# SalesForce Developer Catalyst And SuperBadges

## Apex Triggers

1.Getting Started with Apex Triggers

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

2.Bulk Triggers

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
    List<Task> listoftasks=new List<Task>();
    for(Opportunity opp:Trigger.New){
        if(opp.StageName=='Closed Won'){
            listoftasks.add(new Task(Subject='Follow Up Test

        }
    }
    insert listoftasks;
}
```

## Apex Testing

1.Get Started with Apex Unit Test

Apex Class- TestVerifyDate:

```
@istest
public class TestVerifyDate {
@isTest static void test1() {
    Date
d3=VerifyDate.CheckDates(Date.parse('01/01/2021'),Date.parse('01/

    System.assertEquals(Date.parse('01/03/2021'), d3);
    }

    @isTest static void test2(){
Date
d3=VerifyDate.CheckDates(Date.parse('01/01/2021'),Date.parse('03/

    System.assertEquals(Date.parse('01/31/2021'), d3);
    }
}
```

Apex Class - VerifyDate:

```
public class VerifyDate {
 public static Date CheckDates(Date date1, Date date2) {
       if(DateWithin30Days(date1,date2)) {
             return date2;
       } else {
             return SetEndOfMonthDate(date1);
       }
 }
 private static Boolean DateWithin30Days(Date date1, Date date2)
{
 if( date2 < date1) { return false; }

 Date date30Days = date1.addDays(30);
```

```
    away from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }

    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;
    }

}
```

2. Test Apex Triggers

Apex Class -TestRestrictContactByName

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('invalid last
;

    }
}
```

Apex Class-RestrictContactByName

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

 For (Contact c : Trigger.New) {
       if(c.LastName == 'INVALIDNAME') {
            c.AddError('The Last Name "'+c.LastName+'" is not

       }

 }
}
```

3. Create Test Data for Apex Tests

Apex Class-RandomContactFactory

```apex
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num, String name) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<num;i++) {
            Contact c = new Contact(FirstName=name + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        System.debug(contactList.size());
        return contactList;
    }

}
```

## Asynchronous Apex

2. Use Future Methods

```
public without sharing class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds) {

        List<Account> accounts = [SELECT Id, (SELECT Id FROM Contacts) FROM Account WHERE Id IN :accountIds];

        for (Account acc : accounts) {
            acc.Number_of_Contacts__c = acc.Contacts.size();
        }
        update accounts;
    }
}
```

```
@isTest
private class AccountProcessorTest {

    @isTest
    private static void countContactsTest() {
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<300; i++) {
            accounts.add(new Account(Name='Test Account' + i));
        }
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for (Account acc: accounts) {
            contacts.add(new Contact(FirstName=acc.Name, LastName='TestContact', AccountId=acc.Id));
            accountIds.add(acc.Id);
        }
        insert contacts;
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        List<Account> accs = [SELECT Id, Number_of_Contacts__c FROM Account];
        for (Account acc : accs) {
            System.assertEquals(1, acc.Number_of_Contacts__c, 'ERROR: At least 1 Account record with incorrect Contact count');
        }
    }
}
```

3. Use Batch Apex

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

    public Integer recordCount = 0;
 public Database.QueryLocator start(Database.BatchableContext dbc){
       System.debug('Filling the bucket');
       return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
 }
 public void execute(Database.BatchableContext dbc, List<Lead> leads){
       for (Lead l : leads) {
            l.LeadSource = 'Dreamforce';
       }
       update leads;
        recordCount = recordCount + leads.size();
       System.debug('Records processed so far ' + recordCount);
 }
 public void finish(Database.BatchableContext dbc){
        System.debug('Total records processed ' + recordCount);
 }
}
```

```
@isTest
private class LeadProcessorTest {

    @isTest
    private static void testBatchClass() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
            leads.add(new Lead(LastName='Connock',
Company='Salesforce'));
        }
        insert leads;
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();
        List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE
LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At

    }
}
```

4. Control Processes with Queueable Apex

```
public without sharing class AddPrimaryContact implements
Queueable {

    private Contact contact;
    private String state;

    public AddPrimaryContact(Contact inputContact, String
inputState) {
        this.contact = inputContact;
        this.state = inputState;
    }

    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

        List<Contact> contacts = new List<Contact>();
        for ( Account acc : accounts) {
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }
}
```

```
@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass() {
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<500; i++) {
            Account acc = new Account(Name='Test Account');
            if ( i<250 ) {
                acc.BillingState = 'NY';
            } else {
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact = new Contact(FirstName='Simon', LastName='Connock');
        insert contact;
        Test.startTest();
        Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));
        Test.stopTest();
        List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState = 'CA'];
        System.assertEquals(200, contacts.size(), 'ERROR:

    }
}
```

5. Scheduable

```
public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext ctx) {
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT 200];
        for ( Lead l : leads) {
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}
```

```
@isTest
private class DailyLeadProcessorTest {

    private static String CRON_EXP = '0 0 0 ? * * *';

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

```
          }
      }
      insert leads;
      Test.startTest();
      String jobId = System.schedule('Process Leads', CRON_EXP, new DailyLeadProcessor());
      Test.stopTest();
      List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = 'Dreamforce'];
      System.assertEquals(200, updatedLeads.size(), 'ERROR: At

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

## Apex Integration Services

AnimalLocator.apxc

```
public class AnimalLocator {
 public class cls_animal {
     public Integer id;
     public String name;
     public String eats;
     public String says;
 }
public class JSONOutput{
 public cls_animal animal;

}

   public static String getAnimalNameById (Integer id) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-

      request.setMethod('GET');
      HttpResponse response = http.send(request);
      system.debug('response: ' + response.getBody());
      jsonOutput results = (jsonOutput)
JSON.deserialize(response.getBody(), jsonOutput.class);
     system.debug('results= ' + results.animal.name);
      return(results.animal.name);
   }

}
```

AnimalLocatorMock.apxc

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chic

        return response;
    }

}
```

AnimalLocatorTest.apxc

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

ParkService.apxc

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-

        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-

        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
```

```
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new 
ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

ParkServiceMock.apxc

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

        response.put('response_x', response_x);
    }
}
```

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
    }
}
```

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        RestRequest request = new RestRequest();
        request.requestUri =

'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account  acc = AccountManager.getAccount();
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

## Apex SuperBadge

Challenge 1:

MaintenanceRequest.apxt

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

MaintenanceRequestHelpher.cls

```
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

                    validIds.add(c.Id);
                }
            }
        }

        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>();

            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'));
            }

            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()
                );
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                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;
        }
    }
}
```

Challenge 2:

WarehouseCallout.

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-

    @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());

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                product2.Cost__c = (Integer) mapJson.get('cost');
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                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){
                upsert product2List;
                System.debug('Your equipment was synced with the

            }
        }
    }

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

}
```

Challenge 3:

WarehouseSyncSchedule

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

Challenge 4:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing

        return vehicle;
    }
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing

                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }
    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;
    }
    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(e
        }
        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);
    }
}
```

Challenge 5:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    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":"55d66226726b611100aaf742",
                             "replacement":true,"quantity":183,
                                 "name":"Cooling Fan",
                                     "maintenanceperiod":0,
"lifespan":0,"cost":300,"sku":"100004"},
                         {"_id":"55d66226726b611100aaf743",
                             "replacement":true,"quantity":143,
                                 "name":"Fuse

"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

```
@IsTest
private class WarehouseCalloutServiceTest {
  @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);
    }
}
```

Challenge 6

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