

Apex Triggers :Get Started with Apex Triggers

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

```
for(Account account:Trigger.New){
```

```
if(account.Match_Billing_Address__c==True){
```

```
    account.ShippingPostalCode = account.BillingPostalCode;
```

```
    }
```

```
}
```

Apex Triggers :Bulk Apex Triggers

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

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

```
    for (Opportunity o :[SELECT Id,StageName FROM Opportunity WHERE StageName ='Closed Won'
AND Id IN :Trigger.New])
```

```
    {
```

```
        if(o.StageName == 'Closed Won')
```

```
        {
```

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

```
        }
```

```
    }
```

```
    if(taskList.size() > 0){
```

```
        insert taskList;
```

```
    }
```

```
}
```

Apex Testing :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) {
        //check for date2 being in the past
        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; }
    }

    //method to return the end of the month of a given date
    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

```

@Test
public class VerifyDateTest
{
    static testMethod void testMethod1()
    {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
    }
}

```

Apex Testing :Test Apex Triggers

RestrictContactByName

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

    //check contacts prior to insert or update for invalid data
    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 {
    @isTest static void testName()
    {
        Contact c = new Contact (LastName = 'INVALIDNAME');
        Database.SaveResult result = Database.insert(c, false);
        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());
    }
}
```

Apex Testing :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**

AccountProcessor

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsTopUpdate = 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();
            accountsTopUpdate.add(acc);
        }
        update accountsTopUpdate;
    }
}

```

AccountProcessorTest

```

@Test
private class AccountProcessorTest {
    @Test
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
    }
}

```

```

        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId
= newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId
= newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }

}

```

[Asynchronous Apex](#) :Use Batch Apex

LeadProcessor

global class LeadProcessor implements

Database.Batchable<sObject> {

```

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

```

global void execute(Database.BatchableContext bc, List<Lead> scope){

```

    List<Lead> leads = new List<Lead>();
    for (Lead lead : scope) {

```

```

        lead.LeadSource = 'Dreamforce';
        leads.add(lead);

    }
    update leads;
}
global void finish(Database.BatchableContext bc){

}
}

```

```

LeadProcessorTest
@Test
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();

        // insert 10 accounts
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead'+i,Company='Test Co'));
        }
        insert leads;

    }
    static testmethod void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
    }
}

```

[Asynchronous Apex](#) :Control Processes with Queueable Apex

AddPrimaryContact

```

public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

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

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }
}

```

AddPrimaryContactTest

```

@Test
public class AddPrimaryContactTest {

    @Test
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {

```

```

        account a = new account(name=string.valueOf(50+i),billingstate='CA');
        system.debug('account a = '+a);
        acc_lst.add(a);
    }
    insert acc_lst;
    Test.startTest();
    contact c = new contact(lastname='alex');
    AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
    system.debug('apc = '+apc);
    System.enqueueJob(apc);
    Test.stopTest();
    List<contact> c_lst = new List<contact>([select id from contact]);
    Integer size = c_lst.size();
    system.assertEquals(50, size);
}
}

```

[Asynchronous Apex](#) :Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

global class DailyLeadProcessor implements Schedulable {

```

    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
            }
            update lList;
        }
    }
}

```

DailyLeadProcessorTest

```

@isTest
private class DailyLeadProcessorTest {

```



```

static testMethod void testDailyLeadProcessor() {
    String CRON_EXP = '0 0 1 * * ?';
    List<Lead> IList = new List<Lead>();
    for (Integer i = 0; i < 200; i++) {
        IList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
Inc.', Status='Open - Not Contacted'));
    }
    insert IList;

    Test.startTest();
    String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
}
}

```

Apex Integration Services :**Apex REST Callouts**

AnimalLocator

```

public class AnimalLocator {
    public class cls_animal {
        public Integer id;
        public String name;
        public String eats;
        public String says;
    }
    public class JSONOutput{
        public cls_animal animal;

        //public JSONOutput parse(String json){
        //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
        //}
    }

    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.setHeader('id', String.valueOf(id)); -- cannot be used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
        system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }
}

```

AnimalLocatorTest

@IsTest

```

public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //HttpResponse response = AnimalLocator.getAnimalNameById(1);
        String s = AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}

```

AnimalLocatorMock

@IsTest

```

global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPResponse respond(HTTPPrerequest request) {
        HttpResponse response = new HttpResponse();
        response.setStatusCode(200);
    }
}

```

```

        response.setBody({"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}});
        return response;
    }

}

```

Apex Integration Services :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;
        }
    }
}

```

ParkLocator

```

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

```

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

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

```

ParkLocatorTest

```

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

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

```

Apex Integration Services :**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, Name From Contacts) From
Account Where Id =: accountId];
        system.debug('Result is :'+Result);
        Return Result;
    }
}

```

AccountManagerTest

```

@isTest
private class AccountManagerTest
{
    @isTest static void testGetAccount ()
    {
        Id recordId = createTestRecord ();
        RestRequest request = new RestRequest ();
        request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/' + recordId +
'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert (thisAccount != null);
        System.assertEquals ('Test Record', thisAccount.Name);
    }

    static Id createTestRecord ()
    {
        Account testAccount = new Account (Name = 'Test Record');
        insert testAccount;
    }
}

```

```

        Contact testContact = new Contact (AccountId = testAccount.Id);
        return testAccount.Id;
    }

}

```

APEX SPECIALIST SUPERBADGE

Automate record creation

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);
                }
            }
        }

        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

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

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

    MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

}
}

```

Synchronize Salesforce data with an external system

```

WarehouseCalloutService

public with sharing class WarehouseCalloutService {

    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 = (Decimal) mapJson.get('lifespan');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
        }

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

    }
}
}

```

execute anonymous window

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

Schedule synchronization

WarehouseSyncSchedule

```

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

```

Test automation logic

MaintenanceRequestHelperTest

@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);
}

```



```

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

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

```

MaintenanceRequest

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

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

 MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

 }
}

Test callout logic

WarehouseCalloutServiceTest

@isTest

```

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

```

WarehouseCalloutServiceMock

```

@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        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;
    }
}

```

Test scheduling logic

WarehouseSyncScheduleTest

```

@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();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job
on UNIX systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
    }
}

```



```
System.assertEquals(jobID, a.Id,'Schedule ');
```

```
}
```

```
}
```