Apex Triggers Module

Get Started with Apex Triggers

AccountAddressTrigger.apxc :-

```
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

ClosedOpportunityTrigger.apxc :-

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

Apex Testing Module

Get Started with Apex Unit Tests

VerifyDate.apxc :-

```
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.apxc :-

```
@isTest
public class TestVerifyDate {
    @isTest static void testCheck1(){
     Date Date1 = Date.parse('01/08/2000');
     Date Date2 = Date.parse('01/10/2000');
     Date D = verifyDate.CheckDates(Date1,Date2);
     system.assertEquals(Date.parse('01/10/2000'),d);
          }
    @isTest static void testCheck2(){
     Date Date1 = Date.parse('01/01/2000');
     Date Date2 = Date.parse('03/03/2000');
     Date D = verifyDate.CheckDates(Date1,Date2);
     system.assertEquals(Date.parse('01/31/2000'),d);
    }
}
```

Test Apex Triggers

RestrictContactByName.apxc :-

```
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 TestCont(){
      Contact c = new Contact();
      c.LastName = 'INVALIDNAME';
      Database.SaveResult result = Database.insert(c,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests

RandomContactFactory.apxc :-

```
 public class RandomContactFactory {
    public static  List<Contact> generateRandomContacts(Integer n , String LName){
       List<Contact>ContactList= new List<Contact>();
        for(Integer i=0;i<n;i++){
            Contact c = new Contact(FirstName = 'Test' +i, LastName=LName);
            ContactList.add(c);
        }
        return ContactList;
    }
}
```

Asynchronous Apex Module

Use Future Method

AccountProcessor.apxc :-

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> AccountIds){
      List<Account> AccountList = [select Id,Number_Of_Contacts__c,(select Id from Contacts)
from Account where Id in : AccountIds];
        for(Account a : AccountList){
            a.Number_Of_Contacts__c = a.Contacts.size();
        }
        update AccountList;
```

```
    }
}
```

AccountProcessorTest.apxc :-

```
@isTest
public class AccountProcessorTest {
   @isTest public static void testAccount(){
      Account ac = new Account();
      ac.Name = 'Test Account';
      insert ac;
      Contact ct = new Contact();
      ct.FirstName = 'Adi';
      ct.LastName = 'R';
      ct.AccountId = ac.id;
      insert ct;

      List<Id> AccountListId = new List<Id>();
      AccountListId.add(ac.Id);

      Test.startTest();
      AccountProcessor.countContacts(AccountListId);
      Test.stopTest();
    }
}
```

Use Batch Apex

LeadProcessor.apxc :-

```
public without sharing class LeadProcessor implements Database.Batchable<sObject>{


   public Database.QueryLocator start(Database.BatchableContext dbc){
       return Database.getQueryLocator('Select ID,LeadSource FROM Lead');
    }
    public void execute(Database.BatchableContext dbc, List<Lead> leads){
       for(Lead Lead:leads){
           lead.LeadSource = 'Dreamforce';
       }
       update leads;
    }
    public void finish(Database.BatchableContext dbc){
    }
}
```

LeadProcessorTest.apxc :-

```
@isTest
private class LeadProcessorTest {


    @isTest
    private static void TestBatch(){
       List<Lead> lds = new List<Lead>();
       for(Integer i=0;i<200;i++){
```

```
            lds.add(new Lead(LastName='Connor', Company='Salesforce'));
        }
        insert lds;
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp,200);
        Test.stopTest();

        List<Lead>updatedLeads = [Select Id from Lead where LeadSource = 'DreamHouse'];
        System.assertEquals(200, updatedLeads.size(), 'Error');
    }
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apxc :-

```
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;
    }
}
```

AddPrimaryContactTest.apxc :-

```
@isTest
public class AddPrimaryContactTest {
@isTest
    private static void testQueueable(){
        List<Account> accounts=new List<Account>();
        for(Integer i=0;i<500;i++){
            Account acct = new Account(Name='Test Account');
            if(i<250){
                acct.BillingState='NY';
            }
            else{
                acct.BillingState='CA';
            }
            accounts.add(acct);
        }
        insert accounts;
        Contact contact = new Contact(FirstName='A', LastName='R');
```

```
        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');
    }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc :-

```
public without sharing 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.apxc :-

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

```
    @isTest
    private static void TestSchedulable(){
        List<Lead> leads= new List<Lead>();
        for(Integer i=0;i<500;i++){
            if(i<250){
                leads.add(new Lead(LastName='R', company='Salesforce'));
            }else{
                leads.add(new Lead(LastName='R', company='Salesforce',LeadSource='Other'));
            }
        }
        insert leads;

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

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

Apex Integration Services Module

Apex REST Callouts

AnimalLocator.apxc :-

```
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.apxc :-

```
@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.apxc :-

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HttpResponse respond(HttpRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('contentType', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

ParkLocator.apxc :-

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

ParkLocatorTest.apxc :-

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        string country = 'India';
        List<String> expectedParks= new List<String>{'Ye','Sa','CL'};
        System.assertEquals(expectedParks,ParkLocator.country(country));
         }
}
```

ParkServiceMock.apxc :-

```
@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>{'Ye','Sa','CL'};
        response.put('response_x',response_x); }
}
```

Apex Web Services

AccountManager.apxc :-

```
@RestResource(urlMapping='/Accounts/*/contacts')


global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result = [Select Id, Name,(Select Id, FirstName, LastName from Contacts)from
            Account where Id=:accountId];
        return result;
           }
}
```

AccountManagerTest.apxc :-

```
@isTest
private class AccountManagerTest {


    @isTest
    static void testgetaccount(){
        Account a = new Account(Name='Test Account');
        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/apexrest//Accounts/'
        +a.id+'/contacts';
    request.httpMethod='Get';
    RestContext.request=request;
    Account myAcc = AccountManager.getAccount();
    System.assert(myAcc!=null);
    System.assertEquals('Test Account', myAcc.Name);
  }
}
```

Apex Specialist Superbadge

Automate record creation

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders(List<case>UpWkOrder,Map<id,Case>nonUpCaMap) {
        set<Id> vId= new Set<Id>();
        for(case ca : UpWkOrder){
            if(nonUpCaMap.get(ca.Id).Status !='Closed' && ca.Status == 'Closed'){
                if(ca.type == 'Repair' || ca.Type == 'Routine Maintenance'){
                    vId.add(ca.Id); }
            }
        }
        if(!vId.isEmpty()){
            Map<Id,case>CldCases = 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 : vId]);
            Map<Id,Decimal>MtnCycles = new Map<Id,Decimal>();
        AggregateResult[] res = [Select
Maintenance_Request__c,Min(Equipment__r.Maintenance_Cycle__c)cycle From
Equipment_Maintenance_Item__c Where Maintenance_Request__c In : VId Group By
Maintenance_Request__c];
            for(AggregateResult ar : res){
                MtnCycles.put((Id) ar.get('Maintenance_Request__c'),(Decimal)ar.get('cycle'));
            }
            List<case> nc = new List<case>();
            for(case c : CldCases.values()){
```

```
            case cs = new Case(ParentId=c.Id,
                              Status='New',
                              Subject='Routine Maintenance',
                              Type='Routine Maintenance',
                              Vehicle__c=c.Vehicle__c,
                              ProductId=c.ProductId,
                              Origin='Web',
                              Date_Reported__c=Date.today()
                              );
              if(MtnCycles.containskey(c.Id)){
                 cs.Date_Due__c=Date.today().addDays((integer)MtnCycles.get(c.Id));
              }
              nc.add(cs);
           }
           insert nc;
           List<Equipment_Maintenance_Item__c>cWp = new
List<Equipment_Maintenance_Item__c>();
           for(case cs : nc){
              for(Equipment_Maintenance_Item__c wp :
CldCases.get(cs.ParentId).Equipment_Maintenance_Items__r){
                 Equipment_Maintenance_Item__c WpCln = wp.clone();
                 WpCln.Maintenance_Request__c = cs.Id;
                 cWp.add(WpCln);
              }
           }
           insert cwp;
        }
```

```
    }
}
```

MaintenanceRequest.apxc :-

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

Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts{


    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';


    public  void execute(QueueableContext context){
        Http hp = new http();
        HttpRequest HReq= new HttpRequest();
        Hreq.setMethod('GET');
        Hreq.setEndpoint(WAREHOUSE_URL);
        HttpResponse HRes = hp.send(HReq);
        if(HRes.getStatusCode()==200){
            List<Object> ResBody = (List<Object>)JSON.deserializeUntyped(HRes.getBody());
            List<Product2> EquipList = new List<Product2>();
```

```
        for(Object Ob : ResBody){
            Map<String, Object>ResMap = (Map<String, Object>)ob;
            Product2 NewEquip = new Product2();
            NewEquip.Name=(String)ResMap.get('name');
            NewEquip.Replacement_Part__c=true;
            NewEquip.Cost__c=(Decimal)ResMap.get('cost');
            NewEquip.Current_Inventory__c=(Decimal)ResMap.get('quantity');
            NewEquip.Lifespan_Months__c=(Decimal)ResMap.get('lifespan');
            NewEquip.Maintenance_Cycle__c=(Decimal)ResMap.get('maintenanceperiod');
            NewEquip.Warehouse_SKU__c=(String)ResMap.get('sku');
            EquipList.add(NewEquip);
        }
        if(EquipList.size()>0){
            upsert EquipList;}
      }
   }
}
```

Schedule synchronization using Apex code

WarehouseSyncSchedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
   // implement scheduled code here
   global void execute(SchedulableContext cont){
      System.enqueueJob(new WarehouseCalloutService());
   }
}
```

Test automation logic

MaintenanceRequestHelperTest.apxc :-

```
@isTest
public class MaintenanceRequestHelperTest{
    @testSetup
    static void testData(){
        Vehicle__c v= new Vehicle__c();
        v.Name = 'Rv';
        v.Awning__c='Manual';
        v.Bedrooms__c= 1.0;
        insert v;
        List<Product2> eqpList = new List<Product2>();
        Product2 equip = new Product2(Name='Test',
                                      Replacement_Part__c=true,
                                      Cost__c=110,
                                      Current_inventory__c=12,
                                      Lifespan_Months__c=12,
                                      Maintenance_Cycle__c=12,
                                      Warehouse_SKU__c='xyz');
        eqpList.add(equip);
        insert eqpList;
        list<case> tcList = new List<Case>();
        for(integer i=0;i<300;i++){
            case tc = new case();
            tc.Subject = 'Test Case'+i;
            tc.Vehicle__c = v.id;
```

```
        tc.Status = 'New';
        tc.Priority = 'Medium';
        tc.Origin = 'Phone';
        tc.Type = 'Repair';
        tcList.add(tc);
    }
    insert tcList;
    list<Equipment_Maintenance_Item__c> itemList = new
    List<Equipment_Maintenance_Item__c>();
    for(case ca : tcList){
        Equipment_Maintenance_Item__c item = new Equipment_Maintenance_Item__c();
        item.Maintenance_Request__c = ca.Id;
        item.Equipment__c = eqpList[0].Id;
        item.Quantity__c = 2;
        itemList.add(item);
    }
    insert itemList;
}
@isTest
public static void PositiveTesting1(){
    case tcs = [Select id,status from case limit 1];
    tcs.Status='Closed';
    Test.startTest();
    update tcs;
    Test.stopTest();
    Case nwCase= [Select id, Type From Case where id=:tcs.id];
    System.assertEquals(nwCase.Type,'Repair');
```

```
}
@isTest
public static void PositiveTesting(){
    Vehicle__c v= new Vehicle__c(Name = 'Rv',
                                 Awning__c='Manual',
                                Bedrooms__c= 1.0);
    insert v;
    Product2 equip = new Product2(Name='Test',
                                  Replacement_Part__c=true,
                                  Cost__c=110,
                                  Current_inventory__c=12,
                                  Lifespan_Months__c=12,
                                  Maintenance_Cycle__c=12,
                                  Warehouse_SKU__c='xyz');
    insert equip;
    case tc = new case();
    tc.Subject = 'Test Case';
    tc.Vehicle__c = v.id;
    tc.Status = 'New';
    tc.Priority = 'Medium';
    tc.Origin = 'Phone';
    tc.Type = 'Repair';
    insert tc;
    Equipment_Maintenance_Item__c item = new Equipment_Maintenance_Item__c();
    item.Maintenance_Request__c = tc.Id;
    item.Equipment__c = equip.Id;
    item.Quantity__c = 2;
```

```
    insert item;
    tc.Status = 'Closed';
    Test.startTest();
    update tc;
    Test.stopTest();
}
@isTest
public static void negativeTesting(){
    Vehicle__c v= new Vehicle__c(Name = 'Rv',
                                 Awning__c='Manual',
                                 Bedrooms__c= 1.0);
    insert v;
    Product2 equip = new Product2(Name='Test',
                                  Replacement_Part__c=true,
                                  Cost__c=110,
                                  Current_inventory__c=12,
                                  Lifespan_Months__c=12,
                                  Maintenance_Cycle__c=12,
                                  Warehouse_SKU__c='xyz');
    insert equip;
    case tc = new case();
    tc.Subject = 'Test Case';
    tc.Vehicle__c = v.id;
    tc.Status = 'New';
    tc.Priority = 'Medium';
    tc.Origin = 'Phone';
    tc.Type = 'Repair';
```

```
        insert tc;
        Equipment_Maintenance_Item__c item = new Equipment_Maintenance_Item__c();
        item.Maintenance_Request__c = tc.Id;
        item.Equipment__c = equip.Id;
        item.Quantity__c = 2;
        insert item;
        tc.Status = 'Pending';
        Test.startTest();
        update tc;
        Test.stopTest();
    }
    @isTest
    public static void bulkTesting(){
        List<case> blkUpdList = new List<Case>();
        List<case> blkCaList = [Select Id , Status from case];
        for(case ca : blkCaList){
            ca.Status = 'Closed';
            blkUpdList.add(ca);
        }
        Test.startTest();
        update blkUpdList;
        Test.stopTest();
    } }
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders(List<case>UpWkOrder,Map<id,Case>nonUpCaMap) {
        set<Id> vId= new Set<Id>();
```

```
    for(case ca : UpWkOrder){
        if(nonUpCaMap.get(ca.Id).Status !='Closed' && ca.Status == 'Closed'){
            if(ca.type == 'Repair' || ca.Type == 'Routine Maintenance'){
                vId.add(ca.Id);
            }
        }
    }
    if(!vId.isEmpty()){
        Map<Id,case>CldCases = 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 : vId]);
        Map<Id,Decimal>MtnCycles = new Map<Id,Decimal>();
    AggregateResult[] res = [Select
Maintenance_Request__c,Min(Equipment__r.Maintenance_Cycle__c)cycle From
Equipment_Maintenance_Item__c Where Maintenance_Request__c In : VId Group By
Maintenance_Request__c];
        for(AggregateResult ar : res){
            MtnCycles.put((Id) ar.get('Maintenance_Request__c'),(Decimal)ar.get('cycle'));
        }
        List<case> nc = new List<case>();
        for(case c : CldCases.values()){
          case cs = new Case(ParentId=c.Id,
                             Status='New',
                             Subject='Routine Maintenance',
                             Type='Routine Maintenance',
                             Vehicle__c=c.Vehicle__c,
                             ProductId=c.ProductId,
                             Origin='Web',
```

```
                    Date_Reported__c=Date.today()
                    );
            if(MtnCycles.containskey(c.Id)){
              cs.Date_Due__c=Date.today().addDays((integer)MtnCycles.get(c.Id));

            }
            nc.add(cs);
        }
        insert nc;
        List<Equipment_Maintenance_Item__c>cWp = new
List<Equipment_Maintenance_Item__c>();
        for(case cs : nc){
            for(Equipment_Maintenance_Item__c wp :
CldCases.get(cs.ParentId).Equipment_Maintenance_Items__r){
              Equipment_Maintenance_Item__c WpCln = wp.clone();
              WpCln.Maintenance_Request__c = cs.Id;
              cWp.add(WpCln);
            }
        }
        insert cwp;
      }
   }
}
```

MaintenanceRequest.apxc :-

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

Test callout logic

WarehouseCalloutServiceTest.apxc :-

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @isTest
    static void test1(){
        Test.startTest();
        Test.setMock(HTTPCalloutMock.class,new WarehouseCalloutServiceMock());
        system.enqueueJob(new WarehouseCalloutService());
        Test.stopTest();
        Product2 eqp = [Select Id, Warehouse_SKU__c, Maintenance_Cycle__c from Product2 limit
1];
        System.assertEquals('122221', eqp.Warehouse_SKU__c);
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock{
    // implement http mock callout
    global HttpResponse respond(HttpRequest req){
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type','application/json');

res.setBody('[{\"_id\":\"55d66226726b611100aaf741\",\"replacement\":false,\"quantity\":2,\"
name\":\"TestClass 1000
kW\",\"maintenanceperiod\":225,\"lifespan\":100,\"cost\":5000,\"sku\":\"122221\"}]');
        res.setStatusCode(200);
        return res;
    }
}
```

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts{

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    public  void execute(QueueableContext context){
        Http hp = new http();
        HttpRequest HReq= new HttpRequest();
        Hreq.setMethod('GET');
        Hreq.setEndpoint(WAREHOUSE_URL);
```

```
        HttpResponse HRes = hp.send(HReq);
        if(HRes.getStatusCode()==200){
            List<Object> ResBody = (List<Object>)JSON.deserializeUntyped(HRes.getBody());
            List<Product2> EquipList = new List<Product2>();
            for(Object Ob : ResBody){
                Map<String, Object>ResMap = (Map<String, Object>)ob;
                Product2 NewEquip = new Product2();
                NewEquip.Name=(String)ResMap.get('name');
                NewEquip.Replacement_Part__c=true;
                NewEquip.Cost__c=(Decimal)ResMap.get('cost');
                NewEquip.Current_Inventory__c=(Decimal)ResMap.get('quantity');
                NewEquip.Lifespan_Months__c=(Decimal)ResMap.get('lifespan');
                NewEquip.Maintenance_Cycle__c=(Decimal)ResMap.get('maintenanceperiod');
                NewEquip.Warehouse_SKU__c=(String)ResMap.get('sku');
                EquipList.add(NewEquip);
            }
            if(EquipList.size()>0){
                upsert EquipList;
            }
        }
    }
}
```

Test scheduling logic

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
      public static string CRON_EXP = '0 0 1 * * ?';
    @isTest
    static void tSchedulemethod(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        Test.startTest();
        string jID = system.schedule('WarehouseSyncScheduleTest', CRON_EXP, new
WarehouseSyncSchedule());
        Test.stopTest();
        List<Product2> eqpList = [Select Id, Warehouse_SKU__c from Product2];
        System.assertEquals(0,eqpList.size());


    }
}
```

WarehouseSyncSchedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    // implement scheduled code here
    global void execute(SchedulableContext cont){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```