

1.Apex Trigger:

A. Create an Apex trigger:

a. Account Address Trigger:

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

B. Create a Bulk Apex trigger :

a. Closed Opportunity Trigger:

```
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;  
    }  
}
```

2.Apex Testing :

A. Create an Apex Class & Place the unit tests in a separate test class:

a. Verify Date:

```
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  
    @TestVisible 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  
    @TestVisible 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;  
    }  
}
```

b. Test Verify Date:

```
@isTest  
private 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(false, flag);
    }

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

```

B. Create an Apex trigger on the Contact object & Place the unit tests in a separate test class:

a. Restrict Contact By Name:

```

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');
        }
    }
}

```

b. Test Restrict Contact By Name:

```

@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()); }
}

```

c. Create a Contact Test Factory :

a. Random Contact Factory :

```

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;
    }
}

```

3.Asynchronous Apex:

A. Create an Apex class that uses the @future annotation to update Account records:

a. Account Processor:

```

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;
    }
}

```

b. Account Processor Test:

```

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

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

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

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

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

```

B. Create an Apex class that uses Batch Apex to update Lead records:

a. Lead Processor:

```

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);
    }
}

```

b. Lead Processor Test:

```

@Test
public class LeadProcessorTest {
    @Test
    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();
    }
}

```

c. Create a Queueable Apex class that inserts Contacts for Accounts:

a. Add Primary Contact:

```

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);
        }
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }
}

```

b. Add Primary Contact Test:

```

@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 = 'Steven', LastName = 'Strange');

        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')]);
    }
}

```

D. Create an Apex class that uses Scheduled Apex to update Lead

records:

a. Daily Lead Processor:

```
public class DailyLeadProcessor implements Schedulable{
    Public void execute(SchedulableContext ctx){
        List<Lead> LeadObj=[SELECT Id from Lead where LeadSource = null limit 200];
        for(Lead l:LeadObj){
            l.LeadSource='Dreamforce';
            update l;
        }
    }
}
```

b. Daily Lead Processor Test:

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

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

4.Lighting Web Components (LWC):

a. Create an app page for the bike card component:

a. Selector.html :

```
<template>
  <divclass="wrapper">
    <headerclass="header">Select a Bike</header>
    <sectionclass="content">
      <divclass="columns">
        <mainclass="main">
```



```

        <b>{name}</b>
      </c-list>
    onproductselected={handleProductSelected}></c-list>
  </main>
  <asideclass="sidebar-second">
    <c-detailproduct-id={selectedProductId}></c-detail>
  </aside>
</div>
</section>
</div>
</template>

```

b. Selector.js :

```

import { LightningElement } from 'lwc';

export default class Selector extends LightningElement {
  selectedProductId;

  handleProductSelected(evt) {
    this.selectedProductId = evt.detail;
  }
}

```

c. Selector.js-meta.xml :

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundlexmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>48.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>

```

```
</targets>  
</LightningComponentBundle>
```

5. Apex Integration Services :

a. Apex REST Callouts :

a. Animal Locator :

```
public class AnimalLocator {  
    public static String getAnimalNameById(Integer animalId) {  
        String animalName;  
        Http http = new Http();  
        HttpRequest request = new HttpRequest();  
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+animalId);  
        request.setMethod('GET');  
        HttpResponse response = http.send(request);  
        // If the request is successful, parse the JSON response.  
        if(response.getStatusCode() == 200) {  
            Map<String, Object> r=(Map<String, Object>)  
                JSON.deserializeUntyped(response.getBody());  
            Map<String, Object> animal = (Map<String, Object>)r.get('animal');  
            animalName = string.valueOf(animal.get('name'));  
        }  
        return animalName;  
    }  
}
```

b. Animal Locator Test :

```
@isTest  
private class AnimalLocatorTest{  
    @isTest static void getAnimalNameByIdTest () {  
        // Set mock callout class  
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());  
        // This causes a fake response to be sent  
        // from the class that implements HttpCalloutMock.  
        String response = AnimalLocator.getAnimalNameById(1);  
        // Verify that the response received contains fake values  
        System.assertEquals('chicken', response);  
    }  
}
```

c. Animal Locator Mock :

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        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;
    }
}
```

b. Apex SOAP Callouts :

a. Park Locator :

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

b. Park Locator Test :

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        String country = 'United States';
        List<String> result = ParkLocator.Country(country);
        List<String> parks = new List<String>();
        parks.add('Yellow Stone');
        parks.add('Yellowstone');
        parks.add('Another Park');
        // Verify that a fake result is returned
        System.assertEquals(parks, result);
    }
}
```

```
    }
}
```

c. Apex Web Services :

a. Account Manager :

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {

    @HttpGet
    global static account getAccount() {

        RestRequest request = RestContext.request;

        String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
            request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account where id =
:accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= ' + a[0]);
        return a[0];

    }

}
```

b. Account Manager Test :

```
@istest
public class AccountManagerTest {
    @istest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        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);
    }

    // Helper method
    static Id createTestRecord() {

        // Create test record
```

```

Account accountTest = new Account(
    Name='Test record');
insert accountTest;
Contact contactTest = new Contact(
    FirstName='John',
    LastName='Doe',
    AccountId=accountTest.Id
);
return accountTest.Id;
}
}

```

6. Apex Specialist Superbadge :

a. Maintenance Request Helper :

```

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;
}
}
}

```

b. Maintenance Request Helper Test :

```

@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);
    }
}

```

c. Maintenance Request :

```

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

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

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

    }

}

```

d. Warehouse Callout Service :

```

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);
            }
        }
    }
}

```

e. Warehouse Callout Service Test :

```

@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]);
    }
}

```

f. Warehouse Callout Service Mock :

```
@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;
    }
}
```

g. Warehouse Sync Schedule :

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

h. Warehouse Sync Schedule Test

```
@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 ');
    }
}
```

