Apex Triggers

Get Started with Apex Triggers

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

Bulk Apex Triggers

```
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;
}
}
```

Apex Testing

Get Started with Apex Unit Tests

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

```
System.assertEquals(false,flag);
}
@isTest static void Test_DateWithin30Days_case3(){
Boolean flag =
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'));
}
}
```

Test Apex Triggers

```
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');
}
}
}
@isTest
public class TestRestrictContactByName {
@isTest static void Test_insertupdateContact(){
Contact cnt = new Contact();
cnt.LastName = 'INVALIDNAME';
Test.startTest();
Database.SaveResult result = Database.insert(cnt,false);
Test.startTest();
System.assert(!result.isSuccess());
System.assert(result.getErrors().size()>0);
System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getmessage());
}
}
```

Create Test Data for Apex Tests

```
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;
}
}
```

Asynchronous Apex

Use Future Methods

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

Use Batch Apex

```
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);
}
}
@isTest
public class LeadProcessorTest {
@testSetup
static void setup() {
List<Lead> leads = new List<Lead>();
for(Integer counter=0 ;counter <200;counter++){
```

```
Lead lead = new Lead();
lead.FirstName ='FirstName';
lead.LastName ='LastName'+counter;
lead.Company ='demo'+counter;
leads.add(lead);
}
insert leads;
}
@isTest static void test() {
Test.startTest();
LeadProcessor leadProcessor = new LeadProcessor();
Id batchId = Database.executeBatch(leadProcessor);
Test.stopTest();
}
}
```

Control Processes with Queueable Apex

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

Schedule Jobs Using the Apex Scheduler

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

Apex Integration Services

Apex REST Callouts

```
public class AnimalLocator{
public static String getAnimalNameById(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');
}
}
@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 );
}
}
@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;
}
}
```

Apex SOAP Callouts

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

Apex Web Services

```
@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;
}
}
@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 2

```
trigger MaintenanceRequest on Case (before update, after update) {
//ToDo: Call MaintenanceRequestHelper.updateWorkOrders
if(trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders();
}
```

```
}
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,
//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle_c, Equipmentc,
Equipmentr.Maintenance_Cycle_c,
(SELECT Id,Equipment_c,Quantityc FROM
Equipment_Maintenance_Items_r)
FROM Case WHERE Id IN :validIds]);
Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
//calculate the maintenance request due dates by using the maintenance cycle defined
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_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()
);
//If multiple pieces of equipment are used in the maintenance request,
//define the due date by applying the shortest maintenance cycle to today’s date.
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.Equipmentr.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;
}
}
}
```

challenge 3

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

```
//Write a class that makes a REST callout to an external warehouse system to get a list
of
equipment that needs to be updated.
//The callout's JSON response returns the equipment records that you upsert in
Salesforce.
@future(callout=true)
public static void runWarehouseEquipmentSync(){
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:
//warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
for (Object jR : jsonResponse){
Map<String,Object> mapJson = (Map<String,Object>)jR;
Product2 product2 = 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){
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');
}
}
```

challenge 4

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

challenge 5

```
@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(equipmentList.
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);
}
}
```

challenge 6

```
@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":"55d66226
726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
response.setStatusCode(200);
return response;
}
}
@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);
}
}
```

challenge 7

```
@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":"55d66226
726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
response.setStatusCode(200);
return response;
}
}
```

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
global void execute(SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService());
}
}
@isTest
public with sharing class WarehouseSyncScheduleTest {
// implement scheduled code here
//
@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();
}
}
```