Salesforce Developer - Apex Codes

Apex Triggers Module

Get started with Apex Triggers

Trigger name: AccountAddressTrigger
Code:

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    for(Account a:Trigger.New){
        if(a.Match_Billing_Address__c == true){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

Trigger name: ClosedOpportunityTrigger
Code:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();
    for(Opportunity opp : Trigger.new){
        if(opp.StageName=='Closed Won'){
            taskList.add(new Task(Subject='Follow Up Test Task',WhatId=opp.Id));
        }
    }
    if (taskList.size() > 0) {
        insert taskList;
    }
}
```

Apex Testing Module

Getting Started with Apex Unit Tests

Test Name: TestVerifyDate

Salesforce Developer - Apex Codes

Code:

```
@IsTest
public class TestVerifyDate {

      @isTest static void testDate1Date2() {
      Date testdate = VerifyDate.CheckDates(date.valueof('2022-06-
04'),date.valueOf('2022-06-22'));
      System.assertEquals(date.valueOf('2022-06-22'), testdate);
   }
   @isTest static void test2Date1Date2() {
      Date testdate2 = VerifyDate.CheckDates(date.valueof('2022-06-
04'),date.valueOf('2022-07-22'));
      System.assertEquals(date.valueOf('2022-06-30'), testdate2);
   }
}
```

Test Apex Triggers

Test Name: TestRestrictContactByName
Code:

```
@IsTest
public class TestRestrictContactByName {
   @IsTest static void testingContact(){
      Contact c = new Contact(FirstName='A', LastName='INVALIDNAME');
      Test.startTest();
      Database.SaveResult result = Database.insert(c, false);
      Test.stopTest();
      System.assert(!result.isSuccess());
   }
}
```

Create Test Data for Apex Tests

Class Name: RandomContactFactory

Salesforce Developer - Apex Codes

Code:

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer num, String lastName){
      List<Contact> contacts = new List<Contact>();
      for(Integer i=0;i<num;i++) {
         Contact a = new Contact(FirstName='Test ' + i, LastName = lastName);
         contacts.add(a);
      }
      return contacts;
   }
}
```

Asynchronous Apex Module

Use Future Methods

Class Name: AccountProcessor
Code:

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds) {
      List<Account> accountupdate = new List<Account>();
      List<Account> accounts = [Select Id, Name,( Select Id from Contacts) from Account
Where Id IN :accountIds];
      for(Account acc:accounts){
         List<Contact> contactlist = acc.Contacts;
         acc.Number_Of_Contacts__c = contactlist.size();
         accountupdate.add(acc);
      }
      update accountupdate;
   }
}
```

Test Class Name: AccountProcessorTest

Salesforce Developer - Apex Codes

Code:

```
@IsTest
private class AccountProcessorTest {
   @IsTest
   private static void testcontact() {
      Account newacc = new Account(Name='Test 1');
      insert newacc;
      Contact newcontact = new Contact(FirstName='John', LastName='Doe',AccountId =
newacc.Id);
      insert newcontact;
      Contact newcontact2 = new Contact(FirstName='Jim', LastName='Doe',AccountId =
newacc.Id);
      insert newcontact2;
      List<ID> accountIds = new List<Id>();
      accountIds.add(newAcc.Id);
      Test.startTest();
      AccountProcessor.CountContacts(accountIds);
      Test.stopTest();
   }
}
```

Use Batch Apex

Class Name: LeadProcessor
Code:

```
public class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {
   public Database.QueryLocator start(Database.BatchableContext bc) {
      return Database.getQueryLocator(
         'SELECT ID from Lead'
      );
   }
   public void execute(Database.BatchableContext bc, List<Lead> scope){
      // process each batch of records
      List<Lead> updatelead = new List<Lead>();
```

```
        for (Lead lead : scope) {
            lead.LeadSource = 'Dreamforce';
            updatelead.add(lead);
        }
        update updatelead;
    }
    public void finish(Database.BatchableContext bc){
    }
}
```

Test Class Name: LeadProcessorTest
Code:

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,Company='TestCompany'));
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myBatchObject = new LeadProcessor();
        Id batchId = Database.executeBatch(myBatchObject);
        Test.stopTest();
        System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce']);
    }
}
```

Control Processes with Queueable Apex

Class Name: AddPrimaryContact

Salesforce Developer - Apex Codes

Code:

```
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> primarycontact = new List<Contact>();
      for(Account acc:accounts){
         Contact c = con.clone();
         c.AccountId = acc.Id;
         primarycontact.add(c);
      }
      if(primarycontact.size()>0){
         insert primarycontact;
      }
   }
}
```

Test Class Name: AddPrimaryContactTest
Code:

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

```
        insert testacc;
        Contact testcontact = new Contact(FirstName='Test1',LastName='test2');
        insert testcontact;
        String state='CA';
       AddPrimaryContact updater = new AddPrimaryContact(testcontact, state);
        Test.startTest();
        System.enqueueJob(updater);
        Test.stopTest();
        System.assertEquals(50,[Select count() from Contact where accountId in (Select Id
from Account where BillingState ='CA')]);
    }
}
```

Schedule Jobs Using the Apex Scheduler

Class Name: DailyLeadProcessor
Code:

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx) {
        List<Lead> leadupdate = new List<Lead>();
        List<Lead> leads = [SELECT Id FROM Lead
                            WHERE LeadSource = NULL Limit 200];
        for(Lead l:leads){
            l.LeadSource = 'DreamForce';
            leadupdate.add(l);
        }
        update leadupdate;
    }
}
```

Test Class Name: DailyLeadProcessorTest
Code:

```
@isTest
private class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 6 ? 2023';
```

```
    static testmethod void testScheduledJob() {
        // Create some out of date Opportunity records
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
            Lead l = new Lead(
                FirstName='First',
                LastName = 'Lead ' + i,
                Company = 'test company');
            leads.add(l);
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('ScheduledApexTest',
                                       CRON_EXP,
                                       new DailyLeadProcessor());
        Test.stopTest();
        List<Lead> check = new List<Lead>();
        check = [SELECT Id
                 FROM Lead
                 WHERE LeadSource= 'DreamForce' and Company = 'test company'];
        System.assertEquals(200,check.size(),'Tasks were not created');
    }
}
```

Apex Integration Services Module

Apex REST Callouts

Class Name: AnimalLocator
Code:

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer animalId) {
        String animalname;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
```

```
callout.herokuapp.com/animals/'+animalId);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      if(response.getStatusCode() == 200) {
        Map<String, Object> res = (Map<String, Object>)
          JSON.deserializeUntyped(response.getBody());
        Map<String, Object> animal = (Map<String, Object>)res.get('animal');
        animalname = string.valueof(animal.get('name'));
        }
      return animalname;
   }
}
```

Mock Class Name: AnimalLocatorMock
Code:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');
      response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
      response.setStatusCode(200);
      return response;
   }
}
```

Test Class Name: AnimalLocatorTest
Code:

```
@isTest
private class AnimalLocatorTest {
@isTest static void animalTest() {
   Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
   String response = AnimalLocator.getAnimalNameById(1);
   System.assertEquals('chicken',response);
```

```
}
}
```

Apex SOAP Callouts

Class Name: ParkLocator
Code:

```
public class ParkLocator {
   public static List<String> country(String country){
      ParkService.ParksImplPort parkservice = new ParkService.ParksImplPort();
      return parkservice.byCountry(country);
   }

}
```

Mock Class Name: ParkServiceMock
Code:

```
@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) {
         List<String> parks = new List<String>();
         parks.add('Gir');
         parks.add('Kaziranga');
         ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();
         response_x.return_x = parks;
         response.put('response_x', response_x);
```

```
    }
}
```

Test Class Name: ParkLocatorTest
Code:

```
@isTest
private class ParkLocatorTest {
  @isTest static void testCallout() {
    Test.setMock(WebServiceMock.class, new ParkServiceMock());
    String country = 'India';
    List<String> result = ParkLocator.country(country);
    List<String> parks = new List<String>();
    parks.add('Gir');
    parks.add('Kaziranga');
    System.assertEquals(parks,result);
  }
}
```

Apex Web Services

Class Name: AccountManager
Code:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
  @HttpGet
  global static Account getAccount() {
    RestRequest request = RestContext.request;
    String AccountId = request.requestURI.substringBetween('Accounts/','/contacts');
    Account result =  [SELECT Id,Name,(Select Id,Name from contacts) from Account
where Id= :AccountId];
    return result;

  }
}
```

Salesforce Developer - Apex Codes

Test Class Name: AccountManagerTest

Code:

```
@IsTest
private class AccountManagerTest {
   @isTest static void testGetContactById() {
      Id recordId = createTestRecord();
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
         + recordId+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisacc = AccountManager.getAccount();
      System.assert(thisacc != null);
      System.assertEquals('Test record', thisacc.Name);
   }
   static Id createTestRecord() {
      Account accTest = new Account(
         Name='Test record');
      insert accTest;
      Contact testcontact = new Contact(FirstName='Test',
LastName='1',AccountId=accTest.Id);
      insert testcontact;
      return accTest.Id;
   }
}
```

Apex Specialist Superbadge

Step 2 : Automate Record Creation

Trigger Name: MaintenanceHelper

Salesforce Developer - Apex Codes

Code:

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

Class Name: MaintenanceRequestHelper
Code:

```
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);
            }
         }
      }
      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 (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

Class Name: WarehouseCalloutService
Code:

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setMethod('GET');
    request.setEndpoint(WAREHOUSE_URL);
    HttpResponse response = http.send(request);
    if(response.getStatusCode() == 200) {
      List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
      system.debug('~~ '+jsonResponse);
      List<Product2> productList = new List<Product2>();
      for(Object ob : jsonResponse) {
        Map<String,Object> mapJson = (Map<String,Object>)ob;
        Product2 pr = new Product2();
        pr.Replacement_Part__c = (Boolean)mapJson.get('replacement');
        pr.Name = (String)mapJson.get('name');
        pr.Maintenance_Cycle__c = (Integer)mapJson.get('maintenanceperiod');
        pr.Lifespan_Months__c = (Integer)mapJson.get('lifespan');
        pr.Cost__c = (Decimal) mapJson.get('lifespan');
        pr.Warehouse_SKU__c = (String)mapJson.get('sku');
        pr.Current_Inventory__c = (Double) mapJson.get('quantity');
        productList.add(pr);
      }
      if(productList.size()>0)
        upsert productList;
```

```
    }
  }
  public static void execute(QueueableContext context){
    runWarehouseEquipmentSync();
  }
}
```

To Enqueue jobs run following code in execute anonymous window

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

Step 4: Schedule Synchronization

Class Name: WarehouseSyncSchedule
Code:

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
  global void execute(SchedulableContext ctx){
    System.enqueueJob(new WarehouseCalloutService());
  }
}
```

Step 5: Test Automation Logic

Modifying the Apex class made previously

Class Name: MaintenanceRequestHelper
Code:

```
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);
```

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

Test Class Name: MaintenanceRequestHelperTest
Code:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
  private static Vehicle__c createVehicle(){
    Vehicle__c vehicle = new Vehicle__C(name = 'Test Vehicle');
    return vehicle;
  }
  private static Product2 createEquipment(){
    product2 equipment = new product2(name = 'Test equipment',
                      lifespan_months__c = 10,
                      maintenance_cycle__c = 10,
                      replacement_part__c = true);
    return equipment;
  }
  private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cse = new case(Type='Repair',
              Status='New',
              Origin='Web',
              Subject='Test subject',
              Equipment__c=equipmentId,
              Vehicle__c=vehicleId);
```

```
        return cse;
    }
    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;
    }
    @isTest
    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);
    }
}
```

Step 6: Test Callout Logic
Mock Class Name: WarehouseCalloutServiceMock
Code:

```
@isTest
public class WarehouseCalloutServiceMock implements HTTPCalloutMock {
    public 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":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Test Class Name: WarehouseCalloutServiceTest
Code:

```
@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);
    }
}
```

Step 7: Test Scheduling Logic
Test Class Name: WarehouseSyncScheduleTest
Code:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static void test() {
        String scheduleTime = '0 0 0 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Schedule 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();
    }
}
```

Nihar Dalvi
nihardalvi90@gmail.com