Apex Triggers

Get Started with Apex Triggers

```
AccountAddressTrigger:
trigger AccountAddressTrigger on Account (before insert,before update)
{
    for(Account account:Trigger.new){
        if((account.Match_Billing_Address__c == true) &&
(account.BillingPostalCode!=NULL)){
            account.ShippingPostalCode=Account.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

```
ClosedOpportunityTrigger:
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

Get Started with Apex Unit Tests

VerifyDate:

```
public class VerifyDate {
      public static Date CheckDates(Date d1, Date d2) {
            if(DateWithin30Days(d1,d2)) {
                  return d2;
            } else {
                  return SetEndOfMonthDate(d1);
            }
      }

      private static Boolean DateWithin30Days(Date d1, Date d2) {
      if( d2 < d1) { return false; }
      Date date30Days = d1.addDays(30);
            if( d2 >= 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 {
    @isTest static void test1(){
        Date
d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/202
0'));
```

```
        System.assertEquals(Date.parse('01/03/2020'), d);
    }

    @isTest static void test2(){
        Date
d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/202
0'));
        System.assertEquals(Date.parse('01/31/2020'), d);
    }
}
```

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
public class TestRestrictContactByName {
    @isTest
    public static void testContact(){
        Contact c= new Contact();
        c.LastName='INVALIDNAME';
        Database.SaveResult res=Database.insert(c,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not
allowed for DML',res.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Tests

```
RandomContactFactory:
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numcnt,string lastname){
        List<Contact> contacts=new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt =new
Contact(FirstName='Test'+i,LastName=lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}
```

Asynchronous Apex

Use Future Methods

```
AccountProcessor:
public class AccountProcessor {

  @future
  public static void countContacts(List<Id>  accountIds){
    List<Account> accup=new List<Account>();
    List<Account> accounts=[SELECT Id,(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();
      accup.add(acc);
    }
    update accup;
  }
}
```

AccountProcessorTest:

```
@isTest
public class AccountProcessorTest {

    public static testmethod void testAccountProcessor(){

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

        Contact c =new Contact();
        c.FirstName='Binary';
        c.LastName='Programming';
        c.AccountId=a.Id;
        insert c;

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

        Test.startTest();
        AccountProcessor.countContacts(acclistid);
        Test.stopTest();
        Account acc = [Select Number_Of_Contacts__c from Account where
Id=:a.Id];

System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);

    }
}
```

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 test(){
        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();
    }
}
```

Control Processes with Queueable Apex

```
AddPrimaryContact:
public class AddPrimaryContact implements Queueable{

    private Contact c;
    private String state;
    public AddPrimaryContact(Contact c , String state){
        this.c=c;
        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 con=c.clone();
            con.AccountId=acc.Id;
            primaryContacts.add(con);
        }
        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')]);
    }
}
```

Schedule Jobs Using the Apex Scheduler

```
DailyLeadProcessor:
public class DailyLeadProcessor implements Schedulable{
    public void execute(SchedulableContext ctx){

        List<lead> leads=[select id,LeadSource from Lead where
```

```
LeadSource=NULL Limit 200];

        for(Lead l: leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}

DailyLeadProcessorTest:
@isTest
public class DailyLeadProcessorTest {
     private static String CRON_EXP='0 0 0 ? * * *';
    @isTest
    private static void testSchedulableClass(){
        List<Lead> leads=new List<lead>();
        for(Integer i=0;i<500;i++){
            if(i<250){
                leads.add(new Lead(LastName='Connock',
Company='Salesforce'));

            }
            else{
                leads.add(new Lead(LastName='Connock',
Company='Salesforce',LeadSource='Other'));
            }

        }
        insert leads;

        Test.startTest();
        String jobId=System.Schedule('Process Leads',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> checkleads= new List<Lead>();
        checkleads=[Select Id,LeadSource from Lead Where
```

```
LeadSource='Dreamforce'];
        System.assertEquals(200, checkleads.size(),'Leads were not
created');

        List<CronTrigger> cts=[Select Id, TimesTriggered, NextFireTime
FROM CronTrigger WHERE Id= :jobId];
        System.debug('Next Fire Time'+cts[0].NextFireTime);
    }
}
```

Apex Integration Services

Apex REST Callouts

```
AnimalLocator:
public class AnimalLocator {
    public static String getAnimalNameById(Integer i){
        Http http=new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+i);
        request.setMethod('GET');
        HttpResponse response= http.send(request);

        Map<String, Object> result = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
     Map<String, Object> animal = (Map<String,
Object>)result.get('animal');
        System.debug('name '+string.valueOf(animal.get('name')));
        return string.valueOf(animal.get('name'));

    }
}
```

```
AnimalLocatorTest:
@isTest
private class AnimalLocatorTest {
     @isTest
    static void animalLocatortest1(){
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected ='moose';
        System.assertEquals(actual, expected);
    }
}

AnimalLocatorMock:
@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":"moose","eats":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

```
ParkService:
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
```

```
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new
ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x
= new Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
```

```
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}

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) {
            parkService.byCountryResponse response_x= new
parkService.byCountryResponse();
            response_x.return_x=new
List<String>{'Yosemite','Sequoia','Crater Lake'};
            response.put('response_x',response_x);

        }
    }

ParkLocator:
public class ParkLocator {
    public static List <String > country(String country){
```

```
        ParkService.ParksImplPort prksvc=new
ParkService.ParksImplPort();
        return prksvc.byCountry(Country);
    }
}

ParkLocatorTest:
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkserviceMock());
        String country ='United States';
        List<String> expectedparks=new List<String>{'Yosemite',
'Sequoia', 'Crater Lake'};

System.assertEquals(expectedparks,ParkLocator.country(country));
    }
}
```

Apex Web Services

```
AccountManager:
@RestResource(urlMapping='/Accounts/*/contacts')
global 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,FirstName,LastName
FROM Contacts)
                        FROM Account
                        WHERE Id = :accountId];
        return result;
    }
}
```

```
AccountManagerTest:
@isTest
private class AccountManagerTest {
    @isTest
    static void testGetAccount(){
        Account a = new Account(Name='TestAccount');
        insert a ;
        Contact c = new
Contact(AccountId=a.Id,FirstName='Test',LastName='Test');
        insert c;

        RestRequest request=new RestRequest();

request.requestURI='https://yourInstance.salesforce.com/services/apexr
est/Accounts/'+a.id+'/contacts';
        request.httpMethod='GET';
        RestContext.request=request;

        Account myacc=AccountManager.getAccount();
        System.assert(myacc!=null);
        System.assertEquals('TestAccount', myacc.Name);
    }
}
```

Apex Specialist

STEP 2:

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

```
}

(apex class)
MaintenanceRequestHelper:
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> updateWorders,
Map<Id,Case> nonUpdateMap) {
    Set<Id> validIds = new Set<Id>();

    For (Case c : updateWorders){
        if (nonUpdateMap.get(c.Id).Status != 'Closed' && c.Status ==
'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine
Maintenance'){
                validIds.add(c.Id);
            }
        }
    }

    if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id,
Vehicle__c, ProductId, Product.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'));
    }
```

```
        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
            Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                ProductId =cc.ProductId,
                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.Product.maintenance_Cycle__c);
            //}

            newCases.add(nc);
        }

       insert newCases;

       List<Equipment_Maintenance_Item__c> clonewp = new
List<Equipment_Maintenance_Item__c>();
       for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                clonewp.add(wpClone);

            }
        }
```

```
        insert clonewp;
    }
}
}
```

STEP 3:

WarehouseCalloutService:

```
public with sharing class WarehouseCalloutService implements Queueable
{
    private static final String WAREHOUSE_URL = 'https://th-
superbadge-apex.herokuapp.com/equipment';

    @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());

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();

                product2.Replacement_Part__c = (Boolean)
mapJson.get('replacement');

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

```
                product2.Current_Inventory__c = (Double)
mapJson.get('quantity');

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

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

                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');
    }
}
```

STEP 4:

WarehouseSyncSchedule:

```
global with sharing class WarehouseSyncSchedule implements
```

```
Schedulable{
    // implement scheduled code here
    global void execute (SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

STEP 5:

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;
    }

    @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(equipm
entList.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 idfrom casewhere 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:

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":fal
se,"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:

```
WarehouseSyncScheduleTest:
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    @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();
    }
}
```