# APPEX TRIGGERS

*Get Started with Apex Triggers*

```
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 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

```
@isTest
public class TestVerifyDate {
```

```
    @isTest static void test1(){
        Date d=verifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('01/03/2022'));
        System.assertEquals(Date.parse('01/03/2022'),d);
    }

     @isTest static void test2(){
        Date d=verifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
        System.assertEquals(Date.parse('01/31/2022'),d);
    }

}
```

*Test Apex Triggers*

```
@isTest
public class TestRestrictContactByName {
    @isTest static void Test_insertupdateContact(){
      Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';

      Test.startTest();
      Database.saveResult result = Database.insert(cnt,false);
      Test.stopTest();

      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() >0);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
    }
}
```

*Create Test Data for Apex Tests*

```
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*

```
public without sharing class AccountProcessor {
  @future
  public static void countContacts(List<Id> accountIds){
                List<Account> accounts = [select Id,(select Id from Contacts) from Account
where Id in : accountIds];
    for(Account acc:Accounts){
      acc.Number_Of_Contacts__c=acc.Contacts.size();
    }
    update accounts;

  }
}

@isTest
private class AccountProcessorTest {
        @isTest
  private static void countContactsTest(){
    List<Account> accounts = new List<Account>();
    for(integer i=0;i<300;i++){
      accounts.add(new Account(Name = 'TestAccount' + i));
    }
    insert accounts ;
    List<Contact> contacts = new List<Contact>();
```

```
        List<Id> accountIds = new List<Id>();
        for(account acc :accounts){
            contacts.add(new Contact(FirstName = acc.Name , LastName = 'TestContact' ,AccountId
= acc.Id));
            accountIds.add(acc.Id);
        }
        insert contacts;
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        List<Account> accs =[select Id,Number_Of_Contacts__c from Account];
        for(Account acc:accs){
            System.AssertEquals(1,acc.Number_Of_Contacts__c,'ERROR: Atleast 1 Account Record
with incorrect');
        }
    }
}
```

*Use Batch Apex*

```
public without sharing class LeadProcessor implements  Database.Batchable<sObject>,
Database.Stateful{
    public integer recordcount=0;
    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([select id,name from lead]);
    }
    public void execute(Database.BatchableContext dbc,List<Lead>leads){
        for(Lead l : leads){
            l.LeadSource = 'DreamForce';
        }
        update leads;
        recordcount=recordcount+leads.size();
    }
    public void finish(Database.BatchableContext dbc){
                                System.debug('Total records processed '+recordcount);
    }

}

@isTest
```

```
private class LeadProcessorTest {
        @isTest
   private static void testBatchClass(){
      List<Lead> leads = new List<Lead>();
      for(integer i=0;i<200;i++){
         leads.add(new Lead(LastName = ' Connock ' , Company = ' Salesforce '));
      }
      insert leads;
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp,200);
      Test.stopTest();
      List<Lead> updatedLeads = [select Id from Lead where LeadSource = 'Dreamforce'];
      System.assertEquals(200,updatedLeads.size(),'ERROR: At least 1 lead record not updated
correctly');
   }
}
```

*Control Processes with Queueable Apex*

```
public without sharing class AddPrimaryContact implements Queueable {
   private Contact contact;
   private String state;
   public AddPrimaryContact(Contact inputContact , string inputState){
      this.Contact = inputContact;
      this.state = inputState;
   }
   public void execute(QueueableContext context){
      List<Account> accounts =[select id from Account where billingstate= : state limit 200];
      List<Contact> contacts = new List<Contact>();
      for(Account acc : accounts ){
         Contact contactClone = contact.clone();
         contactClone.AccountId = acc.id;
         contacts.add(contactclone);
      }
      insert contacts;
   }

}
```

```
@isTest
public class AddPrimaryContactTest {
                    @isTest
   private static void testQueueableClass(){
                    List<Account> accounts = new List<Account>();
      for(integer i=0;i<500;i++){
         Account acc = new Account(Name='Test Account');
         if(i<250){
            acc.BillingState = 'NY';
         }
         else{
            acc.BillingState = 'CA';
         }
         accounts.add(acc);
      }
      insert accounts;
      Contact contact = new Contact(FirstName = 'Simon',LastName = 'Connock');
      insert contact;
      Test.startTest();
      Id jobId = System.enqueueJob(new AddPrimaryContact(contact,'CA'));
      Test.stopTest();
      List<contact> contacts = [select Id from contact where Contact.Account.BillingState = 'CA'];
      System.assertEquals(200,contacts.size(),'ERROR : Incorrect number of contact records
found');
   }
}
```

*Schedule Jobs Using the Apex Scheduler*

```
 global class DailyLeadProcessor implements Schedulable{
  global void execute(SchedulableContext sc){
    List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
    List<Lead> lstOfUpdatedLead = new List<Lead>();
    if(!lstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
        ld.LeadSource = 'Dreamforce';
        lstOfUpdatedLead.add(ld);
      }
      UPDATE lstOfUpdatedLead;
```

```
    }
  }
}

@isTest
private class DailyLeadProcessorTest{
  @testSetup
  static void setup(){
    List<Lead> listOfLead = new List<Lead>();
    for(Integer i = 1; i <= 200; i++){
      Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working -
Contacted');
      listOfLead.add(ld);
    }
    Insert listOfLead;
  }
  static testmethod void testDailyLeadProcessorScheduledJob(){
    String sch = '0 5 12 * * ?';
    Test.startTest();
    String jobId = System.schedule('ScheduledApexTest', sch, new DailyLeadProcessor());
    List<Lead> listOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
    System.assertEquals(200, listOfLead.size());
    Test.stopTest();
  }
}
```

# Apex Integration Services

Apex REST Callouts

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

```
        HttpResponse response = http.send(request);
          String strResp = '';
           system.debug('******response '+response.getStatusCode());
           system.debug('******response '+response.getBody());

        if (response.getStatusCode() == 200)
        {
           // Deserializes the JSON string into collections of primitive data types.
          Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
           // Cast the values in the 'animals' key as a list
          Map<string,object> animals = (map<string,object>) results.get('animal');
           System.debug('Received the following animals:' + animals );
           strResp = string.valueof(animals.get('name'));
           System.debug('strResp >>>>>>' + strResp );
        }
        return strResp ;
    }

}

@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);
    }
}

@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;
    }
```

```
}
```

*Apex SOAP Callouts*

```
public class ParkLocator {
   public static String[] country(String country){
      ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
      String[] parksname = parks.byCountry(country);
      return parksname;
   }
}


@isTest
private class ParkLocatorTest{
   @isTest
   static void testParkLocator() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String[] arrayOfParks = ParkLocator.country('India');

      System.assertEquals('Park1', arrayOfParks[0]);
   }
}

@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();
      List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = lstOfDummyParks;
```

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

*Apex Web Services*

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
    }
}

@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();

        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account  acc = AccountManager.getAccount();
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;
```

```
        return acc.Id;
    }
}
```

# SUPERBADGE

## APEX SPECIALIST

### Step 2 : Automated Record Creation

MaintenanceRequestHelper.apxc

```
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()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = 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'));
    }

      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,
          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> clonedWPs = 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;
          ClonedWPs.add(wpClone);
```

```
            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaitenanceRequest.apxt

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

## Step 3 : Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc

```
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(){
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint(WAREHOUSE_URL);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      List<Product2> warehouseEq = new List<Product2>();
      if (response.getStatusCode() == 200){
        List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
         System.debug(response.getBody());
         for (Object eq : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)eq;
```

```
            Product2 myEq = new Product2();
            myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            myEq.Name = (String) mapJson.get('name');
            myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            myEq.Cost__c = (Integer) mapJson.get('cost');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            myEq.ProductCode = (String) mapJson.get('_id');
            warehouseEq.add(myEq);
          }
          if (warehouseEq.size() > 0){
            upsert warehouseEq;
            System.debug('Your equipment was synced with the warehouse one');
          }
        }
    }
    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
        }
 }
```

## Step 4 : Schedule synchronization

WarehouseSyncShedule.apxc

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

## Step 5:Test automation logic

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }
```

```
   PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
      Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                Maintenance_Request__c = requestId);
      return wp;
   }



   @istest
   private static void testMaintenanceRequestPositive(){
      Vehicle__c vehicle = createVehicle();
      insert vehicle;
      id vehicleId = vehicle.Id;

      Product2 equipment = createEq();
      insert equipment;
      id equipmentId = equipment.Id;

      case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
      insert somethingToUpdate;

      Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
      insert workP;

      test.startTest();
      somethingToUpdate.status = CLOSED;
      update somethingToUpdate;
      test.stopTest();

      Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                  from case
                  where status =:STATUS_NEW];

      Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];
```

```
    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

@istest
private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();

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

    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c = :emptyReq.Id];

    system.assert(workPart != null);
    system.assert(allRequest.size() == 1);
}

@istest
```

```
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
          vehicleList.add(createVehicle());
           equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
           requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
           workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
           req.Status = CLOSED;
           oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

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

```
        system.assert(allRequests.size() == 300);
    }
}
```

MaintenanceRequestHelper.apxc :-

```
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()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = 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'));
        }

            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,
                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));
              }

              newCases.add(nc);
            }

          insert newCases;

          List<Equipment_Maintenance_Item__c> clonedWPs = 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;
              ClonedWPs.add(wpClone);
            }
          }
           insert ClonedWPs;
        }
    }
}
```

MaintenanceRequest.apxt :

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

}

## Step 6 : Test callout logic

WarehouseCalloutServiceTest.apxc

```
@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
 Test.startTest();
 Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
 WarehouseCalloutService.runWarehouseEquipmentSync();
 Test.stopTest();
 System.assertEquals(1, [SELECT count() FROM Product2]);
 }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
 global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
 System.assertEquals('GET', request.getMethod());
 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"}]');
 response.setStatusCode(200);
 return response;
 }
}
```

## Step 7:Test scheduling logic

WarehouseSyncScheduleTest.apxc

```
@isTest
public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
 String scheduleTime = '00 00 01 * * ?';
 Test.startTest();
 Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
 String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
 Test.stopTest();
 CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
 System.assertEquals(jobID, a.Id,'Schedule ');
}
}
```