SPSGP-32620-Salesforce Developer Catalyst
Self-Learning & Super Badges

APEX SPECIALIST SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.axpt:

```
trigger AccountAddressTriggeron Account (before insert,before update) {
   for(Account account:Trigger.New){
         if(account.Match_Billing_Addressc == True){
         account.ShippingPostalCode = account.BillingPostalCode;
    }
 }
}
```

ClosedOpportunityTrigger.axpt:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
   for(Opportunity opp : Trigger.new) {
//Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on
//Create
       if(Trigger.isInsert) {
           if(Opp.StageName == 'Closed Won') {
                   taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
}
}
//Only create Follow Up Task only once when Opp StageName changed to 'Closed Won'
//on Update
if(Trigger.isUpdate) {
if(Opp.StageName == 'Closed Won' && Opp.StageName !=
Trigger.oldMap.get(opp.Id).StageName) {
     taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
```

```
}
}
}
if(taskList.size()>0) {
insert taskList;
}
}
```

APEX TESTING

VerifyData.apxc:

```
public static Date CheckDates(Date date1, Date date2) {
      if(DateWithin30Days(date1,date2)) {
          return date2;
} else {
}
}
return SetEndOfMonthDate(date1);
     @TestVisible privatestatic Boolean DateWithin30Days(Datedate1, Date date2) {
//check for date2being 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 returntheend of the monthof a given date
@TestVisible
  private staticDate SetEndOfMonthDate(Date date1){
  IntegertotalDays =Date.daysInMonth(date1.year(), date1.month());
  Date lastDay = Date.newInstance(date1.year(), date1.month(),totalDays);
   return lastDay;
}
```

```
}
```

TestVerifyData.apxc:

```
@isTest
   private class TestVerifyDate {
@isTest
static void Test_CheckDates_case1(){
Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('01/05/2022'));
          System.assertEquals(date.parse('01/05/2022'), D);
}
@isTest
    static void Test_CheckDates_case2(){
       Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),
date.parse('05/05/2022'));
     System.assertEquals(date.parse('01/31/2022'), D);
}
@isTest
static void Test_Within30Days_case1(){
    Boolean flag =
        VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
    date.parse('12/30/2021'));
    System.assertEquals(false, flag);
}
@isTest
 static void Test_Within30Days_case2(){
          Boolean flag =VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
          date.parse('02/02/2021'));
           System.assertEquals(false, flag);
}
```

RestrictContactByName.apxt:

```
trigger RestrictContactByName on Contact(beforeinsert, 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 allowedforDML');
}
}
}
```

TestRestrictContactByName.apxc:

```
@isTest
private class TestRestrictContactByName
{
@isTeststatic
   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());
}
}
```

RandomContactFactory.apxc:

```
public class RandomContactFactory {
  public static List<Contact> generateRandomContacts(Integer num_cnts, string
lastname) {
        List<Contact> contacts = new List<Contact>();
        for(Integer i = 0; i < num_cnts; i++) {
           Contact cnt = new Contact(FirstName = 'Test' +i,LastName = lastname);
           contacts.add(cnt);
}
        return contacts;
}
}
```

SPSGP-32620-Salesforce Developer Catalyst
Self-Learning & Super Badges

APEXSPECIALIST SUPER BADGE CODES

ASYNCHRONOUSAPEX

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
       public class AccountProcessorTest {
@isTest
     private static void testCountContacts() {
        Account newAccount = new Account(Name = 'TestAccount'); insert newAccount;
      Contact newContact1 = new Contact(FirstName = 'John',LastName =
'Doe',AccountId=newAccount.id);
    Contact newContact2 = new Contact(FirstName = 'John',LastName = 'Doe',AccountId
=newAccount.Id);
  insert newContact2;
    List<Id> accountIds = new List<Id>();
    accountIds.add(newAccount.Id);
   Test.startTest();
  AccountProcessor.countContacts(accountIds); Test.stopTest();
}
}
```

SPSGP-32620-Salesforce Developer Catalyst
Self-Learning & Super Badges

LeadProcessor.apxc:

```
global class LeadProcessor implements
        Database.Batchable<sObject>{
global Integercount = 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 {
@testSetup
static void setup() {
    List<Lead> leads = new List<Lead>();
    for(Integer counter=0 ;counter <200;counter++){
    Lead lead = new Lead();
    lead.FirstName ='FirstName';
   lead.LastName ='LastName'+counter;
   lead.Company='demo'+counter;
     leads.add(lead);
}
   insert leads;
}
```

```
@isTest static void test() {
  Test.startTest();
  LeadProcessor leadProcessor = new LeadProcessor();
  Id batchId = Database.executeBatch(leadProcessor);
  Test.stopTest();
}
}
```

AddPrimaryContact.apxc:

```
public class AddPrimaryContact implements Queueable{
      private Contactcon;
      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 Accountwhere 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;
}
}
```

AddPrimaryContactTest.apxc:

```
@isTest public class AddPrimaryContactTest{
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')]);
}
}
```

DailyLeadProcessor.apxc:

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

DailyLeadProcessorTest.apxc:

```
@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());
}
}
```

APEXSPECIALIST SUPER BADGE CODES

AccountManagerTest.apxc:

```
@isTest
private class AccountManagerTest {
   private static testMethod void
           getAccountTest1() {
      Id recordId=createTestRecord();
//Set up a test request
    RestRequest request=new RestRequest();
       request.requestUri = 'https: /na1.salesforce.com/services/apexrest/Accounts/'+
recordId+'/contacts' ;
   request.httpMethod = 'GET';
    RestContext.request = request;
/ /Call the methodtotest
Account thisAccount = AccountManager.getAccount();
/ /Verify results
    System.assert(thisAccount !=null);
    System.assertEquals('Test record', thisAccount.Name);
}
/ Helper method
static Id createTestRecord() {
/Create test record
       Account TestAcc = new Account( Name='Testrecord');
       insert TestAcc;
       Contact TestCon= new Contact(LastName='Test',AccountId=TestAcc.id);
       return TestAcc.Id;
}
}
```

SPSGP-32620-Salesforce Developer Catalyst
Self-Learning & Super Badges

APEXSPECIALIST SUPER BADGE CODES

APEX SPECIALIST SUPERBADGE Challenge

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,(SELECTId,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);
}
}
```

SPSGP-32620-Salesforce Developer Catalyst
Self-Learning & Super Badges

Challenge-2

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();
}
}
```

Challenge-3

WarehouseSyncSchedule.apxc:

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

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 Scheduleto Test', scheduleTime,
new WarehouseSyncSchedule());
      Test.stopTest();
//Contains schedule information for a scheduledjob. CronTrigger is similartoa 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 ');
}
}
```

Challenge-4

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 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;
}
}
}
```

MaintenanceRequestHelper.apxc:

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

Challenge-5

WarehouseCalloutService.apxc:

```
public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-superbadgeSPSGP-
apex.herokuapp.com/equipment';
//@future(callout=true)
   public static void runWarehouseEquipmentSync(){
       Http http = new Http();
       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);
}
}
}
}
```

WarehouseCalloutServiceMock.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]);
}
}
```

WarehouseCalloutServiceTest.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

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, newWarehouseSyncSchedule());
        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 ');
}
}
```