Account Address Trigger:

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

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

ClosedOpportunityTrigger:

```
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;
    }
}
```

VerifyDate:

```
public class VerifyDate {

        //method to handle potential checks against two dates
        public static Date CheckDates(Date date1, Date date2) {
                //if date2 is within the next 30 days of date1, use date2.  Otherwise use
the end of the month
```

```
            if(DateWithin30Days(date1,date2)) {
                return date2;
            } else {
                return SetEndOfMonthDate(date1);
            }
        }

        //method to check if date2 is within the next 30 days of date1
        @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;
        }

}
```

TestVerifyDate:

```
@isTest
public class TestVerifyDate {

   static testMethod void testCheckDate(){

     date d1 = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));
      date d3 = VerifyDate.CheckDates(date.today(), date.today().addDays(28));
        }
```

```
    static testMethod void testCheckGreater(){
        date d2 = VerifyDate.CheckDates(date.today(), date.today().addDays(30));
    }
}
```

RestrictContactByName:

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

        //check contacts prior to insert or update for invalid data
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {       //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                }

        }



}
```

TestRestrictContactByName:

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        //try inserting a Contact with INVALIDNAME
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;

        // Perform test
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        // Verify
```

```
        // In this case the creation should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                             result.getErrors()[0].getMessage());

    }
}
```

RandomContactFactory:

```
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;

    }
}
```

AccountProcessor:

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
            account.Number_of_Contacts__c = [Select count() from Contact Where AccountId
=: account.Id];
```

```
            System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
        update updatedAccounts;
    }

}
```

AccountProcessorTest:

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

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

}
```

LeadProcessor:

```
global class LeadProcessor implements    Database.Batchable<Sobject>
{
   global Database.QueryLocator start(Database.BatchableContext bc)
   {
      return Database.getQueryLocator([Select LeadSource From Lead ]);
   }

   global void execute(Database.BatchableContext bc, List<Lead> scope)
   {
         for (Lead Leads : scope)
         {
            Leads.LeadSource = 'Dreamforce';
         }
      update scope;
   }

   global void finish(Database.BatchableContext bc){   }
}
```

LeadProcessorTest:

```
@isTest
public class LeadProcessorTest
{
   static testMethod void testMethod1()
   {
      List<Lead> lstLead = new List<Lead>();
      for(Integer i=0 ;i <200;i++)
      {
         Lead led = new Lead();
         led.FirstName ='FirstName';
         led.LastName ='LastName'+i;
         led.Company ='demo'+i;
         lstLead.add(led);
```

```
        }

        insert lstLead;

        Test.startTest();

            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);

        Test.stopTest();
    }
}
```

AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
        List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName
from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
        List<Contact> lstContact = new List<Contact>();
        for (Account acc:ListAccount)
        {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId =  acc.id
;
            lstContact.add( cont );
        }
```

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

    }

}
```

AddPrimaryContactTest:

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
           Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
           Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;

        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';

         AddPrimaryContact apc = new AddPrimaryContact(co, state);
         Test.startTest();
```

```
        System.enqueueJob(apc);
       Test.stopTest();
    }
}
```

DailyLeadProcessor:

```
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:

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

AnimalLocator:

```
public class AnimalLocator {
   public static String getAnimalNameById(Integer id) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
         /*Map<String,Object> results =
(Map<String,Object>)JSON.deserializeUntyped(response.getBody());
      system.debug('---->results'+results);
      List<Object> animals = (List<Object>) results.get('animal');
      system.debug('---->animal'+animals);*/
      Map<Integer,String> mapAnimal = new Map<Integer,String>();
      Integer varId;
      String varName;
      JSONParser parser1= JSON.createParser(response.getBody());
      while (parser1.nextToken() != null) {
         if ((parser1.getCurrentToken() == JSONToken.FIELD_NAME) &&
(parser1.getText() == 'id')) {
            // Get the value.
            parser1.nextToken();
            // Fetch the ids for all animals in JSON Response.
            varId=parser1.getIntegerValue();
            System.debug('---->varId-->'+varID);
            parser1.nextToken();
         }
         if ((parser1.getCurrentToken() == JSONToken.FIELD_NAME) &&
(parser1.getText() == 'name')) {
            parser1.nextToken();
            // Fetch the names for all animals in JSON Response.
            varName=parser1.getText();
            System.debug('---->varName-->'+varName);
         }
         mapAnimal.put(varId,varName);
```

```
        }
        system.debug('---->mapAnimal-->'+mapAnimal);
        return mapAnimal.get(id);

    }
}
```

AnimalLocatorTest:

```
@isTest
private class AnimalLocatorTest {
@isTest static void testGetCallout() {
    // 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);
    system.debug('Test Response1--->'+response);
    String expectedValue = 'chicken';
    System.assertEquals(expectedValue,response);
    String response2 = AnimalLocator.getAnimalNameById(2);
    system.debug('Test Response2--->'+response2);
    String expectedValue2 = 'duck';
    System.assertEquals(expectedValue2,response2);
}
}
```

AnimalLocatorMock :

```
@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"},{"id":2,"name":"duck","eats":"worms","says":"pek pek"}]}');
        response.setStatusCode(200);
        return response;
    }
}
```

ParkLocator:

```
public class ParkLocator {
    public static string[] country(string theCountry){
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort();
        return parkSvc.byCountry(theCountry);
    }
}
```

ParkLocatorTest:

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

ParkServiceMock :

```
@isTest
```

```
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
         Object stub,
         Object request,
         Map<String, Object> response,
         String endpoint,
         String soapAction,
         String requestName,
         String responseNS,
         String responseName,
         String responseType) {
       ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
       List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
       response_x.return_x = lstOfDummyParks;
       response.put('response_x', response_x);
   }
}
```

AccountManager:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
   @HttpGet
   global static Account getAccount() {
      RestRequest req = RestContext.request;
      String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                  FROM Account WHERE Id = :accId];
                   return acc;
   }
}
```

AccountManagerTest:

```
@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 method to test
        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='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

MaintenanceRequest:

```
trigger MaintenanceRequest on Case (before update, after update) {
    // call MaintenanceRequestHelper.updateWorkOrders
```

```
    if ( Trigger.isAfter && Trigger.isUpdate )  {
MaintenanceRequestHelper.updateWorkOrders(Trigger.newMap); }
}
```

MaintenanceRequestHelper:

```
public class MaintenanceRequestHelper {

        public static final Set<String> validRequestTypes = new Set<String>{'Repair',
'Routine Maintenance'};
    public static final String closedStatus = 'closed';
    public static final String newCaseStatus = 'New';
        public static final String newCaseType = 'Routine Maintenance';
        public static final String newCaseSubject = 'Routine Check';

    public static void updateWorkOrders(Map<Id, SObject> newSoMap){
        Map<Id, Case> caseMap = (Map<Id, Case>) newSoMap;
        Map<Integer, Case> insertCaseMap = new Map<Integer, Case>();
        Map<Integer, List<Work_Part__c>> insertWorkPartMap = new Map<Integer,
List<Work_Part__c>>();
        List<Work_Part__c> insertWorkPartList = new List<Work_Part__c>();
        Map<Id, List<Work_Part__c>> reqWorkPartMap = new Map<Id,
List<Work_Part__c>>();
        Set<Id> casesToProcess = new Set<Id>();
        Set<Id> equipmentsToProcess = new Set<Id>();
        Map<Id, Id> idWorkPartEquipMap = new Map<Id, Id>();
        Map<Id, Product2> equipMap;
        Integer key = 0;
```

```
        for (Case c: caseMap.values()) {
            if ( !c.Status.toLowerCase().equals( MaintenanceRequestHelper.closedStatus ) )
{ continue; }
            else if ( !validRequestTypes.contains( c.Type ) ) { continue; }
            else { casesToProcess.add(c.Id); }
        }
        System.debug('Cases to Process: ' + casesToProcess);

        if ( casesToProcess.size() == 0 ) { return; }

        for (Work_Part__c wp: [Select Id, Name, Equipment__c, Maintenance_Request__c,
Quantity__c From Work_Part__c Where Maintenance_Request__c IN :casesToProcess]) {
            equipmentsToProcess.add(wp.Equipment__c);
            idWorkPartEquipMap.put(wp.Id, wp.Equipment__c);
            List<Work_Part__c> tmpWpList =
reqWorkPartMap.get(wp.Maintenance_Request__c);
            if ( tmpWpList == null ) { tmpWpList = new List<Work_Part__c>(); }
            tmpWpList.add(wp);
            reqWorkPartMap.put(wp.Maintenance_Request__c, tmpWpList);
        }

        equipMap = new Map<Id, Product2>([Select Id, Name, Cost__c,
Current_Inventory__c, Lifespan_Months__c, Maintenance_Cycle__c,
Replacement_Part__c, Warehouse_SKU__c From Product2 Where Id IN
:equipmentsToProcess]);

            for (Id caseId: casesToProcess) {
                    List<Work_Part__c> partList = reqWorkPartMap.get(caseId);
                    List<Work_Part__c> insertPartList = new List<Work_Part__c>();

                    Integer min;
                    if ( partList == null || partList.size() == 0 ) { partList = new
List<Work_Part__c>(); min = 0; }
                    for (Work_Part__c wp: partList) {
                            Product2 equipment = equipMap.get(wp.Equipment__c);
                            Integer lifeSpan = (Integer)equipment.Maintenance_Cycle__c;
                            if ( (min == null) || (lifeSpan < min) ) { min = lifeSpan; }
```

```
                    Work_Part__c newWp = new Work_Part__c();
                    newWp.Equipment__c = wp.Equipment__c;
                    newWp.Quantity__c = wp.Quantity__c;
                    insertPartList.add(newWp);
                }
                Case oldCase = caseMap.get(caseId);
                Case newCase = new Case();
                newCase.Type = MaintenanceRequestHelper.newCaseType;
                newCase.Vehicle__c = oldCase.Vehicle__c;
                newCase.Equipment__c = oldCase.Equipment__c;
                //newCase.Product__c = oldCase.Product__c;
                newCase.Subject = newCaseSubject + ' - ' + newCase.Vehicle__c + ' - ' +
newCase.Product__c;
                newCase.Date_Reported__c = System.today();
                newCase.Date_Due__c = System.today().addDays(min);
                newCase.Status = newCaseStatus;
                insertCaseMap.put(key, newCase);
                insertWorkPartMap.put(key, insertPartList);
        }
        System.debug('Cases to Insert: ' + insertCaseMap);

        insert insertCaseMap.values();

        for (Integer i: insertCaseMap.keySet()) {
                List<Work_Part__c> insertPartList = insertWorkPartMap.get(i);
                Case newC = insertCaseMap.get(i);
                for (Work_Part__c newWp: insertPartList) {
                        newWp.Maintenance_Request__c = newC.Id;
                        insertWorkPartList.add(newWp);
                }
        }
        insert insertWorkPartList;
   }

}
Add a quote, <Ctrl+Shift+.>
Add code, <Ctrl+e>
```

Add a link, <Ctrl+k>
Directly mention a user or team
Reference an issue or pull request

WarehouseCalloutService:

```
public with sharing class WarehouseCalloutService {

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

    // complete this method to make the callout (using @future) to the
    // REST endpoint and update equipment on hand.
    @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);
                // If the request is successful, parse the JSON response.
                if (response.getStatusCode() == 200) {
                // Deserialize the JSON string into collections of primitive data types.
                List<Object> equipments = (List<Object>)
JSON.deserializeUntyped(response.getBody());
            List<Product2> products = new List<Product2>();
            for(Object o :  equipments){
                Map<String, Object> mapProduct = (Map<String, Object>)o;
                Product2 product = new Product2();
                product.Name = (String)mapProduct.get('name');
                product.Cost__c = (Integer)mapProduct.get('cost');
                product.Current_Inventory__c = (Integer)mapProduct.get('quantity');
                product.Maintenance_Cycle__c =
(Integer)mapProduct.get('maintenanceperiod');
                product.Replacement_Part__c = (Boolean)mapProduct.get('replacement');
                product.Lifespan_Months__c = (Integer)mapProduct.get('lifespan');
```

```
            product.Warehouse_SKU__c = (String)mapProduct.get('sku');
            product.ProductCode = (String)mapProduct.get('_id');
            products.add(product);
        }
        if(products.size() > 0){
            System.debug(products);
            upsert products;
        }
                }
    }
}
```

WarehouseSyncSchedule:

```
global class WarehouseSyncSchedule implements Schedulable
{
   global void execute ( SchedulableContext sc )
   {
      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
```

MaintenanceRequestHelperTest:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
// createVehicle
private static Vehicle__c createVehicle(){
Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
return vehicle;
}
// createEquipment
private static Product2 createEquipment(){
product2 equipment = new product2(name = 'Testing equipment',
lifespan_months__c = 10,
maintenance_cycle__c = 10,
```

```
replacement_part__c = true);
return equipment;
}
// createMaintenanceRequest
private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
case cse = new case(Type='Repair',
Status='New',
Origin='Web',
Subject='Testing subject',
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cse;
}
// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return equipmentMaintenanceItem;
}
@isTest
private static void testPositive(){
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
Product2 equipment = createEquipment();
insert equipment;
id equipmentId = equipment.Id;
case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
insert createdCase;
Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
insert equipmentMaintenanceItem;
test.startTest();
createdCase.status = 'Closed';
```

```
update createdCase;
test.stopTest();
Case newCase = [Select id,
subject,
type,
Equipment__c,
Date_Reported__c,
Vehicle__c,
Date_Due__c
from case
where status ='New'];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newCase.Id];
list<case> allCase = [select id from case];
system.assert(allCase.size() == 2);
system.assert(newCase != null);
system.assert(newCase.Subject != null);
system.assertEquals(newCase.Type, 'Routine Maintenance');
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}
@isTest
private static void testNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
product2 equipment = createEquipment();
insert equipment;
id equipmentId = equipment.Id;
case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
insert createdCase;
Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
insert workP;
test.startTest();
```

```
createdCase.Status = 'Working';
update createdCase;
test.stopTest();
list<case> allCase = [select id from case];
Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :createdCase.Id];
system.assert(equipmentMaintenanceItem != null);
system.assert(allCase.size() == 1);
}
@isTest
private static void testBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>();
list<Product2> equipmentList = new list<Product2>();
list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList =
new list<Equipment_Maintenance_Item__c>();
list<case> caseList = new list<case>();
list<id> oldCaseIds = new list<id>();
for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle());
equipmentList.add(createEquipment());
}
insert vehicleList;
insert equipmentList;
for(integer i = 0; i < 300; i++){
caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
}
insert caseList;
for(integer i = 0; i < 300; i++){
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipment
List.get(i).id, caseList.get(i).id));
}
insert equipmentMaintenanceItemList;
test.startTest();
for(case cs : caseList){
cs.Status = 'Closed';
```

```
oldCaseIds.add(cs.Id);
}
update caseList;
test.stopTest();
list<case> newCase = [select id
from case
where status ='New'];
list<Equipment_Maintenance_Item__c> workParts = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in:
oldCaseIds];
system.assert(newCase.size() == 300);
list<case> allCase = [select id from case];
system.assert(allCase.size() == 600);
}
}
```

WarehouseCalloutServiceTest:

```
@isTest
private class WarehouseCalloutServiceTest {
  // implement your mock callout test here
        @isTest
   static void WarehouseEquipmentSync(){
      Test.startTest();
      // Set mock callout class
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      // This causes a fake response to be sent from the class that implements
HttpCalloutMock.
      WarehouseCalloutService.runWarehouseEquipmentSync();
      Test.stopTest();
      System.assertEquals(1, [SELECT count() FROM Product2]);
   }
}
```

WarehouseCalloutServiceMock:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global 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;
    }
}
```

WarehouseSyncScheduleTest:

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest
    static void test() {
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String CRON_EXP =  '0 0 0 3 9 ? 2022';
        Test.startTest();

        // Schedule the test job
        String jobId = System.schedule('testScheduledApex', CRON_EXP, new
```

```
WarehouseSyncSchedule());

        Test.stopTest();

        // Get the information from the CronTrigger API object
        CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
                          NextFireTime FROM CronTrigger WHERE id = :jobId];

        // Verify the expressions are the same
        System.assertEquals(CRON_EXP,ct.CronExpression);
        // Verify the job has not run
        System.assertEquals(0, ct.TimesTriggered);
    }
}
```
