Get Started With Apex Triggers

AccountAddressTrigger.apxt:

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

Bulk Apex Triggers

ClosedOpportunityTrigger.apxt:

```
 trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List taskList = new List();
 //Iterate through the input records.
 for(Opportunity opp: Trigger.new) {
 // Check if the StageName is Closed Won and isChanged incase of update.
if(opp.StageName == 'Closed Won' && (Trigger.isInsert || opp.StageName !=
Trigger.oldMap.get(opp.Id).StageName)) {
 taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
 }
 }
 // Check if the taskList is empty or not.
if(!taskList.isEmpty()){
 insert taskList;
 }
}
```

Get Started With Apex Unit Tests

VerifyDate.apxc:

```
public class VerifyDate {
 //method to handle potential checks against two dates public static Date
CheckDates(Date date1, Date date2) {
```

```
//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the
month
 if(DateWithin30Days(date1,date2)) {
 return date2;
 } else {
       return SetEndOfMonthDate(date1);
 }
 }
 //method to check if date2 is within the next 30 days of date1
 @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
 //check for date2 being in the past
 if( date2 < date1) { return false; }
 //check that date2 is within (>=) 30 days of date1
 Date date30Days = date1.addDays(30); //create a date 30 days away from date1
 if( date2 >= date30Days ) { return false
 } else { return true;
 }
 }
 //method to return the end of the month of a given date @TestVisible private static
Date SetEndOfMonthDate(Date date1) {
Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
 return lastDay;
}
 }
```

TestVerifyDate.apxc:

```
 @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('02/02/2019'));
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') );
 }
 }
```

Create Test Data For Apex Tests

RandomContactFactory.apxc:

```
 public class RandomContactFactory {
 public static List generateRandomContacts(Integer accountsToUpdatenumcnt,string
lastname){
 List contacts = new List();
for(Integer i=0;i accountIds){
 List  = new List(); List accounts = [Select Id,Name,(Select Id from Contacts) from
Account Where Id in :accountIds];
For(Account acc:accounts){ List contactList = acc.Contacts;
acc.Number_Of_Contacts__c = contactList.size();
accountsToUpdate.add(acc);
 }
 Update accountsToUpdate;
}
}
```

AccountProcessorTest.apxc:

```
@IsTest
public class AccountProcessorTest {
@IsTest
private static void testCountContacts(){
Account newAccount = new Account(Name='Tes Account');
insert newAccount;
Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
insert newContact1;
Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
insert newContact2;
List accountIds = new List(); accountIds.add(newAccount.Id);
Test.startTest();
AccountProcessor.countContacts(accountIds);
Test.stopTest();
}
}
```

Use Batch Apex

LeadProcessor.apxc:

```
global class LeadProcessor implements Database.Batchable {
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 l_lst) {
List l_lst_new = new List();
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.apxc:

```
 @isTest
 public class LeadProcessorTest {
@isTest
 public static void testit() {
List l_lst = new List();
 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.apxc:

```
 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 ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts )
FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
List lstContact = new List();
 for (Account acc:ListAccount)
 Contact cont = c.clone(false,false,false,false);
cont.AccountId = acc.id;
 lstContact.add( cont );
 }
 if(lstContact.size() >0 )
 {

 insert lstContact;
 }
 }
 }
```

AddPrimaryContactTest.apxc:

```
 @isTest
 public class AddPrimaryContactTest {
 @isTest static void TestList()
 {
 List Teste = new List (); 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';
Test.startTest(); System.enqueueJob(apc); Test.stopTest();
}
}
```

Schedule Jobs Using Apex Scheduler

DailyLeadProcessor.apxc:

```
 global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
 //Retrieving the 200 first leads where lead source is in blank.
 List leads = [SELECT ID, LeadSource FROM Lead where LeadSource = '' LIMIT 200];
//Setting the LeadSource field the 'Dreamforce' value.
for (Lead lead : leads) {
 lead.LeadSource = 'Dreamforce'
}
 //Updating all elements in the list.
 update leads;
}
 }
```

DailyLeadProcessorTest.apxc:

```
 @isTest
 private class DailyLeadProcessorTest
{
@isTest
public static void testDailyLeadProcessor(){
 //Creating new 200 Leads and inserting them.
List leads = new List(); for (Integer x = 0; x < 200; x++) {
leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
insert leads;
//Starting test. Putting in the schedule and running the DailyLeadProcessor execute
method.
Test.startTest();
 String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new
DailyLeadProcessor());
 Test.stopTest();
//Once the job has finished, retrieve all modified leads. List listResult = [SELECT ID,
LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT 200];
 //Checking if the modified leads are the same size number that we created in the start
of this method.
 System.assertEquals(200, listResult.size());
}
 }
```

Apex REST Callouts

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 JSONOutput parse(String json){
//return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
 //}
 }
 public static String getAnimalNameById (Integer id){
  Http http = new Http();
 HttpRequest request = new HttpRequest();
 request.setEndpoint('https://th-apex-httpcallout.herokuapp.com/animals/' + id);
//request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
request.setMethod('GET');
 HttpResponse response = http.send(request);
 system.debug('response: ' + response.getBody());
//Map map_results = (Map) JSON.deserializeUntyped(response.getBody());
jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
 //Object results = (Object) map_results.get('animal');
 system.debug('results= ' + results.animal.name); return(results.animal.name);
}
 }
```

AnimalLocatorTest.apxc:

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

AnimalLocatorMock.apxc:

```
 @IsTest
 global class AnimalLocatorMock implements HttpCalloutMock {
 global HTTPresponse respond(HTTPrequest request) {
Httpresponse response = new Httpresponse();
 response.setStatusCode(200);
 //-- directly output the JSON, instead of creating a logic
//response.setHeader('key, value)
 //Integer id = Integer.valueof(request.getHeader('id'
//Integer id = 1; //List lst_body = new List {'majestic badger', 'fluffy bunny'};
//system.debug('animal return value: ' + lst_body[id]);
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chi cken food","says":"cluck
cluck"}}');
 return response;
}
}
```

Apex SOAP Callouts

ParkLocator.apxc:

```
public class ParkLocator {
public static List country(String country){
ParkService.ParksImplPort park = new ParkService.ParksImplPort();
return park.byCountry(country);
}
}
```

ParkLocatorTest.apxc:

```
@isTest
private class ParkLocatorTest {
@isTest
static void testParking() {
// This causes a fake response to be generated Test.setMock(WebServiceMock.class,
new ParkServiceMock());
// Call the method that invokes a callout String[] parkingKraj =
ParkLocator.country('Japan');
// Verify that a fake result is returned System.assertEquals(new String[]{'Shiretoko
National Park', 'Oze National Park', 'Hakusan National Park'}, parkingKraj);
}
}
```

ParkServiceMock.apxc:

```
@isTest
global class ParkServiceMock implements WebServiceMock {
global void doInvoke(
Object stub,
Object request,
Map response,
String endpoint,
String soapAction,
String requestName,
String responseNS,
 String responseName,
 String responseType) {
 ParkService.byCountryResponse odp = new ParkService.byCountryResponse ();
odp.return_x = new String[]{'Shiretoko National Park', 'Oze National Park', 'Hakusan
National Park'};
// Create response element from the autogenerated class.
 // Populate response element.

 // Add response element to the response parameter, as follows:
response.put('response_x', odp);
}

}
```

ParkService.apxc:

```
 //Generated by wsdl2apex
 public class ParkService {
 public class byCountryResponse {
 public String[] return_x
private String[] return_x_type_info = new String[]{'return','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[]{'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 e
ndpoint_x = 'https://th-apex-soapservice.herokuapp.com/service/parks'; public Map
inputHttpHeaders_x;
 public Map 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 response_map_x = new Map(); 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;
}
}
}

Apex Web Services

AccountManager.apxc:

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

}

}

AccountManagerTest.apxc:

@private class AccountManagerTest {
@isTest static void testGetAccount ()
Id recordId = createTestRecord ();
RestRequest request = new RestRequest ();

request.requestUri = 'https://yourInstance.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 testAccount = new Account (Name = 'Test Record'); insert testAccount; Contact testContact = new Contact (AccountId =

```
testAccount.Id); return testAccount.Id;
 }
 }
```

Automate Record Creation

MaintenanceRequest.apxt:

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

 }
```

MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
 public static void updateworkOrders(List updWorkOrders, Map nonUpdCaseMap) {
 Set validIds = new S
et(); 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 newCases = new List(); Map closedCasesM = new
Map([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
maintenanceCycles = new Map(); 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)); } newCases.add(nc); }
insert newCases; List clonedWPs = new List(); 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; } } }

Synchronize Salesforce Data With An External System

WarehouseCalloutService.apxc:

public with sharing class WarehouseCalloutService implements Queueable { private static final String WAREHOUSE_URL = 'https://thsuperbadge-apex.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 product2List = new List(); System.debug(response.getStatusCode()); if (response.getStatusCode() == 200){ List jsonResponse = (List)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 mapJson = (Map)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'); } } Open Execute Anonymous Window: WarehouseCalloutService.runWarehouseEquipmentSync();

Schedule Synchronization

WarehouseSyncSchedule.apxc:

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

Test Automation Logic

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 REPAIR = 'Repair'; private
static final string REQUEST_ORIGIN = 'Web'; private static final string REQUEST_TYPE =
'Routine Maintenance'; private static final string REQUEST_SUBJECT = 'Testing subject';
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 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 vehicleList = new list(); list equipmentList = new
list(); list workPartList = new list(); list requestList = new list(); list oldRequestIds = new
list(); 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 allRequests =
[select id from case where status =: STATUS_NEW]; list workParts = [select id from
Equipment_Maintenance_Item__c where Maintenance_Request__c in: oldRequestIds];
system.assert(allRequests.size() == 300); } } MaintenanceRequestHelper.apxc: public
with sharing class MaintenanceRequestHelper { public static void
updateworkOrders(List updWorkOrders, Map nonUpdCaseMap) { Set validIds = new
Set(); 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 newCases = new List(); Map
closedCasesM = new Map([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
maintenanceCycles = new Map(); 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)); } newCases.add(nc); } insert newCases; List clonedWPs = new List(); 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; } } }

MaintenanceRequest.apxt:

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

Test Callout Logic

WarehouseCalloutService.apxc:

```
 public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://thsuperbadge-apex.herokuapp.com/equipment';
//@future(callout=true) public static void runWarehouseEquipmentSync(){ Http http = new Http();
HttpRequest request = new HttpRequest();
 request.setEndpoint(WAREHOUSE_URL);
 request.setMethod('GET');
 HttpResponse response = http.send(request);
 List warehouseEq = new List();
if (response.getStatusCode() == 200){ List jsonResponse = (List)JSON.deserializeUntyped(response.getBody());
System.debug(response.getBody());
 for (Object eq : jsonResponse){ Map mapJson = (Map)eq; Product2 myEq = new Product2();
myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
 myEq.Name = (String) mapJson.get('name'); myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod'); myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan'); myEq.Cost__c = (Decimal) mapJson.get('lifespan');
myEq.Warehouse_SKU__c = (String) mapJson.get('sku'); myEq.Current_Inventory__c =
(Double) mapJson.get('quantity'); warehouseEq.add(myEq); } if (warehouseEq.size() >
0){ upsert warehouseEq; System.debug('Your equipment was synced with the
warehouse one'); System.debug(warehouseEq); } } } }
```

WarehouseCalloutServiceTest.apxc:

```
@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 product2List = new List();
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 { //
implement http mock callout global static HttpResponse respond(HttpRequest request)
{ HttpResponse response = new HttpResponse(); response.setHeader('Content-Type',
'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"},{"_id":"55d66226726b611100aaf742","replacement":true,"qua
ntity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004
"},{"_id":"55d66226726b611100aaf743","replacement":true,"quantit y":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005" }]');
response.setStatusCode(200); return response; } }
```

Test Scheduling Logic

WarehouseSyncSchedule.apxc:

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

WarehouseSyncScheduleTest.apxc:

```
@isTest
 public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
 String scheduleTime = '00 00 01 * * ?'; Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
 String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime,
new WarehouseSyncSchedule());
 Test.stopTest();
//Contains schedule information for a scheduled job. CronTrigger is similar to a cron job
on UNIX systems.
 // This object is available in API version 17.0 and later. CronTrigger a=[SELECT Id FROM
CronTrigger where NextFireTime > today]; System.assertEquals(jobID, a.Id,'Schedule ');
 }
 }
```

Test Apex Triggers

RestrictContactByName.apxt:

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

TestRestrictContactByName.apxc:

```
 @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.stopTest();
 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());

 }
}
```