AccountAddressTrigger:

```
trigger AccountAddressTrigger on Account(before insert,
beforeupdate) {
    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
TestTask', WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0
        ){ insert
        tasklist;
    }
}
```

VerifyDate:

```
public class VerifyDate {
          //method to handle potential checks against two
     datespublic staticDate CheckDates(Date date1,Date date2)
     {
          //if date2 is within the next 30 days of date1,
usedate2.  Otherwiseuse the end of the month
          if(DateWithin30Days(date1,date2))
               {returndate2;
          } else {
               return SetEndOfMonthDate(date1);
          }
     }
     //method to check if date2 is within the next 30 days
ofdate1
     @TestVisible private static Boolean
DateWithin30Days(Datedate1,Date date2) {
          //check for date2 being in the
     pastif( date2 < date1) { return
     false;}

     //check that date2 is within (>=) 30 days of date1
     Date date30Days = date1.addDays(30); //createa date
     30
days away from date1
          if( date2 >= date30Days ) { return false;
          }else { returntrue; }
     }

     //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
private class TestVerifyDate {
    @isTest staticvoid 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 staticvoid 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(){Booleanflag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'
),date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }
    @isTest static void
        Test_DateWithin30Days_case2(){Booleanflag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'
),date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }
    @isTest static void
        Test_DateWithin30Days_case3(){Booleanflag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'
),date.parse('01/15/2020'));
        System.assertEquals(true, flag);

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

RestrictContactByName:

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

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

TestRestrictContactByName:

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

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


          }
     }
}
```

RandomContactFactory :

```
public class RandomContactFactory {

  public static List<Contact> generateRandomContacts(Integernumcnt, 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> accountsToUpdate = new
        List<Account>();
        List<Account> accounts = [Select Id, Name, (Select
Idfrom Contacts) from AccountWhere 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:

```
public class
    AccountProcessor{@future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new
        List<Account>();
        List<Account> accounts = [Select Id, Name, (Select
Idfrom Contacts) from AccountWhere Id in :accountIds];
        for(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts___c =
            contactList.size();accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

LeadProcessor:

```
global class LeadProcessor
implements
Database.Batchable<sObject> {
    global Integer count= 0;

    global database.QueryLocator start(Database.BatchableContext

bc){
        return Database.getQueryLocator('SELECT ID, LeadSource
FROM Lead');
    }
    global void execute(Database.BatchableContext bc,
List<Lead>L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead  L:L_list){
            L.leadsource =
            'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
   }
    global void finish(Database.BatchableContext
        bc){system.debug('count = ' + count);
    }

}
```

LeadProcessorTest:

```
@isTest
public class LeadProcessorTest {

    @isTest
    public staticvoid 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();
    }
}
```
AddPrimaryContact:
```
public class AddPrimaryContact implements

    Queueable{privateContact con;
    private Stringstate;

    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
whereBillingState = :state Limit200];
        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){insertprimaryContacts;
        }
    }

}
```

AddPrimaryContactTest:

```
public class AddPrimaryContact implements
```

```
}


    Queueable{privateContact con;


    private Stringstate;


    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
whereBillingState = :state Limit200];
        List<Contact> primaryContacts = new List<Contact>();


        for(Account acc:accounts){
            Contactc =
            con.clone();


            c.AccountId = acc.Id;
            primaryContacts.add(c
            );
        }


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


}
```

DailyLeadProcessor:

```
global class DailyLeadProcessor implements
    Schedulable{globalvoid execute(SchedulableContext
    sc){
        List<Lead> lstofLead = [SELECT Id FROM Lead
WHERELeadsource = null LIMIT200];
        List<Lead> lstofupdatedLead=new
        List<Lead>();if(!lstofLead.isEmpty()){
           for (Lead ld:lstofLead){
                 ld.Leadsource='Dreamforce
                 ';
                 lstofupdatedLead.add(ld);
           }
            UPDATE lstofupdatedLead;
           }
        }
    }
```

DailyLeadProcessorTest:

```
@isTest
private class
    DailyLeadProcessorTest{
    @testSetup
     static void setup(){
          List<Lead> lstofLead = new
          List<Lead>();for(Integer i = 1; i
          <=200; i++){
          Lead ld = new Lead(Company = 'Comp' + i, LastName
='LN' + i, status='working - Contacted');
          lstofLead.add(ld);
        }
     Insert lstofLead;
    }
          static testmethod void
testDailyLeadProcessorscheduledJob(
){
               String sch = '0 5 12 * * ?';
               Test.startTest();
               String jobId =
System.Schedule('ScheduledApexText', sch,
newDailyLeadProcessor());

            List<Lead> lstofLead=[SELECT Id FROM Lead
WHERELeadsource = null LIMIT200];
               system.assertEquals(200,
               lstoflead.size());Test.stopTest();
        }
    }
```

AnimalLocator:

```
public class AnimalLocator{
    public static StringgetAnimalNameById(Integer
        x){Http http = new Http();
        HttpRequest req = new HttpRequest();

        req.setEndpoint('https://th-apex-
http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal=new Map<String,
        Object>();HttpResponse res = http.send(req);
            if(res.getStatusCode() == 200) {
        Map<String, Object> results= (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
      animal = (Map<String, Object>)results.get('animal');
        }
return (String)animal.get('name');
    }
}
```

AnimalLocatorTest:

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1()
        { Test.setMock(HttpCalloutMock.class,
        new
AnimalLocatorMock());
        string result =
        AnimalLocator.getAnimalNameById(3);String
        expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}
```

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('{"animals": ["majestic badger", "fluffy bunny", "scary
      bear",
"chicken", "mighty moose"]}');
      response.setStatusCode(200);
      return response;
   }
}
```

ParkLocator:

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

ParkLocatorTest:

```
@isTest
private class  ParkLocatorTest  {
    @isTest staticvoid 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) {
      /start - specify the response you want to send
      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};/ end
      response.put('response_x', response_x);
  }
}
```

AccountManager:

```
@RestResource(urlMapping =
'/Accounts/*/contacts')globalwith sharing
classAccountManager {

    @HttpGet
    global static Account getAccount(){
        RestRequest request =
        RestContext.request;string accountId=
request.requestURI.substringBetween('Accounts/','/contacts');
        Accountresult = [SELECTId, Name, (SelectId, Name from
Contacts)from Account whereId=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest:

```
@IsTest
private class AccountManagerTest {
    @isTest static void
        testGetContactsByAccountId(){Id recordId  =
        createTestRecord(); RestRequest request =
        new RestRequest(); request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Acco
un ts/'

               +
        recordId+'/contacts';request
        .httpMethod = 'GET';
        RestContext.request =
        request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test
        record',thisAccount.Name);
    }

    static Id createTestRecord(){
        Account accountTest = new
     Account(Name ='Test record');
        insert accountTest;

        Contact contactTest = new
     Contact(FirstName='John',
     LastName = 'Doe',
     AccountId=
     accountTest.Id

        );
        insert contactTest;

        return accountTest.Id
;
    }
}
```

MaintenanceRequest:

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

MaintenanceRequestHelper:

```
public with sharingclass MaintenanceRequestHelper
    {public staticvoid 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);



        //When an existing maintenance request of type Repair
orRoutineMaintenance is closed,
        //createa new maintenance request for a futureroutine

checkup.

if (!validIds.isEmpty()){
    Map<Id,Case> closedCases = 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)


Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new
Map<ID,Decimal>();
```

```
FROM


            //calculate the maintenance request due dates
byusing the maintenance cycle defined on the related
equipment records.
            AggregateResult[] results =
[SELECTMaintenance_Request__c,

MIN(Equipment__r.Maintenance_Cycle__c)cycle
                                          FROM
Equipment_Maintenance_Item__c
                                          WHERE
Maintenance_Request__c IN :ValidIds GROUP
BYMaintenance_Request__c];

            for (AggregateResult ar :
                results){
                maintenanceCycles.put((Id)
ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));

            }

            List<Case> newCases = new
            List<Case>();for(Case cc :
            closedCases.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 multiple pieces of equipment are used
inthe maintenance request,
                //define the due date by applying the
shortestmaintenance cycle to today’sdate.
                //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> clonedList =
newList<Equipment_Maintenance_Item_c>();
            for (Case nc : newCases){
                    for (Equipment_Maintenance_Item___c

clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Itemsr){
                         Equipment_Maintenance_Item_c item
=clonedListItem.clone();
                         item.Maintenance_Request_c =
                         nc.Id;clonedList.add(item);
                    }
               }
               insert clonedList;
          }
     }
}
```

WarehouseCalloutService:

```
public with sharing class WarehouseCalloutService implements Queueable {
   privatestatic final String WAREHOUSE_URL = 'https:/ th-superbadge-
apex.herokuapp.com/equipment';

   / Write a classthat makes a REST calloutto an external warehouse systemto get a list of
equipment that needs to be updated.
   / The callout's JSON response returns the equipmentrecords that you upsert in Salesforce.

   @future(callout=true)
   public static void runWarehouseEquipmentSync(){
      System.debug('go into
      runWarehouseEquipmentSync'); Http http = new
      Http();
      HttpRequest request = new HttpRequest();

      request.setEndpoint(WAREHOUSE_URL);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
```

```
        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() ==
           200){List<Object> jsonResponse
           =
(List<Object>)JSON.deserializeUntyped(response.getBody());

           System.debug(response.getBody());

           / class maps the following fields:
           / warehouseSKU will be external ID for identifying which equipment recordsto update
withinSalesforce
           for (Object jR : jsonResponse){
              Map<String,Object> mapJson =
              (Map<String,Object>)jR;Product2product2 = new
              Product2();
              / replacement part (always true),
              product2.Replacement_Part_c = (Boolean) mapJson.get('replacement');
              / cost
              product2.Cost_c = (Integer) mapJson.get('cost');
              / current inventory
              product2.Current_Inventory_c = (Double) mapJson.get('quantity');
              / lifespan
              product2.Lifespan_Months_c = (Integer) mapJson.get('lifespan');
              / maintenance cycle
              product2.Maintenance_Cycle_c = (Integer) mapJson.get('maintenanceperiod');
              / warehouse SKU
              product2.Warehouse_SKU_c = (String) mapJson.get('sku');

              product2.Name = (String) mapJson.get('name');
              product2.ProductCode = (String) mapJson.get('_id');
              product2List.add(product2);
           }

           if (product2List.size() > 0){
              upsertproduct2List;
              System.debug('Your equipment was synced with the warehouse one');
```

```
            }
        }
    }

    public    static    void    execute    (QueueableContext
        context){System.debug('start
        runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end  runWarehouseEquipmentSync');
    }

}
```

WarehouseSyncSchedule:

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

MaintenanceRequest:

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

MaintenanceRequestHelper:

```
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);
                }
            }
        }

        / When an existing maintenance request of type Repair or Routine Maintenance is closed,

        / createa new maintenance request for a futureroutine
        checkup.if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = 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>();

            / calculate the maintenance requestdue dates by using the maintenance cycledefined
on the related equipment records.
            AggregateResult[] results = [SELECT Maintenance_Request_c,
                                        MIN(Equipment_r.Maintenance_Cycle_c)cycle
                                        FROM Equipment_Maintenance_Item_c
                                        WHERE Maintenance_Request_cIN :ValidIds GROUP BY
Maintenance_Request_c];

            for (AggregateResult ar : results){
                maintenanceCycles.put((Id) ar.get('Maintenance_Request_c'), (Decimal)
ar.get('cycle'));
            }

            List<Case> newCases= new List<Case>();
            for(Case cc : closedCases.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 multiple piecesof equipment are used in the maintenance request,
                / define the due date by applying the shortest maintenance cycle to today’sdate.
                / 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> clonedList = new
List<Equipment_Maintenance_Item_c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item_c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items_r){
                    Equipment_Maintenance_Item_c item = clonedListItem.clone();
                    item.Maintenance_Request_c= nc.Id;
                    clonedList.add(item);
                }
            }
            insert clonedList;
        }
    }
}
```

MaintenanceRequestHelperTest:

```
@isTest
public with sharingclass MaintenanceRequestHelperTest {

   // createVehicle
   private static Vehicle__c createVehicle(){
      Vehicle__c vehicle = new Vehicle__C(name = 'Testing
      Vehicle');return vehicle;
   }

   // createEquipment
   private static Product2createEquipment(){
      product2 equipment = new product2(name = 'Testing equipment',




      return equipment;
   }

 lifespan_months__c = 10,
 maintenance_cycle__c = 10, replacement_part__c
 = true);

   // 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 =
newEquipment_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____
               cfrom 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;
        insertequipmentLis
        t;


        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(equipme
nt List.get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;


        test.startTest();
        for(casecs :
        caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.I
            d);
        }
        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);
    }
}
```

WarehouseCalloutService:

```
public with sharing class WarehouseCalloutService implements Queueable {
    privatestatic final String WAREHOUSE_URL = 'https:/ th-superbadge-
apex.herokuapp.com/equipment';

    / Write a classthat makes a REST calloutto an external warehouse systemto get a list of
equipment that needs to be updated.
    / The callout's JSON response returns the equipmentrecords that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into
        runWarehouseEquipmentSync'); Http http = new
        Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() ==
            200){List<Object> jsonResponse
            =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            / class maps the following fields:
            / warehouseSKU will be external ID for identifying which equipment recordsto update
withinSalesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson =
                (Map<String,Object>)jR;Product2product2 = new
                Product2();
                / replacement part (always true),
                product2.Replacement_Part_c = (Boolean) mapJson.get('replacement');
                / cost
                product2.Cost_c = (Integer) mapJson.get('cost');
                / current inventory
                product2.Current_Inventory_c = (Double) mapJson.get('quantity');
                / lifespan
                product2.Lifespan_Months_c = (Integer) mapJson.get('lifespan');
                / maintenance cycle
                product2.Maintenance_Cycle_c = (Integer) mapJson.get('maintenanceperiod');
                / warehouse SKU
                product2.Warehouse_SKU_c = (String) mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsertproduct2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public   static   void   execute   (QueueableContext
        context){System.debug('start
        runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end  runWarehouseEquipmentSync');
    }

}
```

WarehouseCalloutServiceTest:

```
@IsTest
private class WarehouseCalloutServiceTest {
   / implement your mock callout test
          here@isTest
   static void testWarehouseCallout() {
      test.startTest();
      test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      WarehouseCalloutService.execute(null);
      test.stopTest();


      List<Product2> product2List = new List<Product2>();
      product2List = [SELECT ProductCode FROM
      Product2];

      System.assertEquals(3, product2List.size());
      System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
      System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
      System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
   }
}
```

WarehouseCalloutServiceMock:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   /  implement http mock callout
   global static HttpResponse respond(HttpRequest
      request){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"},{"_id":"55d66226726b6
11 100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
```

```
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseSyncSchedule:

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

WarehouseSyncScheduleTest:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    /  implement scheduledcode here
    /
    @isTeststatic void test(){
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test',scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobIddoes not match');

        Test.stopTest();
    }
}
```