

SalesForce Developer Catalyst Project Document

Name: C. ADEEBA KISHWAR

Email: 209X1A0514@GPREC.AC.IN

MODULE: APEX TRIGGERS

Challenge - Get started with Apex Triggers

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    List<Account> acct = new List<Account>();
    for(Account a: Trigger.new){
        if( a.Match_Billing_Address__c == true && a.BillingPostalCode!=null ){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

Challenge - Bulk Apex Triggers

trigger ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

    List<Task> taskList = new List<Task>();

    for(Opportunity opp : Trigger.new) {

        if(Trigger.isInsert) {

            if(Opp.StageName == 'Closed Won') {

                taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));

            }

        }

        if(Trigger.isUpdate) {

            if(Opp.StageName == 'Closed Won' && Opp.StageName!=Trigger.oldMap.get(opp.Id).StageName) {
```

```

        taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
    }
} }

if(taskList.size()>0) {
    insert taskList;
}
}

```

MODULE: APEX TESTING

Challenge - Get Started with Apex Unit Tests

VerifyDate

```

public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {

        //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the
month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
    private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }

        //check that date2 is within (>=) 30 days of date1
    }
}

```

```

        Date date30Days = date1.addDays(30); //create a date 30 days away from date1

        if( date2 >= date30Days ) { return false; }

        else { return true; }

    }

    private static Date SetEndOfMonthDate(Date date1) {

        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());

        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);

        return lastDay;

    }

}

```

TestVerifyDate

```

@isTest

public class TestVerifyDate
{
    static testMethod void testMethod1()
    {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);

        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);

    }

}

```

Challenge - Test Apex Triggers

RestrictContactByName

```

trigger RestrictContactByName on Contact (before insert, before update) {

    For (Contact c : Trigger.New) {

        if(c.LastName == 'INVALIDNAME') { //invalidname is invalid

            c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

        }

    }

}

```

```
}  
}
```

TestRestrictContactByName

@isTest

```
private class TestRestrictContactByName {  
  
    static testMethod void metodoTest()  
    {  
        List<Contact> listContact= new List<Contact>();  
  
        Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' , email='Test@test.com');  
  
        Contact c2 = new Contact(FirstName='Francesco1', LastName =  
'INVALIDNAME',email='Test@test.com');  
  
        listContact.add(c1);  
  
        listContact.add(c2);  
  
        Test.startTest();  
  
        try  
        {  
            insert listContact;  
        }  
  
        catch(Exception ee)  
        {  
        }  
  
        Test.stopTest();  
    }  
}
```

Challenge - Create Test Data for Apex Test

RandomContactFactory class

```
public class RandomContactFactory {
```

```

public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String FName) {
    List<Contact> contactList = new List<Contact>();

    for(Integer i=0;i<numContactsToGenerate;i++) {
        Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
        contactList.add(c);
        System.debug(c);
    }
    //insert contactList;
    System.debug(contactList.size());
    return contactList;
}
}

```

MODULE : ASYNCHRONOUS APEX

Challenge - Use future methods

AccountProcessor

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
            account.Number_of_Contacts__c = [Select count() from Contact Where AccountId =: account.Id];
            System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
    }
}

```

```

    }

    update updatedAccounts;
}
}

```

AccountProcessorTest

```

@Test
public class AccountProcessorTest {

    @Test
    public static void testNoOfContacts(){

        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName = 'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();

        AccountProcessor.countContacts(acctIds);

        Test.stopTest();
    }
}

```

Challenge - Use Batch Apex

LeadProcessor

```
global class LeadProcessor implements Database.Batchable<sObject> {

    global Integer count = 0;

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }

    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count);
    }
}
```

LeadProcessorTest

```
@isTest

public class LeadProcessorTest {

    @isTest
```

```

public static void testit(){
    List<lead> L_list = new List<lead>();

    for(Integer i=0; i<200; i++){
        Lead l = new lead();
        l.LastName = 'name' + i;
        l.Company = 'Company';
        l.Status = 'Random Status';
        L_list.add(l);
    }
    insert L_list;

    Test.startTest();
    LeadProcessor lp = new LeadProcessor();
    Id batchId = Database.executeBatch(lp);
    Test.stopTest();
}
}

```

Challenge - Control Processes with Queueable Apex

AddPrimaryContact

```

public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
}

```



```
}
```

```
public void execute(QueueableContext context){  
    List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)  
        from Account where BillingState = :state Limit 200];  
    List<Contact> primaryContacts = new List<Contact>();  
  
    for(Account acc:accounts){  
        Contact c = con.clone();  
        c.AccountId = acc.Id;  
        primaryContacts.add(c);  
    }  
  
    if(primaryContacts.size() > 0){  
        insert primaryContacts;  
    }  
  
}}
```

AddPrimaryContactTest

@isTest

```
public class AddPrimaryContactTest {  
  
    static testmethod void testQueueable(){  
        List<Account> testAccounts = new List<Account>();  
        for(Integer i=0;i<50;i++){  
            testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));  
        }  
        for(Integer j=0;j<50;j++){
```

```

        testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));
    }
    insert testAccounts;

    Contact testContact = new Contact(FirstName = 'John', LastName = 'Doe');
    insert testContact;

    AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

    Test.startTest();
    system.enqueueJob(addit);
    Test.stopTest();

    system.assertEquals(50, [Select count() from Contact where accountId in (Select Id from Account
    where BillingState = 'CA')]);
}
}

```

Challenge - Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```

global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {

        //Retrieving the 200 first leads where lead source is in blank.

        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = " LIMIT 200];
    }
}

```

```

        //Setting the LeadSource field the 'Dreamforce' value.
        for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }

        //Updating all elements in the list.
        update leads;
    }
}

```

DailyLeadProcessorTest

```

@Test
private class DailyLeadProcessorTest {
    @Test
    public static void testDailyLeadProcessor(){

        //Creating new 200 Leads and inserting them.
        List<Lead> leads = new List<Lead>();
        for (Integer x = 0; x < 200; x++) {
            leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
        }
        insert leads;

        //Starting test. Putting in the schedule and running the DailyLeadProcessor execute method.
        Test.startTest();

        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());

        Test.stopTest();
    }
}

```

```

        //Once the job has finished, retrieve all modified leads.

        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT
200];

        //Checking if the modified leads are the same size number that we created in the start of this
method.

        System.assertEquals(200, listResult.size());

    }
}

```

MODULE: APEX INTEGRATION SERVICES

Challenge - Apex REST Callouts

AnimalLocator

```

public class AnimalLocator{

    public static String getAnimalNameById(Integer x){

        Http http = new Http();

        HttpRequest req = new HttpRequest();

        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);

        req.setMethod('GET');

        Map<String, Object> animal= new Map<String, Object>();

        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {

            Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());

            animal = (Map<String, Object>) results.get('animal');

        }

        return (String)animal.get('name');
    }
}

```

```
}  
}
```

AnimalLocatorTest

@isTest

```
private class AnimalLocatorTest{  
    @isTest static void AnimalLocatorMock1() {  
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());  
        string result = AnimalLocator.getAnimalNameById(3);  
        String expectedResult = 'chicken';  
        System.assertEquals(result,expectedResult );  
    }  
}
```

AnimalLocatorMock

@isTest

```
global class AnimalLocatorMock implements HttpCalloutMock {  
    // Implement this interface method  
    global HTTPResponse respond(HTTPRequest request) {  
        // Create a fake response  
        HTTPResponse response = new HTTPResponse();  
        response.setHeader('Content-Type', 'application/json');  
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');  
        response.setStatusCode(200);  
        return response;  
    }  
}
```

Challenge - Apex SOAP Callouts

ParkLocator

```
public class ParkLocator {  
    public static string[] country(string theCountry) {  
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space  
        return parkSvc.byCountry(theCountry);  
    }  
}
```

ParkLocatorTest

@isTest

```
private class ParkLocatorTest {  
    @isTest static void testCallout() {  
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());  
        String country = 'United States';  
        List<String> result = ParkLocator.country(country);  
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};  
        System.assertEquals(parks, result);  
    }  
}
```

ParkServiceMock

@isTest

```
global class ParkServiceMock implements WebServiceMock {  
    global void doInvoke(  
        Object stub,  
        Object request,  
        Map<String, Object> response,  
        String endpoint,
```

```

        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {
    // start - specify the response you want to send
    ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
    response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
    // end

    response.put('response_x', response_x);
}
}

```

Challenge - Apex Web Services

AccountManager

```

@RestResource(urlMapping='/Accounts/*/contacts')

global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                        FROM Account WHERE Id = :accId];
        return acc;
    }
}

```

AccountManagerTest

```

@isTest
private class AccountManagerTest {

```

```

private static testMethod void getAccountTest1() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts' ;
    request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount != null);
    System.assertEquals('Test record', thisAccount.Name);
}

// Helper method
static Id createTestRecord() {
    // Create test record
    Account TestAcc = new Account(
        Name='Test record');
    insert TestAcc;
    Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
    return TestAcc.Id;
}
}

```


APEX SPECIALIST SUPERBADGE

Step 2: Automate Record Creation –

Trigger Maintenance Request

```
trigger MaintenanceRequest on Case (before update, after update) {  
    if(Trigger.isUpdate && Trigger.isAfter){  
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);  
    }  
}
```

Maintenance Request Helper

```
public with sharing class MaintenanceRequestHelper {  
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {  
        Set<Id> validIds = new Set<Id>();  
        For (Case c : updWorkOrders){  
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){  
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){  
                    validIds.add(c.Id);  
                }  
            }  
        }  
    }  
}
```

```
}
```

```
//When an existing maintenance request of type Repair or Routine Maintenance is closed,
```

```
//create a new maintenance request for a future routine checkup.
```

```
if (!validIds.isEmpty()){
```

```
    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,  
Equipment__r.Maintenance_Cycle__c,
```

```
                (SELECT Id,Equipment__c,Quantity__c FROM  
Equipment_Maintenance_Items__r)
```

```
                FROM Case WHERE Id IN :validIds]);
```

```
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
```

```
//calculate the maintenance request due dates by using the maintenance cycle defined on the  
related equipment records.
```

```
    AggregateResult[] results = [SELECT Maintenance_Request__c,  
                MIN(Equipment__r.Maintenance_Cycle__c)cycle  
                FROM Equipment_Maintenance_Item__c  
                WHERE Maintenance_Request__c IN :ValidIds GROUP BY  
Maintenance_Request__c];
```

```
    for (AggregateResult ar : results){
```

```
        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
```

```
    }
```

```
List<Case> newCases = new List<Case>();
```

```
for(Case cc : closedCases.values()){
```

```
    Case nc = new Case (
```

```
        ParentId = cc.Id,
```

```
        Status = 'New',
```

```
        Subject = 'Routine Maintenance',
```

```

        Type = 'Routine Maintenance',

        Vehicle__c = cc.Vehicle__c,

        Equipment__c =cc.Equipment__c,

        Origin = 'Web',

        Date_Reported__c = Date.Today() );

//If multiple pieces of equipment are used in the maintenance request,

//define the due date by applying the shortest maintenance cycle to today's date.

//If (maintenanceCycles.containsKey(cc.Id)){

    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

//} else {

    //  nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);

//}

    newCases.add(nc);

}

insert newCases;

List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();

for (Case nc : newCases){

    for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){

        Equipment_Maintenance_Item__c item = clonedListItem.clone();

        item.Maintenance_Request__c = nc.Id;

        clonedList.add(item);

    }

}

insert clonedList;

}

}

}

```

Step 3: Synchronize Salesforce data with an External System–

WarehouseCallOutService

```
public with sharing class WarehouseCalloutService implements Queueable {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //Write a class that makes a REST callout to an external warehouse system to get a list of equipment
that needs to be updated.

    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)

    public static void runWarehouseEquipmentSync(){

        System.debug('go into runWarehouseEquipmentSync');

        Http http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();

        System.debug(response.getStatusCode());

        if (response.getStatusCode() == 200){

            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());

            System.debug(response.getBody());

            //class maps the following fields:

            //warehouse SKU will be external ID for identifying which equipment records to update within
Salesforce

            for (Object jR : jsonResponse){
```

```

Map<String,Object> mapJson = (Map<String,Object>)jR;

Product2 product2 = new Product2();

//replacement part (always true),
product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

//cost
product2.Cost__c = (Integer) mapJson.get('cost');

//current inventory
product2.Current_Inventory__c = (Double) mapJson.get('quantity');

//lifespan
product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');

//maintenance cycle
product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');

//warehouse SKU
product2.Warehouse_SKU__c = (String) mapJson.get('sku');


product2.Name = (String) mapJson.get('name');
product2.ProductCode = (String) mapJson.get('_id');
product2List.add(product2);
}

if (product2List.size() > 0){
    upsert product2List;
    System.debug('Your equipment was synced with the warehouse one');
}
}
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
}

```

```

        System.debug('end runWarehouseEquipmentSync');
    }

}

```

In Anonymous window for WarehouseCallOutService :

```
System.enqueueJob(New WarehouseCalloutService());
```

Step 4 : Schedule Synchronization –

WarehouseSyncSchedule

```

global with sharing class WarehouseSyncSchedule implements Schedulable {

    // implement scheduled code here

    global void execute (SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

```

Step 5 : Test Automation Logic –

MaintenanceRequestHelper

```

public with sharing class MaintenanceRequestHelper {

    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){

            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

                    validIds.add(c.Id);

                }

            }

        }

    }

}

```

```

//When an existing maintenance request of type Repair or Routine Maintenance is closed,
//create a new maintenance request for a future routine checkup.

if (!validIds.isEmpty()){

    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                FROM Case WHERE Id IN :validIds]);

    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

    //calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.

    AggregateResult[] results = [SELECT Maintenance_Request__c,
                MIN(Equipment__r.Maintenance_Cycle__c)cycle
                FROM Equipment_Maintenance_Item__c
                WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

    for (AggregateResult ar : results){

        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }

    List<Case> newCases = new List<Case>();

    for(Case cc : closedCases.values()){

        Case nc = new Case (
            ParentId = cc.Id
            ,
            Status = 'New',
            Subject = 'Routine Maintenance',

```

```

        Type = 'Routine Maintenance',
        Vehicle__c = cc.Vehicle__c,
        Equipment__c = cc.Equipment__c,
        Origin = 'Web',
        Date_Reported__c = Date.Today()
    );

    //If multiple pieces of equipment are used in the maintenance request,
    //define the due date by applying the shortest maintenance cycle to today's date.
    //If (maintenanceCycles.containsKey(cc.Id)){
        nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
    //} else {
        // nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
    //}

    newCases.add(nc);
}

insert newCases;

List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
        Equipment_Maintenance_Item__c item = clonedListItem.clone();
        item.Maintenance_Request__c = nc.Id;
        clonedList.add(item);
    }
}

```



```
    }  
    insert clonedList;  
  }  
}  
}
```

MaintenanceRequestHelperTest

@isTest

public with sharing class MaintenanceRequestHelperTest{

// createVehicle

private static Vehicle__c createVehicle(){

Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');

return vehicle;

}

// createEquipment

private static Product2 createEquipment(){

product2 equipment = new product2(name = 'Testing equipment',

lifespan_months__c = 10,

maintenance_cycle__c = 10,

replacement_part__c = true);

return equipment;

}

// createMaintenanceRequest

private static Case createMaintenanceRequest(id vehicleId, id equipmentId){

```

        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);

    return cse;
}

// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id equipmentId,id
requestId){
    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
        Equipment__c = equipmentId,
        Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}

@Test
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;
    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;
}

```

```
Equipment_Maintenance_Item__c equipmentMaintenanceItem =  
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
```

```
insert equipmentMaintenanceItem;
```

```
test.startTest();
```

```
createdCase.status = 'Closed';
```

```
update createdCase;
```

```
test.stopTest();
```

```
Case newCase = [Select id, subject,
```

```
    type,
```

```
    Equipment__c,
```

```
    Date_Reported__c,
```

```
    Vehicle__c,
```

```
    Date_Due__c
```

```
from case
```

```
where status ='New'];
```

```
Equipment_Maintenance_Item__c workPart = [select id
```

```
    from Equipment_Maintenance_Item__c
```

```
    where Maintenance_Request__c =:newCase.Id];
```

```
list<case> allCase = [select id from case];
```

```
system.assert(allCase.size() == 2);
```

```
system.assert(newCase != null);
```

```
system.assert(newCase.Subject != null);
```

```
system.assertEquals(newCase.Type, 'Routine Maintenance');
```

```
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
```

```

SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);

SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}

```

@isTest

```

private static void testNegative(){

    Vehicle__C vehicle = createVehicle();

    insert vehicle;

    id vehicleId = vehicle.Id;


    product2 equipment = createEquipment();

    insert equipment;

    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);

    insert createdCase;

    Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(equipmentId,
createdCase.Id);

    insert workP;

    test.startTest();

    createdCase.Status = 'Working';

    update createdCase;

    test.stopTest();

    list<case> allCase = [select id from case];

    Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c = :createdCase.Id];

    system.assert(equipmentMaintenanceItem != null);

    system.assert(allCase.size() == 1);
}

```

```

@isTest

private static void testBulk(){

    list<Vehicle__C> vehicleList = new list<Vehicle__C>();

    list<Product2> equipmentList = new list<Product2>();

    list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemlist = new
list<Equipment_Maintenance_Item__c>();

    list<case> caseList = new list<case>();

    list<id> oldCaseIds = new list<id>();


    for(integer i = 0; i < 300; i++){

        vehicleList.add(createVehicle());

        equipmentList.add(createEquipment());

    }

    insert vehicleList;

    insert equipmentList;


    for(integer i = 0; i < 300; i++){

        caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));

    }

    insert caseList;


    for(integer i = 0; i < 300; i++){

        equipmentMaintenanceItemlist.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,
caseList.get(i).id));

    }

    insert equipmentMaintenanceItemlist;


    test.startTest();

```

```

for(case cs : caseList){
    cs.Status = 'Closed';
    oldCaseIds.add(cs.Id);
}

```

```

update caseList;

```

```

test.stopTest();

```

```

list<case> newCase = [select id
                      from case
                      where status ='New'];

```

```

list<Equipment_Maintenance_Item__c> workParts = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c in: oldCaseIds];

```

```

system.assert(newCase.size() == 300);

```

```

list<case> allCase = [select id from case];

```

```

system.assert(allCase.size() == 600);

```

```

}}

```

Trigger MaintenanceRequest

```

trigger MaintenanceRequest on Case (before update, after update) {

```

```

    if(Trigger.isUpdate && Trigger.isAfter){

```

```

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New

```

```

, Trigger.OldMap);

```

```

    }

```

```

}

```

Step 6 : Test Callout Logic –

WarehouseCallOutService

```
public with sharing class WarehouseCalloutService implements Queueable {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //Write a class that makes a REST callout to an external warehouse system to get a list of equipment
that //needs to be updated.

    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)

    public static void runWarehouseEquipmentSync(){

        System.debug('go into runWarehouseEquipmentSync');

        Http http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();

        System.debug(response.getStatusCode());

        if (response.getStatusCode() == 200){

            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());

            System.debug(response.getBody());

            //class maps the following fields:

            //warehouse SKU will be external ID for identifying which equipment records to update within
Salesforce

            for (Object jR : jsonResponse){

                Map<String,Object> mapJson = (Map<String,Object>)jR;
```

```

        Product2 product2 = new Product2();

        //replacement part (always true),
        product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

        //cost
        product2.Cost__c = (Integer) mapJson.get('cost');

        //current inventory
        product2.Current_Inventory__c = (Double) mapJson.get('quantity');

        //lifespan
        product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');

        //maintenance cycle
        product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');

        //warehouse SKU
        product2.Warehouse_SKU__c = (String) mapJson.get('sku');


        product2.Name = (String) mapJson.get('name');
        product2.ProductCode = (String) mapJson.get('_id');
        product2List.add(product2);
    }

    if (product2List.size() > 0){
        upsert product2List;

        System.debug('Your equipment was synced with the warehouse one');
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}

```



```
}  
  
}
```

WarehouseCallOutServiceTest

@IsTest

```
private class WarehouseCalloutServiceTest {  
    // implement your mock callout test here  
    @isTest  
    static void testWarehouseCallout() {  
        test.startTest();  
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());  
        WarehouseCalloutService.execute(null);  
        test.stopTest();  
  
        List<Product2> product2List = new List<Product2>();  
        product2List = [SELECT ProductCode FROM Product2];  
  
        System.assertEquals(3, product2List.size());  
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);  
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);  
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);  
    }  
}
```

WarehouseCalloutServiceMock

@isTest

```
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
```

```
// implement http mock callout

global static HttpResponse respond(HttpRequest request) {

    HttpResponse response = new HttpResponse();

    response.setHeader('Content-Type', 'application/json');

    response.setBody('[{ "_id": "55d66226726b611100aaf741", "replacement": false, "quantity": 5, "name": "Generator 1000 kW", "maintenanceperiod": 365, "lifespan": 120, "cost": 5000, "sku": "100003"}, { "_id": "55d66226726b611100aaf742", "replacement": true, "quantity": 183, "name": "Cooling Fan", "maintenanceperiod": 0, "lifespan": 0, "cost": 300, "sku": "100004"}, { "_id": "55d66226726b611100aaf743", "replacement": true, "quantity": 143, "name": "Fuse 20A", "maintenanceperiod": 0, "lifespan": 0, "cost": 22, "sku": "100005"}]');

    response.setStatusCode(200);

    return response;
}
}
```

Step 7 : Test Scheduling Logic –

WarehouseSyncScheduleTest

@isTest

```
public with sharing class WarehouseSyncScheduleTest {

    @isTest static void test() {

        String scheduleTime = '00 00 00 * * ? *';

        Test.startTest();

        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new WarehouseSyncSchedule());

        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];

        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');
```

```

        Test.stopTest();
    }
}

```

WarehouseSyncSchedule

```

global with sharing class WarehouseSyncSchedule implements Schedulable {

    // implement scheduled code here

    global void execute (SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

```

WarehouseCalloutServiceMock

```

@isTest

global class WarehouseCalloutServiceMock implements HttpCalloutMock {

    // implement http mock callout

    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type', 'application/json');

        response.setBody('[{ "_id": "55d66226726b611100aaf741", "replacement": false, "quantity": 5, "name": "Generator 1000 kW", "maintenanceperiod": 365, "lifespan": 120, "cost": 5000, "sku": "100003"}, { "_id": "55d66226726b611100aaf742", "replacement": true, "quantity": 183, "name": "Cooling Fan", "maintenanceperiod": 0, "lifespan": 0, "cost": 300, "sku": "100004"}, { "_id": "55d66226726b611100aaf743", "replacement": true, "quantity": 143, "name": "Fuse 20A", "maintenanceperiod": 0, "lifespan": 0, "cost": 22, "sku": "100005"}]');

        response.setStatusCode(200);

        return response;

    }

}

```