Apex Triggers

Get Started with Apex Triggers

AccountAddressTrigger

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

Bulk Apex Triggers

ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

    List<Task> taskToOpp = new List<Task>();

  for (Opportunity o : [ SELECT Id,StageName FROM Opportunity WHERE
StageName = 'Closed Won' AND Id IN :Trigger.New ]) {
    taskToOpp.add(new Task( Subject = 'Follow Up Test Task', WhatId = o.Id));
    }

    if (taskToOpp.size() > 0)
      insert taskToOpp;

}
```

Apex Testing

Get Started with Apex Unit Tests

TestVerifyDate

```
@isTest
private class TestVerifyDate {


@isTest static void testDate2within30daysofDate1() {
Date date1 = date.newInstance(2018, 03, 20);
Date date2 = date.newInstance(2018, 04, 11);
Date resultDate = VerifyDate.CheckDates(date1,date2);
Date testDate = Date.newInstance(2018, 04, 11);
System.assertEquals(testDate,resultDate);
}


@isTest static void testDate2beforeDate1() {
Date date1 = date.newInstance(2018, 03, 20);
Date date2 = date.newInstance(2018, 02, 11);
Date resultDate = VerifyDate.CheckDates(date1,date2);
```

```
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }

    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

Test Apex Triggers

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('Cannot create contact with invalid last name.',
                            result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Tests

RandomContactFactory

```
public class RandomContactFactory {

Public Static List<Contact> generateRandomContacts(integer noOfContact, String
```

```
lastName)
{
List<Contact> con=New list<Contact>();
for(Integer i=0;i<noOfContact;i++)
{
Contact c = new Contact(FirstName='Ank' + i,LastName=lastName);
Con.add(c);
}

Return con;
}
}
```

Asynchronous Apex

Use Future Methods

AccountProcessor

```
public class AccountProcessor {

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

Map<Id,Integer> account_cno = new Map<Id,Integer>();
List<account> account_lst_all = new List<account>([select id, (select id from
contacts) from account]);
for(account a:account_lst_all) {
```

```
        account_cno.put(a.id,a.contacts.size());        }

    List<account> account_lst = new List<account>();
        for(Id accountId : accountId_lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c = account_cno.get(accountId);
                account_lst.add(acc);
            }

        }
        upsert account_lst;
    }

}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
```

```
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;


        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
        AccountProcessor.countContacts(acc_list);
        Test.stopTest();
    List<account> acc1 = new List<account>([select Number_of_Contacts__c from
account where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }

}
```

Use Batch Apex

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_lst) {
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
            count+=1;
        }
        update l_lst_new;
    }

    global void finish (Database.BatchableContext bc) {
        system.debug('count = '+count);
    }
}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit() {
        List<lead> l_lst = new List<lead>();
        for (Integer i = 0; i<200; i++) {
            Lead l = new lead();
            l.LastName = 'name'+i;
```

```
            l.company = 'company';
            l.Status =  'somestatus';
            l_lst.add(l);
        }
        insert l_lst;

        test.startTest();

        Leadprocessor lp = new Leadprocessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();

    }

}
```

Control Processes with Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
```

```
    }

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}
```

AddPrimaryContactTest

```
@IsTest
public class AddPrimaryContactTest {

    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
```

```
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }

}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =
''];

        if(leads.size() > 0){
```

```
        List<Lead> newLeads = new List<Lead>();

        for(Lead lead : leads){
            lead.LeadSource = 'DreamForce';
            newLeads.add(lead);
        }

        update newLeads;
    }
  }
}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
  //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
  public static String CRON_EXP = '0 0 0 2 6 ? 2022';

  static testmethod void testScheduledJob(){
    List<Lead> leads = new List<Lead>();

    for(Integer i = 0; i < 200; i++){
        Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');
        leads.add(lead);
    }
```

```
        insert leads;

        Test.startTest();
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
         Test.stopTest();
    }
}
```

Apex Integration Services

Apex REST Callouts

AnimalLocator

```
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-callout.herokuapp.com/animals/' + id);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        JSON.deserializeUntyped(response.getBody());
    jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
        jsonOutput.class);

        system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }

}
```

AnimalLocatorTest

```
@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);
    }

}
```

AnimalLocatorMock

```
@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":"chicken
        food","says":"cluck cluck"}}');
        return response;
    }

}
```

Apex SOAP Callouts

ParkLocator

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

ParkLocatorTest

```
@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

```
@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);
    }
}
```

Apex Web Services

AccountManager

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

    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
    string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
    Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from
    Account where Id=: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/Accounts/'
            + 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;
    }
}
```

Apex Specialist

CHALLENGE 1

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));
            } 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;
       }
    }
}
```

MaitenanceRequest.apxt

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

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

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

    }

}
```

CHALLENGE 2

WarehouseCalloutService.apxc

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



    @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 warehouse one');
        }
    }
}

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

}

Execute Anonymous Window

System.enqueueJob(new WarehouseCalloutService());

CHALLENGE 3

WarehouseSyncShedule.apxc

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

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 REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_SUBJECT = 'Testing subject';
```

```
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REPAIR = 'Repair';
    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);
    }
}
```

CHALLENGE 5

WarehouseCalloutServiceTest.apxc

```
@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);
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@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":"55d6
6226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d662267
26b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

CHALLENGE 6

WarehouseSyncScheduleTest.apxc

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static 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), 'JobId does not match');

        Test.stopTest();
    }
}
```