
Phase 5: Apex Programming (Developer)

Goal

Enable advanced customization in Salesforce using **Apex programming** to handle scenarios not possible with declarative automation. This includes writing triggers, classes, queries, and test code for maintaining data integrity and automating complex telecom workflows.

---

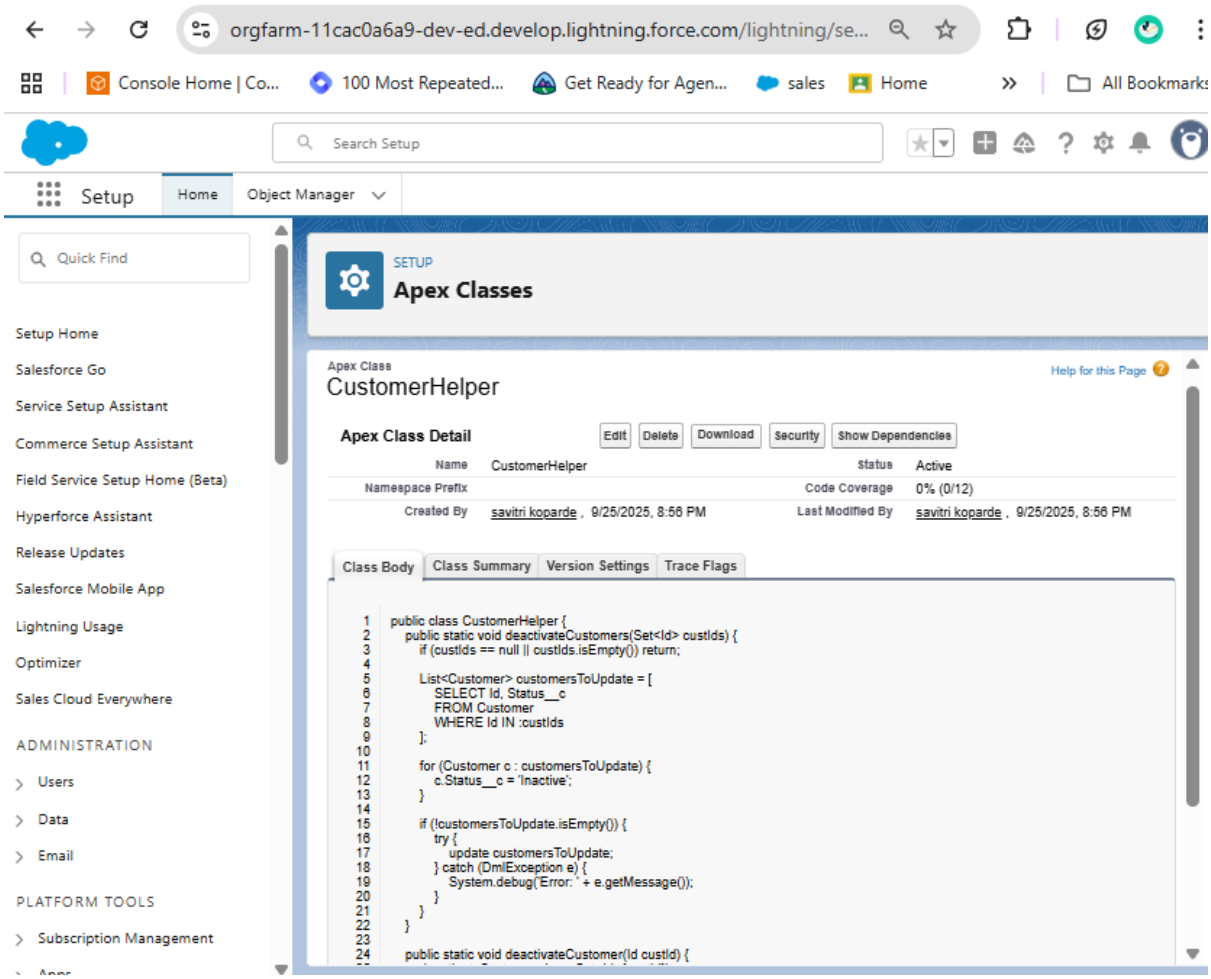
1. Apex Classes & Objects

Implementation:

- Created CustomerHelper.cls with reusable logic to update Customer status when a Service Request is processed.

```
public class CustomerHelper {
    public static void deactivateCustomer(Id custId) {
        try {
            Customer__c cust = [SELECT Id, Status__c FROM Customer__c WHERE Id = :custId LIMIT 1];
            cust.Status__c = 'Inactive';
            update cust;
        } catch (Exception e) {
            System.debug('Error while deactivating customer: ' + e.getMessage());
        }
    }
}
```

Purpose: Centralized business logic for reusability.



---

2. Apex Trigger with Design Pattern

Implementation:

- Created trigger ServiceRequestTrigger on ServiceRequest__c.
- Delegated logic to ServiceRequestTriggerHandler.cls.

Trigger:

```
trigger ServiceRequestTrigger on ServiceRequest__c (after insert, after update) {
    if(Trigger.isAfter && Trigger.isInsert) {
        ServiceRequestTriggerHandler.handleAfterInsert(Trigger.new);
    }
}
```

Handler Class:

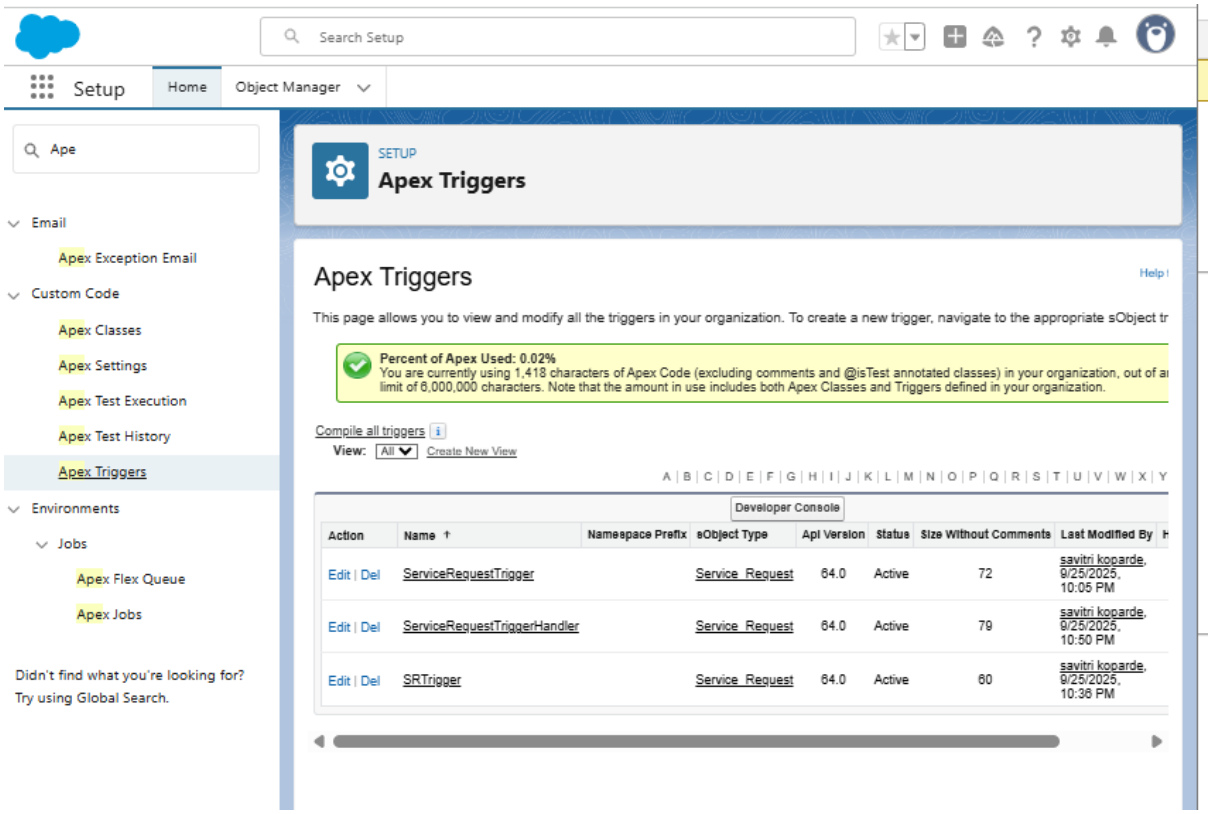
```
public class ServiceRequestTriggerHandler {
    public static void handleAfterInsert(List<ServiceRequest__c> newReqs) {
```

```
        Set<Id> custIds = new Set<Id>();

        for(ServiceRequest__c req : newReqs) {
            if(req.Request_Type__c == 'Cancellation' && req.Customer__c != null) {
                custIds.add(req.Customer__c);
            }
        }

        if(!custIds.isEmpty()) {
            for(Id cId : custIds) {
                CustomerHelper.deactivateCustomer(cId);
            }
        }
    }
}
```

Purpose: Updates related Customer to “Inactive” when a cancellation request is created.



3. SOQL Query

- Used in CustomerHelper.cls to fetch Customer details:

```
Customer__c cust = [SELECT Id, Status__c FROM Customer__c WHERE Id = :custId LIMIT 1];
```

Purpose: Retrieve customer data from Salesforce for processing.

---

4. Collections (List, Set, Map)

- **Set**: custIds ensures only unique Customers are updated.
- **List**: Trigger passes list of new ServiceRequests.
- **Map**: Could be used to map Customer Id → Customer record for bulk operations.

Example:

```
Map<Id, Customer__c> custMap = new Map<Id, Customer__c>(
   [SELECT Id, Status__c FROM Customer__c WHERE Id IN :custIds]
);
```

---

5. Control Statements

- **If condition** checks request type.
- **For loop** iterates over trigger records.

Example:

```
for(ServiceRequest__c req : newReqs) {
   if(req.Request_Type__c == 'Cancellation') {
      // logic
   }
}
```

---

6. Exception Handling

- Used try-catch in CustomerHelper.cls:

```
try {
   update cust;
} catch (Exception e) {
   System.debug('Error: ' + e.getMessage());
}
```

Purpose: Ensures errors don’t break transaction flow.

---

7. Test Class

Implementation:

```
@isTest
public class TestServiceRequestTrigger {
    @isTest
    static void testCancellationFlow() {
        // Create test customer
        Customer__c cust = new Customer__c(
            Name = 'Test Customer',
            Email__c = 'test@test.com',
            SIM_Number__c = '12345',
            Status__c = 'Active'
        );
        insert cust;

        // Create cancellation request
        ServiceRequest__c req = new ServiceRequest__c(
            Request_Type__c = 'Cancellation',
            Customer__c = cust.Id,
            Status__c = 'New'
        );
        insert req;

        // Fetch updated customer
        Customer__c updatedCust = [SELECT Status__c FROM Customer__c WHERE Id = :cust.Id];
        System.assertEquals('Inactive', updatedCust.Status__c);
    }
}
```

Purpose: Validates that trigger and helper class logic works correctly. Achieves code coverage required for deployment.
