

# Apex Triggers

## Getting Started with Apex Triggers

### AccountAddressTrigge.apxt:-

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

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

## Bulk Apex Triggers

### ClosedOpportunityTrigger.apxt:-

```
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

## getting started with apex unit test:-

### VerifyDate.apxc:-

```
@isTest
public class TestVerifyDate {
    private static Date dateToday = date.today();
    private static Integer totalDays = Date.daysInMonth(dateToday.year(), dateToday.month());

    @isTest static void testOldDate(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(-1));
        System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(),
totalDays), dateTest);
    }

    @isTest static void testLessThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(20));
```

```

        System.assertEquals(dateToday.addDays(20), dateTest);
    }

    @isTest static void testMoreThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(31));
        System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(),
totalDays), dateTest);
    }
}

```

### **TestVerifyDate.apxc:-**

```

@isTest
public class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}

```

### **Test Apex Trigger**

### **RestrictContactByNam.apxt:-**

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.apxc:-**

@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 Test**

#### **RandomContactFactory.apxc:-**

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.apxc:-**

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){

        List<Account> accountsToUpdate = 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();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}

```

**AccountProcessorTest.apxc:-**

```

@IsTest
private class AccountProcessorTest {

    @IsTest
    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();
    }
}

```

**use batch apex****LeadProcessor.apxc:-**

```

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.apxc:-**

```

@isTest
public class LeadProcessorTest {

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

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

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

```

## **control processes with queueable apex**

### **AddPrimaryContact.apxc:-**

```

public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

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

```

```

        this.state = state;
    }

    public void execute(QueueableContext Context){
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from
contacts)
                                from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }
    }
}

```

#### **AddPrimaryContactTest.apxc:-**

```

@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.apxc:-**

```

global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = "];

        if(leads.size() > 0){

```

```
List<Lead> newLeads = new List<Lead>();
```

```
for(Lead lead : leads){  
    lead.LeadSource = 'DreamForce';  
    newLeads.add(lead);  
}
```

```
update newLeads;  
}  
}  
}
```

#### **DailyLeadProcessorTest.apxc:-**

```
@isTest  
private class DailyLeadProcessorTest{  
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year  
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';  
  
    static testmethod void testScheduledJob(){  
        List<Lead> leads = new List<Lead>();  
  
        for(Integer i = 0; i < 200; i++){  
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i,  
                Status = 'Open - Not Contacted');  
            leads.add(lead);  
        }  
  
        insert leads;  
  
        Test.startTest();  
        // Schedule the test job  
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new  
            DailyLeadProcessor());  
  
        // Stopping the test will run the job synchronously  
        Test.stopTest();  
    }  
}
```

## **APEX SPECIALIST SUPERBADGE CHALLENGE**

### **1-Automate 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;
}
}
}

```

#### **MaintenanceRequest.apxt :-**

```

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

```

## **CHALLENGE 2 :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';

```

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

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

```

@future(callout=true)
public static void runWarehouseEquipmentSync(){
    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());

        //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU

```

```

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

```

### anonymous window:

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

## CHALLENGE 3: Schedule synchronization using Apex code

### WarehouseSyncSchedule.apxc :-

```

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

```

## CHALLENGE 4: 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);
    }
}
}

```

## CHALLENGE 5: Test callout logic

### WarehouseCalloutService.apxc :-

```

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);
    }

}
}
}

```

#### **WarehouseCalloutServiceTest.apxc :-**

```

@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.apxc :-**

```

@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;
    }
}

```

## CHALLENGE 6 : Test scheduling logic

### WarehouseSyncSchedule.apxc :-

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

### 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();
        //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 ');

    }
}

```