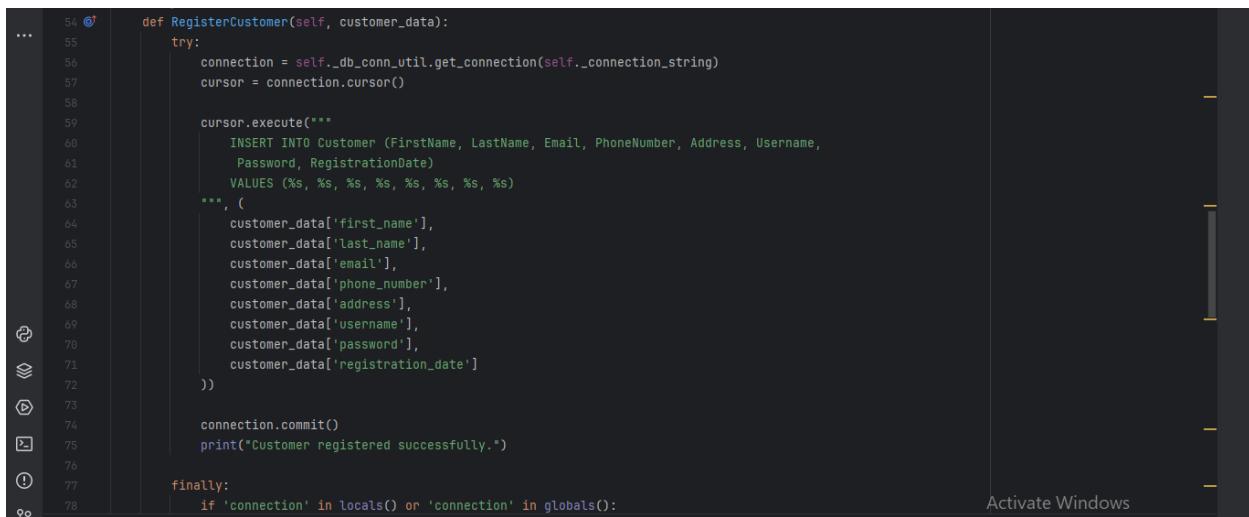
Run main x

C:\Users\welcome\Desktop\Assignment-python\venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\CaseStudy_01\main\main.py

Customer Data: {'customer_id': 2, 'first_name': 'Priya', 'last_name': 'Sharma', 'email': 'priya.sharma@example.com', 'phone_number': '8765432109', 'address': '123 Main Street', 'username': 'priya_sharma', 'password': 'new_password', 'registration_date': datetime.date(2022, 1, 1)}

Process finished with exit code 0

RegisterCustomer

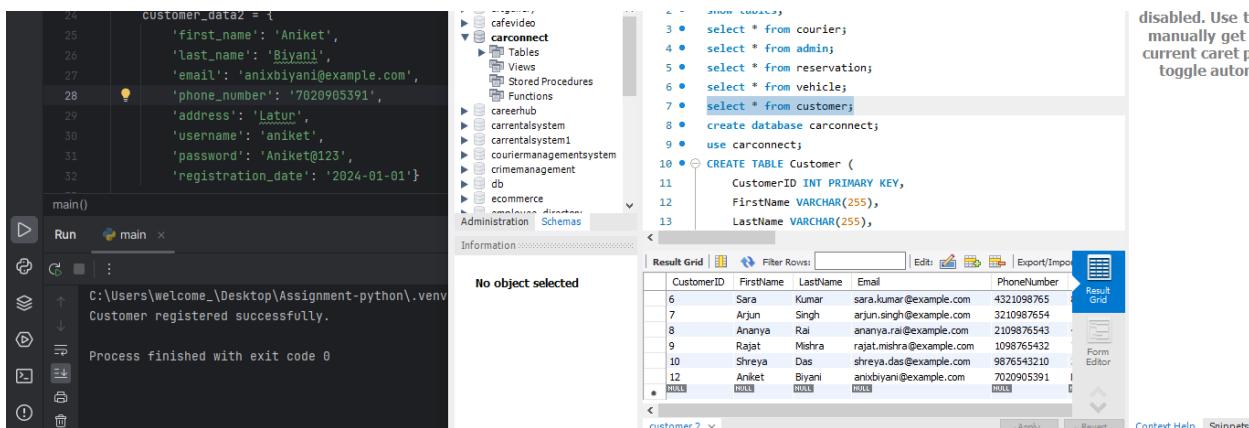


```
def RegisterCustomer(self, customer_data):
    ...
    try:
        connection = self._db_conn_util.get_connection(self._connection_string)
        cursor = connection.cursor()

        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']
                       ))
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.commit()
            print("Customer registered successfully.")
```

Activate Windows

OUTPUT:-



```
customer_data2 = {
    'first_name': 'Aniket',
    'last_name': 'Biyani',
    'email': 'anixbiyani@example.com',
    'phone_number': '7020905391',
    'address': 'Latun',
    'username': 'aniket',
    'password': 'Aniket@123',
    'registration_date': '2024-01-01'
}

main()
```

Run main x

C:\Users\welcome_\Desktop\Assignment-python\.venv
Customer registered successfully.

Process finished with exit code 0

Information

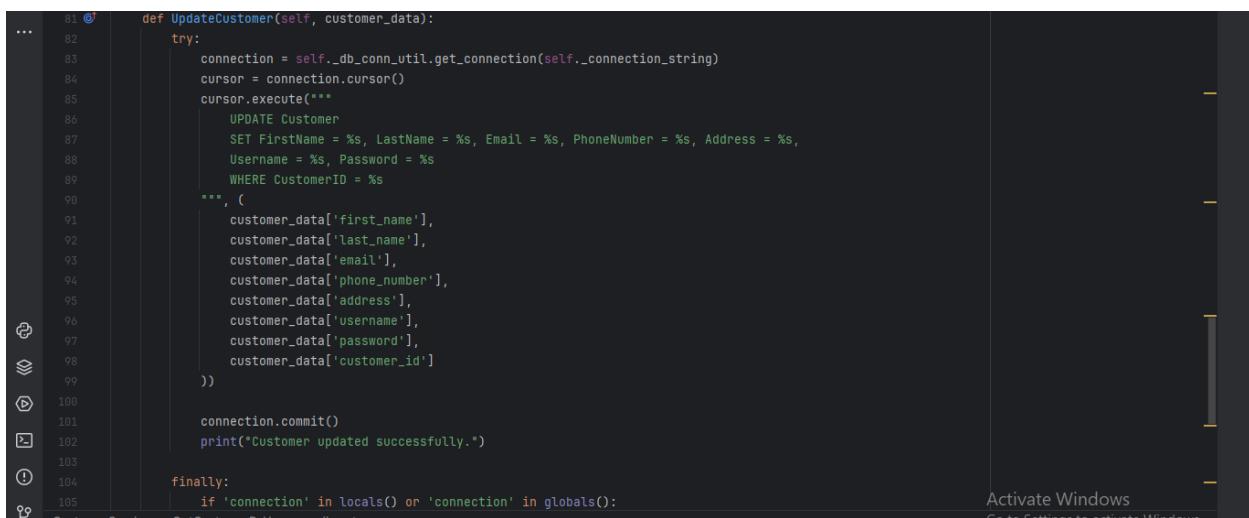
No object selected

Result Grid | Filter Rows: | Edit: | Export/Import | Result Grid | Form Editor | Context Help | Snippets

CustomerID	FirstName	LastName	Email	PhoneNumber
6	Sara	Kumar	sara.kumar@example.com	4321098765
7	Arjun	Singh	arjun.singh@example.com	3210987654
8	Ananya	Rai	ananya.rai@example.com	2109876543
9	Rajat	Mishra	rajat.mishra@example.com	1098765432
10	Shreya	Das	shreya.das@example.com	9876543210
12	Aniket	Biyani	anixbiyani@example.com	7020905391

disabled. Use t
manually get I
current caret p
toggle autor

UpdateCustomer,



```
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']
                       ))
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        if 'connection' in locals() or 'connection' in globals():
            connection.commit()
            print("Customer updated successfully.")
```

Activate Windows

Go to Settings to activate Windows

OUTPUT:-

The screenshot shows the PyCharm IDE. On the left, a code editor displays Python code for updating a customer in a database. On the right, a SQL interface shows the results of a query that includes creating a database, selecting from various tables, and finally selecting all columns from the 'customer' table. The result grid shows three rows of customer data.

```

updated_customer_data = {
    'customer_id': 12,
    'first_name': 'Abhi',
    'last_name': 'Surya',
    'email': 'abhi.surya@example.com',
    'phone_number': '766676743',
    'address': 'Mumbai',
    'username': 'abhi_102',
    'password': 'abhi@123',
    'registration_date': '2023-01-01'
}

customer_service.UpdateCustomer(updated_customer_data)

```

CustomerID	FirstName	LastName	Email	PhoneNumber
9	Rajat	Mishra	rajat.mishra@example.com	1098765432
10	Shreya	Das	shreya.das@example.com	9876543210
12	Abhi	Surya	abhi.surya@example.com	766676743
*	NULL	NULL	NULL	NULL

DeleteCustomer

The screenshot shows the PyCharm IDE. On the left, a code editor displays Python code for deleting a customer by ID. On the right, a SQL interface shows the results of a query that includes creating a database, selecting from various tables, and finally selecting all columns from the 'customer' table. The result grid shows three rows of customer data, identical to the previous update output.

```

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:-

The screenshot shows the PyCharm IDE. On the left, a code editor displays Python code for deleting a customer by ID. On the right, a SQL interface shows the results of a query that includes creating a database, selecting from various tables, and finally selecting all columns from the 'customer' table. The result grid shows three rows of customer data, identical to the previous update output.

```

customer_service.DeleteCustomer(12)

Customer with ID 12 deleted successfully.

Process finished with exit code 0

```

• *VehicleService (implements IVehicleService)*

• Methods:

GetVehicleById,

The screenshot shows the PyCharm IDE. On the left, a code editor displays Python code for getting a vehicle by its ID. On the right, a SQL interface shows the results of a query that includes creating a database, selecting from various tables, and finally selecting all columns from the 'vehicle' table. The result grid shows one row of vehicle data.

```

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:-

```
... 51     vehicle_service = VehicleService(db_conn_util,connection_string)
52
53     vehicle_id = 1
54     vehicle_data = vehicle_service.GetVehicleById(vehicle_id)
55     print("Vechicle Data:", vehicle_data)
56     ...
57
58 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,

```
... 26
27     def GetAvailableVehicles(self):
28         try:
29             connection = self._db_conn_util.get_connection(self._connection_string)
30             cursor = connection.cursor()
31
32             cursor.execute("SELECT * FROM Vehicle WHERE Availability>0")
33             available_vehicles = cursor.fetchall()
34
35             return available_vehicles
36
37         finally:
38             if 'connection' in locals() or 'connection' in globals():
39                 connection.close()
40
```

OUTPUT:-

```
... 57
58     available_vehicles=vehicle_service.GetAvailableVehicles()
59     print("Available Vehicles:")
60     for vehicle in available_vehicles:
61         print(vehicle)
62
63 main() > for vehicle in available_veh...
```

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,

```
... 41     def AddVehicle(self, vehicle_data):
42         try:
43             connection = self._db_conn_util.get_connection(self._connection_string)
44             cursor = connection.cursor()
45
46             cursor.execute("""
47                 INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)
48                 VALUES (%s, %s, %s, %s, %s, %s, %s)
49             """, (
50                 vehicle_data['model'],
51                 vehicle_data['make'],
52                 vehicle_data['year'],
53                 vehicle_data['color'],
54                 vehicle_data['registration_number'],
55                 vehicle_data['availability'],
56                 vehicle_data['daily_rate']
57             ))
58
59             connection.commit()
60             print("Vehicle added successfully.")
61
62         finally:
63             if 'connection' in locals() or 'connection' in globals():
64                 connection.close()
```

Activate Windows

OUTPUT:-

The screenshot shows the PyCharm IDE interface. On the left, the code editor displays Python code for adding a new vehicle to a service. In the center, the Schemas browser shows the database structure with a tree view of tables, views, stored procedures, functions, and system objects. On the right, the SQL tool window displays a history of executed SQL statements, including the creation of the database and tables, and the insertion of vehicle data. The bottom right corner shows the result grid for the last query, listing vehicles with their details.

```
63     new_vehicle_data = {
64         'model': 'Sedan',
65         'make': 'Toyota',
66         'year': 2022,
67         'color': 'Blue',
68         'registration_number': 'ABC123',
69         'availability': True,
70         'daily_rate': 50.0
71     }
72     vehicle_service.AddVehicle(new_vehicle_data)
```

main()

Run main

C:\Users\welcome\Desktop\Assignment-python\.venv\Script
Vehicle added successfully.

Schemas

- Filter objects
- artgallery
- cafedivas
- carconnect**
- Tables
- Views
- Stored Procedures
- Functions
- carhub
- carrentalsystem
- carrentalsystem1
- Administration
- Schemas

Information

No object selected

Result Grid Filter Rows: Edits Export/Import Result Grid

VehicleID	Model	Make	Year	Color	RegistrationNumber	Availability
8	Sentra	Nissan	2021	Yellow	STU234	0
9	Focus	Ford	2022	Orange	VWA567	1
10	Cruze	Chevrolet	2017	Brown	YZA890	1
12	Sedan	Toyota	2022	Blue	ABC123	1
*	ROLE	ROLE	ROLE	ROLE	ROLE	ROLE

UpdateVehicle,

The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical toolbar with various icons: a file icon, a copy/paste icon, a search icon, a refresh icon, a dropdown menu, and a help icon. The main area contains Python code for updating a vehicle record in a database. The code uses a cursor object to execute an UPDATE query with multiple SET clauses. It then commits the transaction and prints a success message. Finally, it checks if the connection is still open and closes it if necessary. The code is numbered from 65 to 92.

```
65     def UpdateVehicle(self, vehicle_data):
66         try:
67             connection = self._db_conn_util.get_connection(self._connection_string)
68             cursor = connection.cursor()
69             cursor.execute("""
70                 UPDATE Vehicle
71                 SET Model = %s, Make = %s, Year = %s, Color = %s,
72                     | RegistrationNumber = %s, Availability = %s, DailyRate = %s
73                 WHERE VehicleID = %s
74             """, (
75                 vehicle_data['model'],
76                 vehicle_data['make'],
77                 vehicle_data['year'],
78                 vehicle_data['color'],
79                 vehicle_data['registration_number'],
80                 vehicle_data['availability'],
81                 vehicle_data['daily_rate'],
82                 vehicle_data['vehicle_id']
83             ))
84
85             connection.commit()
86             print("Vehicle updated successfully.")
87
88         finally:
89             if 'connection' in locals() or 'connection' in globals():
90                 connection.close()
```

OUTPUT:-

The screenshot shows the PyCharm IDE interface. On the left, a code editor displays Python code for updating vehicle data in a database:

```
updated_vehicle_data = {
    'vehicle_id': 12,
    'model': 'swift',
    'make': 'maruti',
    'year': 2021,
    'color': 'Grey',
    'registration_number': 'BM4683',
    'availability': True,
    'daily_rate': 00.0
}
vehicle_service.UpdateVehicle(updated_vehicle_data)
```

Below the code editor is a 'Run' toolbar with a 'main()' button.

The right side of the interface shows the Database tool window. The 'Schemas' tab is selected, displaying the following database structure:

- artgallery
- cafedivas
- carconnect** (selected)
- Tables
- Views
- Stored Procedures
- Functions

Other tabs in the Database window include 'careerhub', 'couriermanagementsystem', 'currentplacement', 'Administration', and 'Schemas'.

The 'SQL File 2' tab contains the following SQL queries:

- USE couriermanagementsystem;
- show tables;
- select * from courier;
- select * from admin;
- select * from reservation;
- select * from vehicle;** (highlighted in blue)
- select * from customer;
- create database carconnect;
- use carconnect;

The results grid shows the following data for the vehicle table:

Result Grid	VehicleID	Model	Make	Year	Color	RegistrationNumber	Availability
8	Sentra	Nissan	2021	Yellow	STU234	0	
9	Focus	Ford	2022	Orange	VWX567	1	
10	Cruze	Chevrolet	2017	Brown	YZA890	1	
12	swift	maruti	2021	Grey	BM4683	1	
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL

Buttons for 'Edit', 'Filter Rows', 'Export/Import', and 'Result Grid' are visible at the bottom of the results grid.

RemoveVehicle

```

 93     def RemoveVehicle(self, vehicle_id):
 94         try:
 95             connection = self._db_conn_util.get_connection(self._connection_string)
 96             cursor = connection.cursor()
 97
 98             cursor.execute("DELETE FROM Vehicle WHERE VehicleID = %s", (vehicle_id,))
 99
100             connection.commit()
101             print("Vehicle removed successfully.")
102
103         finally:
104             if 'connection' in locals() or 'connection' in globals():
105                 connection.close()
106

```

Run main x

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:-

87 vehicle_service.RemoveVehicle[12]

main()

Run main x

C:\Users\welcome_\Desktop\Assignment-python

Vehicle removed successfully.

Process finished with exit code 0

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

- **ReservationService (implements IReservationService):**

- Methods:

GetReservationById,

```

 1 from .IReservationService import IReservationService
 2 from CaseStudy_01.exception.Exception import ReservationNotFoundException
 3 from decimal import Decimal
 4
 5 class ReservationService(IReservationService):
 6     def __init__(self, db_conn_util, connection_string):
 7         self._db_conn_util = db_conn_util
 8         self._connection_string = connection_string
 9
10     def GetReservationById(self, reservation_id):
11         try:
12             connection = self._db_conn_util.get_connection(self._connection_string)
13             cursor = connection.cursor()
14
15             cursor.execute("SELECT * FROM Reservation WHERE ReservationID = %s", (reservation_id,))
16             reservation_data = cursor.fetchone()
17
18             if reservation_data:
19                 return reservation_data
20             else:
21                 raise ReservationNotFoundException(f"Reservation with ID {reservation_id} not found.")
22
23         finally:
24             if 'connection' in locals() or 'connection' in globals():

```

OUTPUT:-

```
90     reservation_service = ReservationService(db_conn_util, connection_string)
91     |
92     reservation_id = 1
93
94     reservation_data = reservation_service.GetReservationById(reservation_id)
95     print("Reservation Data:", reservation_data)
96     """
97
98 main()
99
100 Run main x
101 C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
102 Reservation Data: (1, 1, 1, datetime.datetime(2024, 1, 1, 16, 25), datetime.datetime(2024, 1, 6, 16, 25), Decimal('200.00'), 'Cancelled')
```

GetReservationsByCustomerId,

```
... 26 def GetReservationsByCustomerId(self, customer_id):
27     try:
28         connection = self._db_conn_util.get_connection(self._connection_string)
29         cursor = connection.cursor()
30
31         cursor.execute("SELECT * FROM Reservation WHERE CustomerID = %s", (customer_id,))
32         reservations_data = cursor.fetchall()
33
34         if reservations_data:
35             return reservations_data
36         else:
37             raise ReservationNotFoundException(f"No reservations found for Customer ID {customer_id}.")
38
39     finally:
40         if 'connection' in locals() or 'connection' in globals():
41             connection.close()
```

OUTPUT:-

```
97     customer_id = 1
98     reservations_data = reservation_service.GetReservationsByCustomerId(customer_id)
99     print("Reservations Data:", reservations_data)
100    """
101
102 main()
103 Run main x
104 C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
105 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,

```
... 43 def CreateReservation(self, reservationData):
44     try:
45         connection = self._db_conn_util.get_connection(self._connection_string)
46         cursor = connection.cursor()
47
48         cursor.execute("""
49             INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)
50             VALUES (%s, %s, %s, %s, %s, %s)
51         """, (
52             reservationData['customer_id'],
53             reservationData['vehicle_id'],
54             reservationData['start_date'],
55             reservationData['end_date'],
56             Decimal(reservationData['total_cost']),
57             reservationData['status']
58         ))
59
60         connection.commit()
61         print("Reservation created successfully.")
62
63     finally:
64         if 'connection' in locals() or 'connection' in globals():
65             connection.close()
```

OUTPUT:-

The screenshot shows the PyCharm IDE interface. On the left, a code editor displays Python code for creating a reservation. On the right, a SQL query tool window shows a list of reservations with columns: ReservationID, CustomerID, VehicleID, StartDate, and EndDate.

```

101     reservation_data = {
102         'customer_id': 1,
103         'vehicle_id': 1,
104         'start_date': datetime.now(),
105         'end_date': datetime.now() + timedelta(days=1),
106         'total_cost': Decimal('150.00'),
107         'status': 'Confirmed'
108     }
109
110     reservation_service.CreateReservation(reservation_data)
111
112
113
114
115
116
117
118
119
120
121
122
    
```

ReservationID	CustomerID	VehicleID	StartDate	EndDate
9	9	8	2022-09-15 16:30:00	2022-09-18 11:15:00
10	10	10	2022-10-01 10:00:00	2022-10-03 17:30:00
11	1	1	2024-01-01 16:21:49	2024-01-04 16:21:46
12	1	1	2024-01-01 21:24:48	2024-01-04 21:24:46

UpdateReservation,

The screenshot shows the PyCharm IDE interface. On the left, a code editor displays Python code for updating a reservation. On the right, a SQL query tool window shows a list of reservations with columns: ReservationID, CustomerID, VehicleID, StartDate, and EndDate.

```

67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
    
```

ReservationID	CustomerID	VehicleID	StartDate	EndDate
9	9	8	2022-09-15 16:30:00	2022-09-18 11:15:00
10	10	10	2022-10-01 10:00:00	2022-10-03 17:30:00
11	1	1	2024-01-01 16:21:49	2024-01-04 16:21:46
12	1	1	2024-01-01 21:26:27	2024-01-06 21:26:27

OUTPUT:-

The screenshot shows the PyCharm IDE interface. On the left, a code editor displays Python code for updating a reservation. On the right, a SQL query tool window shows a list of reservations with columns: ReservationID, CustomerID, VehicleID, StartDate, and EndDate.

```

112
113
114
115
116
117
118
119
120
121
122
    
```

ReservationID	CustomerID	VehicleID	StartDate	EndDate
9	9	8	2022-09-15 16:30:00	2022-09-18 11:15:00
10	10	10	2022-10-01 10:00:00	2022-10-03 17:30:00
11	1	1	2024-01-01 16:21:49	2024-01-04 16:21:46
12	1	1	2024-01-01 21:26:27	2024-01-06 21:26:27

CancelReservation

```
98 ⑨    def CancelReservation(self, reservation_id):
99        try:
100            connection = self._db_conn_util.get_connection(self._connection_string)
101            cursor = connection.cursor()
102
103            cursor.execute("""
104                UPDATE Reservation
105                SET Status = 'Cancelled'
106                WHERE ReservationID = %s
107            """, (reservation_id,))
108
109            connection.commit()
110            print(f"Reservation with ID {reservation_id} has been cancelled.")
111
112        finally:
113            if 'connection' in locals() or 'connection' in globals():
114                connection.close()
```

OUTPUT:-

```
123
124     reservation_id_to_cancel = 12
125
126     reservation_service.CancelReservation(reservation_id_to_cancel)
127
main()
```

Run main ×

Run 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,

```
1  from .IAdminService import IAdminService
2  from CaseStudy_01.exception.Exception import AdminNotFoundException
...
8
9
10 ⑩    class AdminService(IAdminService):
11        def __init__(self, db_conn_util, connection_string):
12            self._db_conn_util = db_conn_util
13            self._connection_string = connection_string
14
15        def GetAdminById(self, admin_id):
16            try:
17                connection = self._db_conn_util.get_connection(self._connection_string)
18                cursor = connection.cursor()
19
20                cursor.execute("SELECT * FROM Admin WHERE AdminID = %s", (admin_id,))
21                admin_data = cursor.fetchone()
22
23                if admin_data:
24                    return admin_data
25                else:
26                    raise AdminNotFoundException(f"Admin with ID {admin_id} not found.")
27
28            finally:
29                if 'connection' in locals() or 'connection' in globals():
30                    connection.close()
```

Activate Windows

OUTPUT:-

```
128     admin_service = AdminService(db_conn_util,connection_string)
129     admin_id_to_fetch = 1
130
131     admin_data = admin_service.GetAdminById(admin_id_to_fetch)
132     print("Admin Data:", admin_data)
133
134
135 main()
136
137 Run  main x
138 C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
139 Admin Data: (1, 'UpdatedAdmin', 'UpdatedUser', 'updated_admin@example.com', '9876543210', 'john_admin', 'new_admin_password', 'Super Admin', datetime.date(2024, 1, 1))
```

GetAdminByUsername,

```
1 usage
2 def GetAdminByUsername(self, username):
3     try:
4         connection = self._db_conn_util.get_connection(self._connection_string)
5         cursor = connection.cursor()
6
7         cursor.execute("SELECT * FROM Admin WHERE Username = %s", (username,))
8         admin_data = cursor.fetchone()
9
10        if admin_data:
11            return admin_data
12        else:
13            raise AdminNotFoundException(f"Admin with username {username} not found.")
14
15    finally:
16        if 'connection' in locals() or 'connection' in globals():
17            connection.close()
```

OUTPUT:-

```
136     admin_username_to_fetch = "john_admin"
137
138     admin_data = admin_service.GetAdminByUsername(admin_username_to_fetch)
139     print("Admin Data:", admin_data)
140
141 main()
142
143 Run  main x
144 C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome_\Desktop\Assignment-python\CaseStudy_01\main\main.py
145 Admin Data: (1, 'UpdatedAdmin', 'UpdatedUser', 'updated_admin@example.com', '9876543210', 'john_admin', 'new_admin_password', 'Super Admin', datetime.date(2024, 1, 1))
```

RegisterAdmin,

```
44 def RegisterAdmin(self, admin_data):
45     try:
46         connection = self._db_conn_util.get_connection(self._connection_string)
47         cursor = connection.cursor()
48
49         cursor.execute("""
50             INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)
51             VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
52         """,
53         (
54             admin_data['first_name'],
55             admin_data['last_name'],
56             admin_data['email'],
57             admin_data['phone_number'],
58             admin_data['username'],
59             admin_data['password'],
60             admin_data['role'],
61             admin_data['join_date']
62         ))
63
64         connection.commit()
65         print("Admin registered successfully.")
66
67     finally:
68         if 'connection' in locals() or 'connection' in globals():
69             connection.close()
```

Activate Windows

OUTPUT:-

The screenshot shows a PyCharm IDE window on the left with Python code for registering an admin. The code defines a function `admin_data_to_register` with fields like first_name, last_name, email, phone_number, username, and password. A `main()` function calls this and prints a success message. Below the code is a terminal window showing the output: "Admin registered successfully.". To the right is a SQL interface window titled "couriermanagementsystem". It contains a query window with the following SQL statements:

```
1 • USE couriermanagementsystem;
2 • show tables;
3 • select * from courier;
4 • select * from admin;
5 • select * from reservation;
6 • select * from vehicle;
7 • select * from customer;
8 • create database carconnect;
9 • use carconnect;
```

Below the query window is a "Result Grid" table with the following data:

AdminID	FirstName	LastName	Email	PhoneNum
9	Rajat	Mishra	rajat.mishra@admin.com	109876543
10	Shreya	Das	shreya.das@admin.com	987654321
12	Admin	User	admin@example.com	123456789

UpdateAdmin,

The screenshot shows a PyCharm IDE window on the left with Python code for updating an admin. The code defines a function `UpdateAdmin` that takes `admin_data` as input. It uses a try-finally block to handle database connections. Inside the try block, it executes an UPDATE query on the Admin table with values from `admin_data`. After committing the transaction, it prints a success message. In the finally block, it closes the connection if it's still open. Below the code is a terminal window showing the output: "Admin updated successfully.". To the right is a SQL interface window titled "couriermanagementsystem". It contains a query window with the following SQL statements:

```
1 • USE couriermanagementsystem;
2 • show tables;
3 • select * from courier;
4 • select * from admin;
5 • select * from reservation;
6 • select * from vehicle;
7 • select * from customer;
8 • create database carconnect;
9 • use carconnect;
```

Below the query window is a "Result Grid" table with the following data:

AdminID	FirstName	LastName	Email	PhoneNum
9	Rajat	Mishra	rajat.mishra@admin.com	109876543
10	Shreya	Das	shreya.das@admin.com	987654321
12	UpdatedAdmin	UpdatedUser	updated_admin@example.com	987654321

OUTPUT:-

The screenshot shows a PyCharm IDE window on the left with Python code for updating an admin. The code defines a function `admin_data_to_update` with fields like admin_id, first_name, last_name, email, phone_number, and password. A `main()` function calls this and prints a success message. Below the code is a terminal window showing the output: "Admin updated successfully.". To the right is a SQL interface window titled "couriermanagementsystem". It contains a query window with the following SQL statements:

```
1 • USE couriermanagementsystem;
2 • show tables;
3 • select * from courier;
4 • select * from admin;
5 • select * from reservation;
6 • select * from vehicle;
7 • select * from customer;
8 • create database carconnect;
9 • use carconnect;
```

Below the query window is a "Result Grid" table with the following data:

AdminID	FirstName	LastName	Email	PhoneNum
9	Rajat	Mishra	rajat.mishra@admin.com	109876543
10	Shreya	Das	shreya.das@admin.com	987654321
12	UpdatedAdmin	UpdatedUser	updated_admin@example.com	987654321

DeleteAdmin

```
1 Usage
95     def DeleteAdmin(self, admin_id):
96         try:
97             connection = self._db_conn_util.get_connection(self._connection_string)
98             cursor = connection.cursor()
99
100            cursor.execute("DELETE FROM Admin WHERE AdminID = %s", (admin_id,))
101
102            connection.commit()
103            print("Admin deleted successfully..")
104
105        finally:
106            if 'connection' in locals() or 'connection' in globals():
107                connection.close()
```

OUTPUT:-

The code editor shows the execution of the DeleteAdmin function with admin_id_to_delete set to 12. The terminal output shows "Admin deleted successfully.". To the right, a database interface for "carconnect" is shown, displaying tables like admin, reservation, vehicle, and customer. A query window shows the following SQL statements:

```
4 • select * from admin;
5 • select * from reservation;
6 • select * from vehicle;
7 • select * from customer;
8 • create database carconnect;
9 • use carconnect;
```

A result grid shows the following data:

AdminID	FirstName	LastName	Email	PhoneNum
8	Ananya	Rai	ananya.rai@admin.com	2109876543
9	Rajat	Mishra	rajat.mishra@admin.com	109876543
10	Shreya	Das	shreya.das@admin.com	987654321
*	NULL	NULL	NULL	NULL

- AuthenticationService:

- A class responsible for handling user authentication.

```
1 from CaseStudy_01.exception.Exception import AuthenticationException
2 from CaseStudy_01.dao.CustomerService import CustomerService
3 from CaseStudy_01.dao.AdminService import AdminService
4 from CaseStudy_01.util.DBConnUtil import DBConnUtil
...
2 usages
6 class AuthenticationService:
7     @staticmethod
8     def authenticate_customer(username, password, connection_string):
9         try:
10
11             customer_service = CustomerService(DBConnUtil(), connection_string)
12
13             customer_data = customer_service.GetCustomerByUsername(username)
14
15             if customer_data and customer_data['Password'] == password:
16                 print("Authentication successful.")
17                 return customer_data
18             else:
19                 raise AuthenticationException("Invalid customer credentials")
20
21         except AuthenticationException as e:
22             print(f"Error during customer authentication: {str(e)}")
23             raise
```

OUTPUT:-

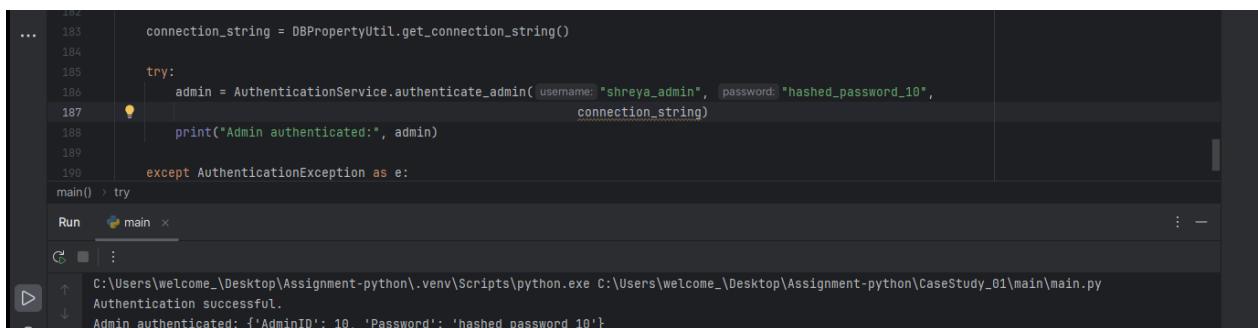
The code editor shows the execution of the authenticate_customer method with username "john_doe" and password "new_password". The terminal output shows "Authentication successful." and "Customer authenticated: {'CustomerID': 1, 'Password': 'new_password'}".

```

1 usage
25 @staticmethod
26 def authenticate_admin(username, password, connection_string):
27     try:
28         admin_service = AdminService(DBConnUtil(), connection_string)
29
30         admin_data = admin_service.GetAdminByUsername(username)
31
32         if admin_data and admin_data['Password'] == password:
33             print("Authentication successful.")
34             return admin_data
35         else:
36             raise AuthenticationException("Invalid admin credentials")
37
38     except AuthenticationException as e:
39         print(f"Error during admin authentication: {str(e)}")
40         raise

```

OUTPUT:-



```

...
183     connection_string = DBPropertyUtil.get_connection_string()
184
185     try:
186         admin = AuthenticationService.authenticate_admin(
187             username="shreya_admin",
188             password="hashed_password_10",
189             connection_string)
190
191     except AuthenticationException as e:
192         print(e)
193
194     main() > try
195
196     Run main ×
197
198     C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe C:\Users\welcome\Desktop\Assignment-python\CaseStudy_01\main\main.py
199     Admin authenticated.
200     Admin authenticated: {'AdminID': 10, 'Password': 'hashed_password_10'}

```

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

```

00 3 usages
...
2 class ReportGenerator:
3     1 usage
4     @staticmethod
5     def generate_reservation_report(reservation_data):
6         print("Reservation Report:")
7         print("Reservation ID | Customer ID | Vehicle ID | Start Date | End Date | Amount | Status")
8         for reservation in reservation_data:
9             print(
10                 f"{reservation[0]:<15} | {reservation[1]:<12} | {reservation[2]:<11} | "
11                 f"{reservation[3]:<11} | {reservation[4]:<9} | {reservation[5]:<6} | {reservation[6]}")
12
13     1 usage
14     @staticmethod
15     def generate_vehicle_report(vehicle_data):
16         print("\nVehicle Report:")
17         print("Vehicle ID | Model | Brand | Year | Status")
18         for vehicle in vehicle_data:
19             print(f" {vehicle[0]:<11} | {vehicle[1]:<5} | {vehicle[2]:<6} | {vehicle[3]:<4} | {vehicle[4]}")

```

```

196
197     reservation_service = ReservationService(db_conn_util, connection_string)
198     vehicle_service = VehicleService(db_conn_util, connection_string)
199
200     reservation_data = reservation_service.GetReservationsList()
201     vehicle_data = vehicle_service.GetAvailableVehicles()
202
203     ReportGenerator.generate_reservation_report(reservation_data)
204     ReportGenerator.generate_vehicle_report(vehicle_data)

```

OUTPUT:-

The screenshot shows a terminal window with two tables of data:

Reservation Report:

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

Activate Windows
Go to Settings to activate Windows.

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.

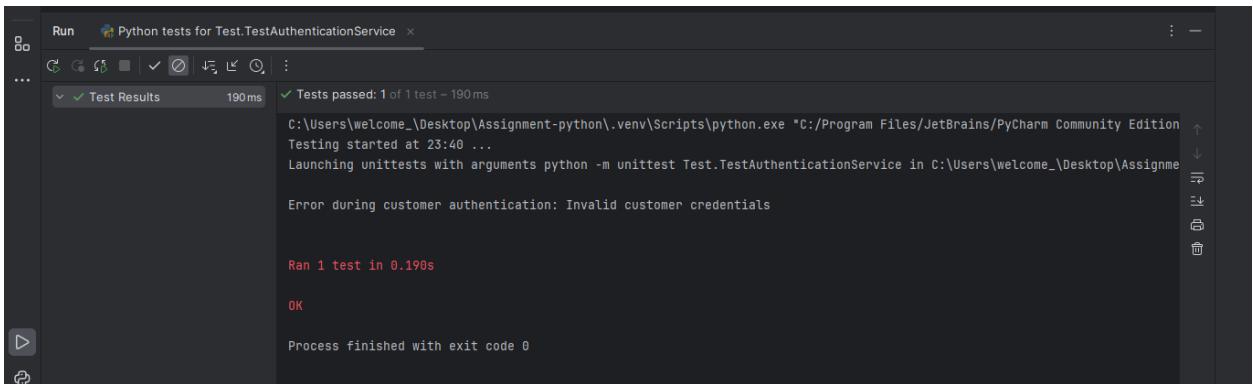
```
import unittest
from unittest.mock import MagicMock
from CaseStudy_01.dao.AuthenticationService import AuthenticationService
from CaseStudy_01.exception.Exception import AuthenticationException

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:-



The screenshot shows the PyCharm interface with the 'Run' tab selected. The 'Test Results' section indicates 1 test passed in 190ms. The log output shows:

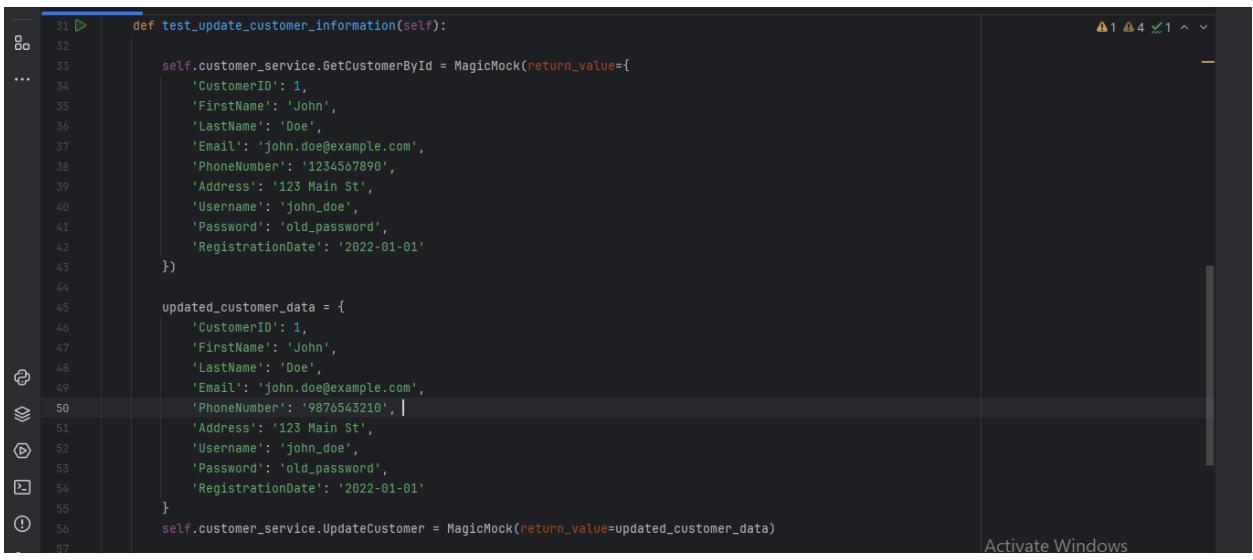
```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition/.../bin/python" C:/Program Files/JetBrains/PyCharm Community Edition/.../bin/unittest Test.TestAuthenticationService in C:\Users\welcome_\Desktop\Assignment...
Testing started at 23:40 ...
Launching unittests with arguments python -m unittest Test.TestAuthenticationService in C:\Users\welcome_\Desktop\Assignment...
Error during customer authentication: Invalid customer credentials

Ran 1 test in 0.190s

OK

Process finished with exit code 0
```

Test updating customer information.

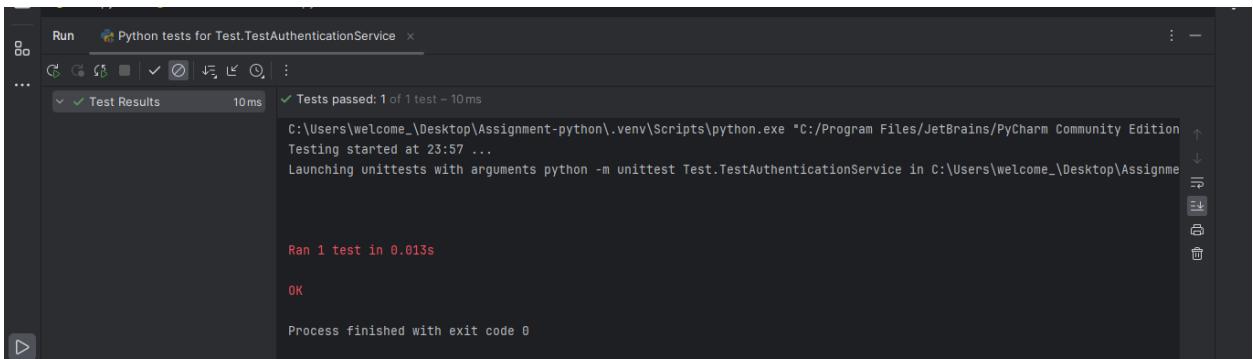


The screenshot shows the PyCharm code editor with the following Python test code:

```
31 def test_update_customer_information(self):
32     ...
33     self.customer_service.GetCustomerById = MagicMock(return_value={
34         'CustomerID': 1,
35         'FirstName': 'John',
36         'LastName': 'Doe',
37         'Email': 'john.doe@example.com',
38         'PhoneNumber': '1234567890',
39         'Address': '123 Main St',
40         'Username': 'john_doe',
41         'Password': 'old_password',
42         'RegistrationDate': '2022-01-01'
43     })
44
45     updated_customer_data = {
46         'CustomerID': 1,
47         'FirstName': 'John',
48         'LastName': 'Doe',
49         'Email': 'john.doe@example.com',
50         'PhoneNumber': '9876543210',
51         'Address': '123 Main St',
52         'Username': 'john_doe',
53         'Password': 'old_password',
54         'RegistrationDate': '2022-01-01'
55     }
56     self.customer_service.UpdateCustomer = MagicMock(return_value=updated_customer_data)
```

Activate Windows

OUTPUT:-



The screenshot shows the PyCharm interface with the 'Run' tab selected. The 'Test Results' section indicates 1 test passed in 10ms. The log output shows:

```
C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition/.../bin/python" C:/Program Files/JetBrains/PyCharm Community Edition/.../bin/unittest Test.TestAuthenticationService in C:\Users\welcome_\Desktop\Assignment...
Testing started at 23:57 ...
Launching unittests with arguments python -m unittest Test.TestAuthenticationService in C:\Users\welcome_\Desktop\Assignment...
Ran 1 test in 0.013s

OK

Process finished with exit code 0
```

3. Test adding a new vehicle

```
80  def test_add_new_vehicle(self):
...  ...
64  ...
65  ...
66  ...
67  ...
68  ...
69  ...
70  ...
71  ...
72  ...
73  ...
74  ...
75  ...
76  ...
77  ...
78  ...
79  ...
80  if __name__ == '__main__':
81      unittest.main()
```

OUTPUT:-

The screenshot shows the PyCharm interface during a test run. The top bar says "Run Python tests for Test.TestAuthenticationService.test_a...". The "Test Results" tab is selected, showing a green checkmark and the message "Tests passed: 1 of 1 test - 8ms". Below this, the terminal output shows the test command and its execution: "C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe *C:/Program Files/JetBrains/PyCharm Community Edition Testing started at 00:00 ... Launching unittests with arguments python -m unittest Test.TestAuthenticationService.test_add_new_vehicle in C:\Users\welcome_\Desktop\Assignment-python\src". The output then shows the test results: "Ran 1 test in 0.009s" and "OK". At the bottom, it says "Process finished with exit code 0".

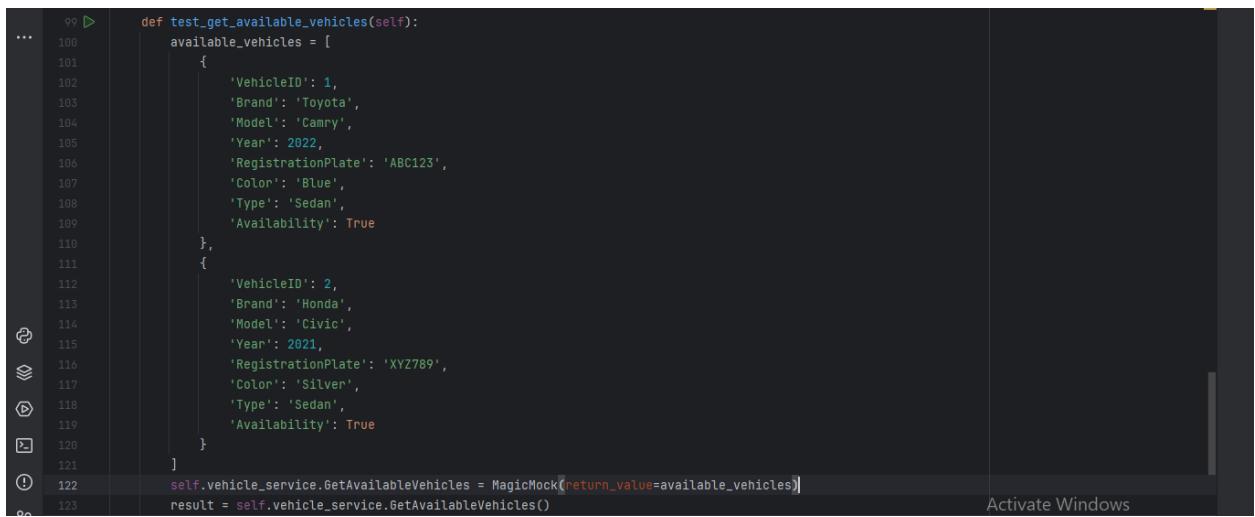
4. Test updating vehicle details.

```
82  def test_update_vehicle_details(self):
83      ...
84      ...
85      ...
86      ...
87      ...
88      ...
89      ...
90      ...
91      ...
92      ...
93      ...
94      ...
95      ...
96      ...
97  if __name__ == '__main__':
98      unittest.main()
```

OUTPUT:-

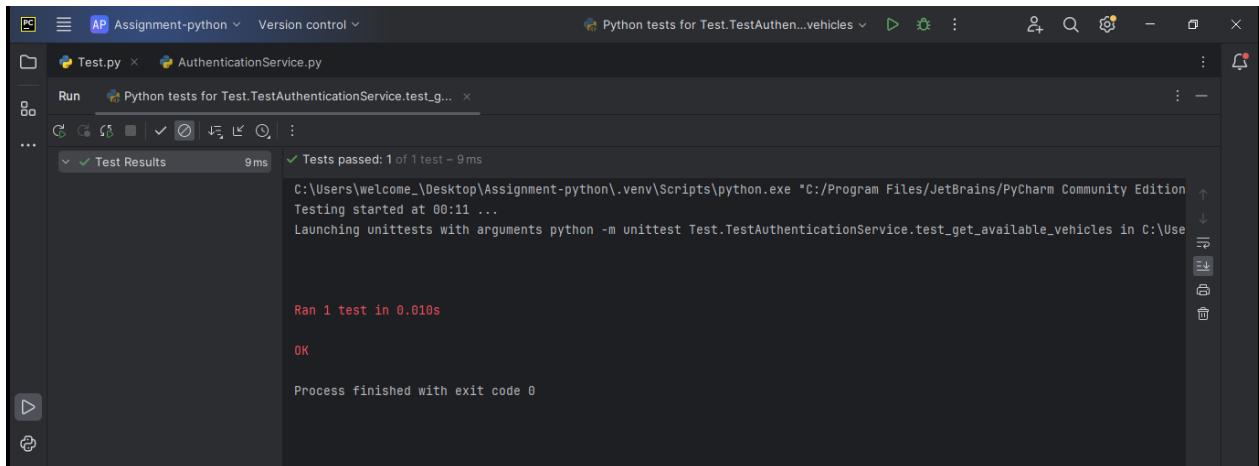
The screenshot shows the PyCharm interface during a test run. The top bar says "Run Python tests for Test.TestAuthenticationService". The "Test Results" tab is selected, showing a green checkmark and the message "Tests passed: 1 of 1 test - 10ms". Below this, the terminal output shows the test command and its execution: "C:\Users\welcome_\Desktop\Assignment-python\.venv\Scripts\python.exe *C:/Program Files/JetBrains/PyCharm Community Edition Testing started at 00:03 ... Launching unittests with arguments python -m unittest Test.TestAuthenticationService in C:\Users\welcome_\Desktop\Assignment-python\src". The output then shows the test results: "Ran 1 test in 0.012s" and "OK". At the bottom, it says "Process finished with exit code 0".

5. Test getting a list of available vehicles.



```
99 def test_get_available_vehicles(self):
100     ...
101     available_vehicles = [
102         {
103             'VehicleID': 1,
104             'Brand': 'Toyota',
105             'Model': 'Camry',
106             'Year': 2022,
107             'RegistrationPlate': 'ABC123',
108             'Color': 'Blue',
109             'Type': 'Sedan',
110             'Availability': True
111         },
112         {
113             'VehicleID': 2,
114             'Brand': 'Honda',
115             'Model': 'Civic',
116             'Year': 2021,
117             'RegistrationPlate': 'XYZ789',
118             'Color': 'Silver',
119             'Type': 'Sedan',
120             'Availability': True
121     ]
122     self.vehicle_service.GetAvailableVehicles = MagicMock(return_value=available_vehicles)
123     result = self.vehicle_service.GetAvailableVehicles()
```

OUTPUT:-



Assignment-python Version control Python tests for Test.TestAuthen...vehicles

Run Python tests for Test.TestAuthenticationService.test_g...

Test Results 9 ms

Tests passed: 1 of 1 test – 9 ms

C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition"

Testing started at 00:11 ...

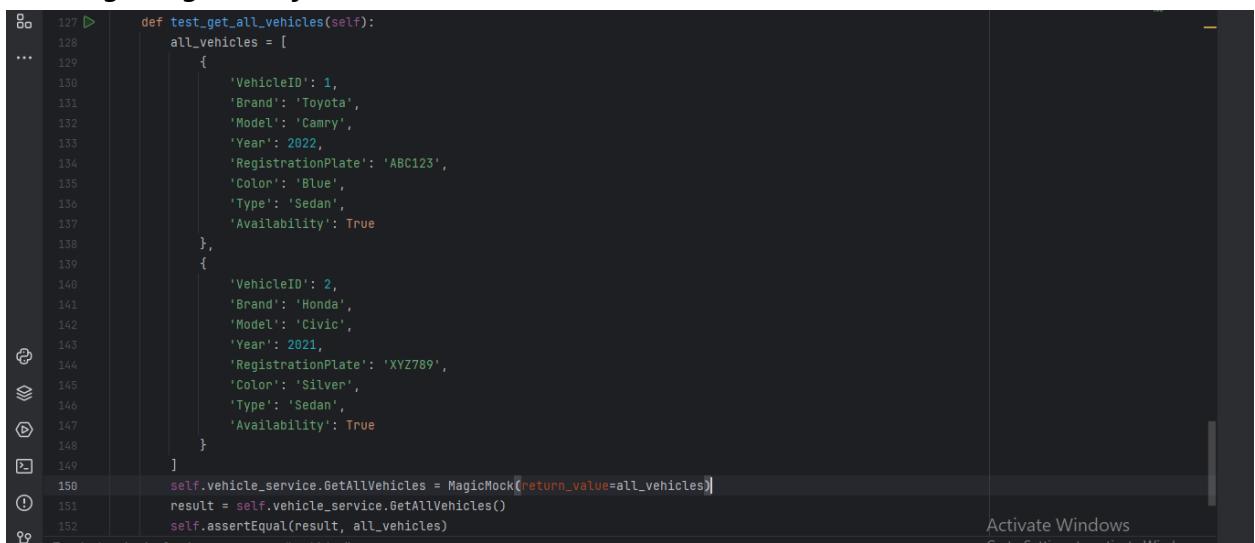
Launching unittests with arguments python -m unittest Test.TestAuthenticationService.test_get_available_vehicles in C:\Use

Ran 1 test in 0.010s

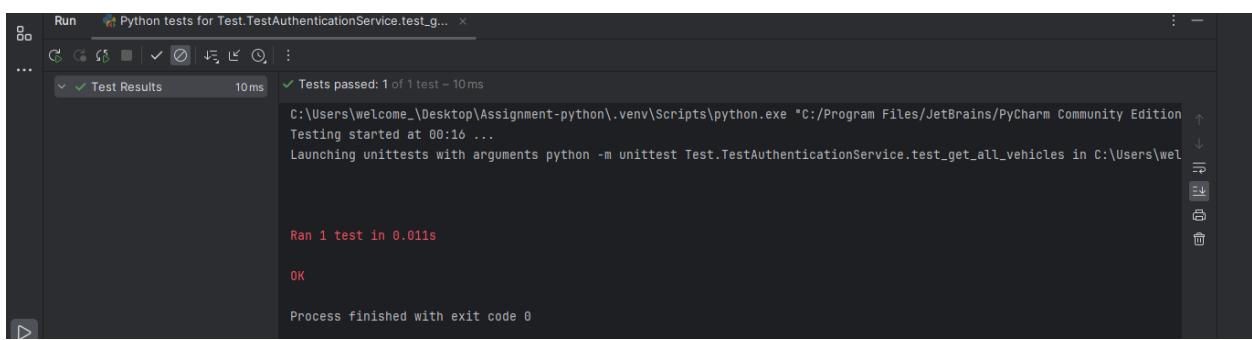
OK

Process finished with exit code 0

6. Test getting a list of all vehicles.



```
127 def test_get_all_vehicles(self):
128     ...
129     all_vehicles = [
130         {
131             'VehicleID': 1,
132             'Brand': 'Toyota',
133             'Model': 'Camry',
134             'Year': 2022,
135             'RegistrationPlate': 'ABC123',
136             'Color': 'Blue',
137             'Type': 'Sedan',
138             'Availability': True
139         },
140         {
141             'VehicleID': 2,
142             'Brand': 'Honda',
143             'Model': 'Civic',
144             'Year': 2021,
145             'RegistrationPlate': 'XYZ789',
146             'Color': 'Silver',
147             'Type': 'Sedan',
148             'Availability': True
149     ]
150     self.vehicle_service.GetAllVehicles = MagicMock(return_value=all_vehicles)
151     result = self.vehicle_service.GetAllVehicles()
152     self.assertEqual(result, all_vehicles)
```

OUTPUT:-

The screenshot shows the PyCharm Run tool window with the title "Python tests for Test.TestAuthenticationService.test_g...". The status bar indicates "Tests passed: 1 of 1 test - 10ms". The output pane displays the following text:

```
C:\Users\welcome\Desktop\Assignment-python\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2021.1.1\helpers\pycharm\testrunner\runner.py" -m unittest Test.TestAuthenticationService.test_get_all_vehicles in C:\Users\welcome\Desktop\Assignment-python\src\test\python\test\AuthenticationService.py
Testing started at 00:16 ...
Launching umittests with arguments python -m unittest Test.TestAuthenticationService.test_get_all_vehicles in C:\Users\welcome\Desktop\Assignment-python\src\test\python\test\AuthenticationService.py

Ran 1 test in 0.011s

OK

Process finished with exit code 0
```