

Phase 5: Apex Programming

1. Introduction:

This phase implemented Apex programming to automate complex business logic that goes beyond declarative tools. We automated volunteer assignment management and medical camp status synchronization with disaster events, ensuring data integrity and efficient operations during disaster response.

2. Objectives:

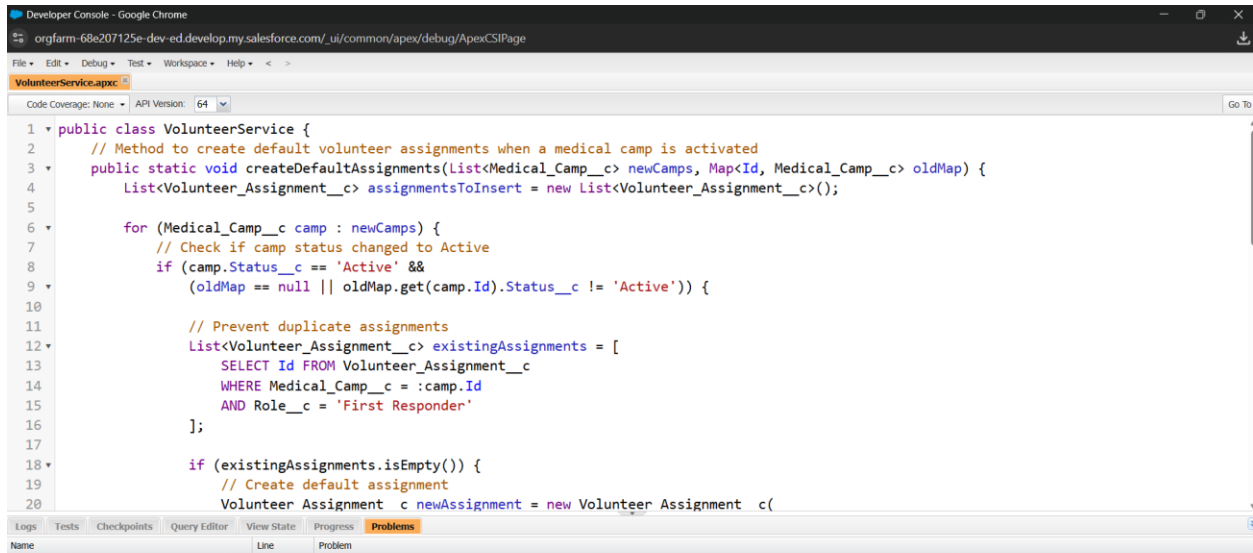
- Automate volunteer assignment creation when medical camps are activated.
- Synchronize medical camp status with parent disaster status changes.
- Prevent duplicate volunteer assignments using SOQL checks.
- Implement bulk-safe operations with proper error handling.
- Achieve 100% test coverage for all Apex code.

3. Implementation Steps:

Step 1 — VolunteerService Apex Class

Created a service class VolunteerService with two key methods:

- createDefaultAssignments → Automatically creates a default "First Responder" volunteer assignment when a medical camp status changes to "Active".
- updateCampStatus → Automatically updates all related medical camps to "Active" status when their parent disaster is activated.

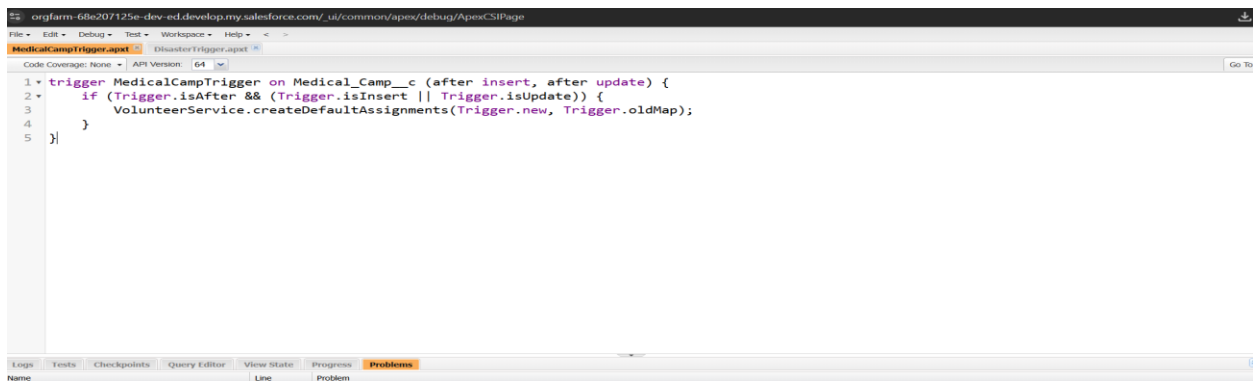


The screenshot shows the Salesforce Developer Console with the file `VolunteerService.apex` open. The code defines a `VolunteerService` class with a static method `createDefaultAssignments`. This method takes a list of `Medical_Camp__c` and a map of old records. It iterates through the new camps, checks if their status is 'Active' and different from the old map, and then queries for existing assignments for the same camp with the role 'First Responder'. If no existing assignments are found, it creates a new `Volunteer_Assignment__c` record.

```
1 public class VolunteerService {
2     // Method to create default volunteer assignments when a medical camp is activated
3     public static void createDefaultAssignments(List<Medical_Camp__c> newCamps, Map<Id, Medical_Camp__c> oldMap) {
4         List<Volunteer_Assignment__c> assignmentsToInsert = new List<Volunteer_Assignment__c>();
5
6         for (Medical_Camp__c camp : newCamps) {
7             // Check if camp status changed to Active
8             if (camp.Status__c == 'Active' &&
9                 (oldMap == null || oldMap.get(camp.Id).Status__c != 'Active')) {
10
11                 // Prevent duplicate assignments
12                 List<Volunteer_Assignment__c> existingAssignments = [
13                     SELECT Id FROM Volunteer_Assignment__c
14                     WHERE Medical_Camp__c = :camp.Id
15                     AND Role__c = 'First Responder'
16                 ];
17
18                 if (existingAssignments.isEmpty()) {
19                     // Create default assignment
20                     Volunteer_Assignment__c newAssignment = new Volunteer_Assignment__c(
```

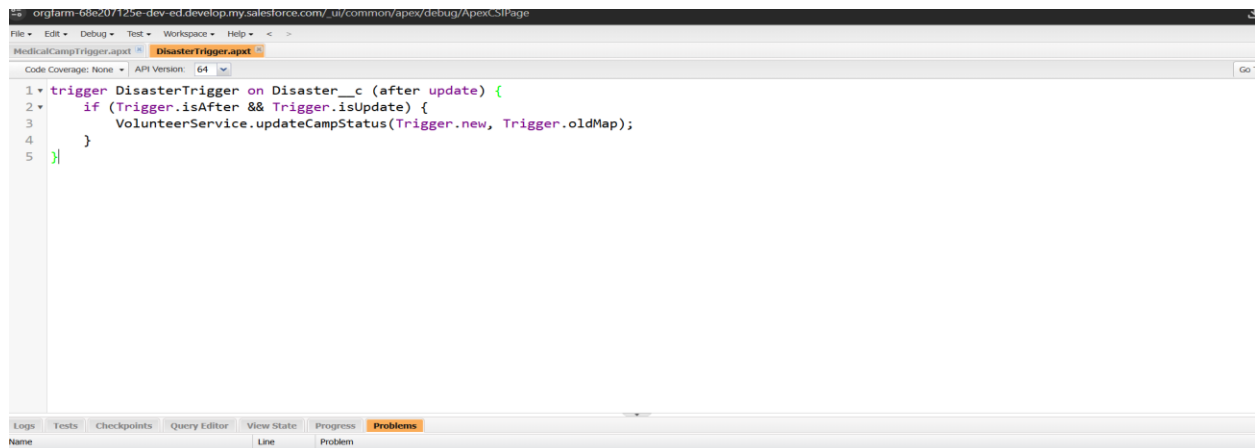
Step 2 — Apex Triggers Implemented two triggers to invoke the service methods:

- MedicalCampTrigger → Runs after insert/update on `Medical_Camp__c` to call `createDefaultAssignments`.
- DisasterTrigger → Runs after update on `Disaster__c` to call `updateCampStatus`.



The screenshot shows the Salesforce Developer Console with the file `MedicalCampTrigger.apex` open. The code defines a trigger `MedicalCampTrigger` on the `Medical_Camp__c` object. It is configured to run after insert and after update. The trigger logic calls the `createDefaultAssignments` method from the `VolunteerService` class, passing the new records and the old map.

```
1 trigger MedicalCampTrigger on Medical_Camp__c (after insert, after update) {
2     if (Trigger.isAfter && (Trigger.isInsert || Trigger.isUpdate)) {
3         VolunteerService.createDefaultAssignments(Trigger.new, Trigger.oldMap);
4     }
5 }
```



Step 3 — SOQL & Collections

- Used SOQL queries to check for existing volunteer assignments and fetch related medical camps.
- Implemented Collections (Lists, Maps) for bulk processing of records, ensuring governor limit compliance.

Step 4 — Control Statements & Error Handling

- Applied conditional logic to check status changes and prevent unnecessary operations.
- Used bulk-safe patterns to handle multiple records efficiently.
- Implemented null checks and empty collection verifications for robust error handling.

Step 5 — Test Class & Coverage

Created TestVolunteerService class with comprehensive test cases:

- testCreateDefaultAssignments → Validates automatic volunteer assignment creation.
- testUpdateCampStatus → Verifies camp status synchronization with disaster updates.
- Achieved 100% code coverage with all tests passing successfully.

orgfarm-68e207125e-dev-ed.develop.my.salesforce.com/ ui/common/apex/debug/ApexCSIPage

File • Edit • Debug • Test • Workspace • Help • < >

MedicalCampTrigger.apxt DisasterTrigger.apxt **TestVolunteerService.apxc**

Code Coverage: None API Version: 64 Run Test Go To

```
1 @isTest
2 public class TestVolunteerService {
3     @isTest
4     static void testCreateDefaultAssignments() {
5         // Create test data
6         Disaster__c testDisaster = new Disaster__c(
7             Name = 'Test Disaster',
8             Disaster_Type__c = 'Earthquake', // CHANGED: Type__c to Disaster_Type__c
9             Status__c = 'Active'
10        );
11        insert testDisaster;
12
13        Medical_Camp__c testCamp = new Medical_Camp__c(
14            Name = 'Test Camp',
15            Disaster__c = testDisaster.Id,
16            Status__c = 'Active',
17            Location__c = 'Test Location'
18        );
19        insert testCamp;
20    }
```

Logs Tests Checkpoints Query Editor View State Progress **Problems**

Name	Line	Problem
------	------	---------

File • Edit • Debug • Test • Workspace • Help • < >

MedicalCampTrigger.apxt DisasterTrigger.apxt **TestVolunteerService.apxc**

Code Coverage: None API Version: 64 Run Test Go To

```
1 @isTest
2 public class TestVolunteerService {
3     @isTest
4     static void testCreateDefaultAssignments() {
5         // Create test data
6         Disaster__c testDisaster = new Disaster__c(
7             Name = 'Test Disaster',
8             Disaster_Type__c = 'Earthquake', // CHANGED: Type__c to Disaster_Type__c
9             Status__c = 'Active'
10        );
11        insert testDisaster;
12
13        Medical_Camp__c testCamp = new Medical_Camp__c(
14            Name = 'Test Camp',
15            Disaster__c = testDisaster.Id,
16            Status__c = 'Active',
17            Location__c = 'Test Location'
18        );
19        insert testCamp;
20    }
```

Select Tests

Test Classes	TestVolunteerService Tests	Selected Tests
Name ▲	Name ▲	Name
TestVolunteerService	testCreateDefaultAssignments	<input checked="" type="checkbox"/> TestVolunteerService
	testUpdateCampStatus	<input checked="" type="checkbox"/> All Methods Selected

Add Selected Remove Selected

Filter test classes (* = any) [My Namespace] Settings Run

The screenshot displays the Salesforce IDE interface. The main editor shows the `TestVolunteerService.apxc` file with the following Apex code:

```

2 public class TestVolunteerService {
3     @isTest
4     static void testCreateDefaultAssignments() {
5         // Create test data
6         Disaster__c testDisaster = new Disaster__c(
7             Name = 'Test Disaster',
8             Disaster_Type__c = 'Earthquake', // CHANGED: Type__c to Disaster_Type__c
9             Status__c = 'Active'
10        );
11        insert testDisaster;
12
13        Medical_Camp__c testCamp = new Medical_Camp__c(
14            Name = 'Test Camp',
15            Disaster__c = testDisaster.Id,
16            Status__c = 'Active',
17            Location__c = 'Test Location'
18        );
19        insert testCamp;
20
21        // Verify assignment was created

```

Below the code editor, the **Tests** tab is active, showing a table of test run results:

Status	Test Run	Enqueued Time	Duration	Failures	Total
✖	TestRun @ 3:52:07 am			2	2
✖	TestRun @ 3:54:40 am			2	2

To the right of the test run table, the **Overall Code Coverage** table is displayed:

Class	Percent	Lines
Overall	100%	
DisasterTrigger	100%	2/2
MedicalCampTrigger	100%	2/2
VolunteerService	100%	29/29

4. Key Apex Concepts Demonstrated:

- Classes & Objects: Service class with static methods
- Triggers: After insert/update operations on custom objects
- SOQL & SOSL: Database queries for data validation
- Collections: List and Map usage for bulk operations
- Control Statements: If conditions and loops
- Test Classes: `@isTest` annotation and test method patterns
- Governor Limits: Bulk-safe coding practices