

Name: Raparthy Abhishek

APEX TRIGGERS

•GET STARTED WITH APEX TRIGGERS:

1.AccountAddressTrigger.apxt

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

•BULK APEX TRIGGERS:

1.ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update)  
{  
    List<Task> taskList = new List<Task>();  
    for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE  
        StageName='Closed Won' AND Id IN : Trigger.New]){  
        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 TEST:

1.VerifyDate.apxc

```
public class VerifyDate {  
    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);
}
}

private static Boolean DateWithin30Days(Date date1, Date date2) {
    Date date30Days = date1.addDays(30); //create a date 30 days 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.TestVerifyDate.apxc

```

@Test
private class TestVerifyDate {
@Test static void testCheckDates() {
    Date now = Date.today();
    Date lastOfTheMonth = Date.newInstance(now.year(), now.month(),
Date.daysInMonth(now.year(), now.month())); Date plus60 =
Date.today().addDays(60);
    Date d1 = VerifyDate.CheckDates(now, now); System.assertEquals(now,
d1);
    Date d2 = VerifyDate.CheckDates(now, plus60);
    System.assertEquals(lastOfTheMonth, d2);
}}

```

•TEST APEX TRIGGERS:

1.RestrictContactByName.apxt

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

•CREATE TEST DATA FOR APEX TESTS:

1.RandomContactFactory.apxc

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num,
String lastName) {
        List<Contact> contacts = new List<Contact>(); for (Integer i = 0; i < num;
i++) {
            Contact c = new Contact(FirstName=i.format(), LastName=lastName);
            contacts.add(c); }
        return contacts; }
}
```

ASYNCHRONOUS APEX

•USE FUTURE METHODS:

1.AccountProcessor.apxc

```
public without sharing class AccountProcessor {
    //Add annotation to declare a future method @future(callout=false)
    public static void countContacts(List<Id> accountIds){ //Query all accounts
in the list of Ids passed
        Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id,
(SELECT Id FROM Contacts) FROM Account WHERE Id IN:accountIds]);
        List<Account> listName = new List<Account>(); //Loop through list of
accounts
        for(Account a: accountMap.values()){
```

```

//Assign field to number of contact
a.Number_of_Contacts__c=accountMap.get(a.Id).Contacts.size();
}
//Update Accounts
update accountMap.values(); }
}

```

2.AccountProcessorTest.apxc

```

@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:

1.LeadProcessor.apxc

```

global class LeadProcessor implements Database.Batchable<sObject>,
Database.Stateful {
// instance member to retain state across transactions
global Integer recordsProcessed = 0;
global Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
}
}

```

```

    global void execute(Database.BatchableContext bc, List<Lead> scope){
    // process each batch of records List<Lead> leads = new List<Lead>(); for
    (Lead lead : scope) {
    lead.LeadSource = 'Dreamforce';
    // increment the instance member counter recordsProcessed =
    recordsProcessed + 1;
    }
    update leads; }
    global void finish(Database.BatchableContext bc){
    System.debug(recordsProcessed + ' records processed. Shazam!');
    }}

```

2.LeadProcessorTest.apxc

```

@isTest
public class LeadProcessorTest {
@testSetup
static void setup() {
    List<Lead> leads = new List<Lead>(); // insert 200 leads
    for (Integer i=0;i<200;i++) {
    leads.add(new Lead(LastName='Lead '+i, Company='Lead', Status='Open -
    Not Contacted'));
    }
    insert leads; }
static testmethod void test() {
    Test.startTest();
    LeadProcessor lp = new LeadProcessor();
    Id batchId = Database.executeBatch(lp, 200); Test.stopTest();
    // after the testing stops, assert records were updated properly
    System.assertEquals(200, [select count() from lead where LeadSource =
    'Dreamforce']); }
}

```

•CONTROL PROCESSES WITH QUEUEABLE APEX:

1.AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable { private Contact
contactObj;
private String state_code;
public AddPrimaryContact(Contact c, String s) {
this.contactObj = c;
this.state_code = s; }
public void execute(QueueableContext context) { List<Account> accounts =
[SELECT Id
FROM Account
WHERE BillingState = :this.state_code LIMIT 200];
List<Contact> contacts = new List<Contact>(); for (Account a : accounts) {
Contact c = this.contactObj.clone(false, false, false, false); c.AccountId =
a.Id;
contacts.add(c);
}
if (contacts.size() > 0) {
insert contacts; }
}}
```

2.AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
@testSetup
static void setup(){
List<Account> lstOfAcc = new List<Account>(); for(Integer i = 1; i <= 100;
i++){
if(i <= 50)
lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
else
lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
}
```

```

INSERT lstoOfAcc; }
static testmethod void testAddPrimaryContact(){
Contact con = new Contact(LastName = 'TestCont'); AddPrimaryContact
addPCIns = new AddPrimaryContact(CON,'CA'); Test.startTest();
System.enqueueJob(addPCIns);
Test.stopTest();
System.assertEquals(50, [select count() from Contact]);
}}

```

•SCHEDULE JOBS USING APEX SCHEDULER:

1.DailyLeadProcessor.apxc

```

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;
}}
}

```

2.DailyLeadProcessorTest.apxc

```

@isTest
private class DailyLeadProcessorTest {
static testMethod void testDailyLeadProcessor() { String CRON_EXP = '0 0
1 * * ?';
List<Lead> IList = new List<Lead>();
for (Integer i = 0; i < 200; i++) {
IList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));
}
insert IList;
Test.startTest();
String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());

```

```
}  
}
```

APEX INTEGRATION SERVICES

•APEX REST CALLOUTS:

1. **AnimalLocator.apxc**

```
public class AnimalLocator {  
    public static String getAnimalNameById(Integer animalId) {  
        String animalName;  
        Http http = new Http();  
        HttpRequest request = new HttpRequest(); request.setEndpoint('https://th-  
apex-  
httpcallout.herokuapp.com/animals/'+animalId); request.setMethod('GET');  
        HttpResponse response = http.send(request);  
        // If the request is successful, parse the JSON response.  
        if(response.getStatusCode() == 200) {  
            Map<String, Object> r = (Map<String, Object>)  
JSON.deserializeUntyped(response.getBody());  
            Map<String, Object> animal = (Map<String, Object>)r.get('animal');  
            animalName = string.valueOf(animal.get('name')); }  
        return animalName; }  
}
```

2. **AnimalLocatorMock.apxc**

```
@isTest  
global class AnimalLocatorMock implements HttpCalloutMock {  
    global HTTPResponse respond(HTTPRequest request) { HttpResponse  
response = new HttpResponse(); response.setHeader('Content-Type',  
'application/json');  
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken  
food","says":"cluck cluck"}}'); response.setStatusCode(200); return  
response;
```



```
}}
```

3.AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest {
    @isTest static void getAnimalNameById() {
// Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock()); // This
causes a fake response to be sent
// from the class that implements HttpCalloutMock.
    String response = AnimalLocator.getAnimalNameById(1);
// Verify that the response received contains fake values
    System.assertEquals('chicken', response);
    }
}
```

•APEX SOAP CALLOUTS:

1.ParkLocator.apxc

```
public class ParkLocator {
    public static String [] country (String x) {
    String parks = x; // {'Yellowstone','Kanha','Mount Fuji'};
    ParkService.ParksImplPort findCountries = new ParkService.ParksImplPort
();
    return findCountries.byCountry (parks); }
}
```

2.ParkLocatorTest.apxc

```
@isTest
public class ParkLocatorTest {
    @isTest static void testCallout () {
// This causes a fake response to be generated
    Test.setMock (WebServiceMock.class, new ParkServiceMock ()); String x
='Yellowstone';
    List <String> result = ParkLocator.country(x);
```

```

string resultstring = string.join (result,',');
System.assertEquals ('USA', resultstring);
}}

```

3.ParkServiceMock

```

@Test
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
();
response_x.return_x = new List <String> {'USA'}; response.put ('response_x',
response_x);
}}

```

•APEX WEB SERVICES:

1.AccountManager.apxc

```

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

```

```
return acc; }  
}
```

2.AccountManagerTest.apxc

```
@IsTest  
private class AccountManagerTest{  
@isTest static void testAccountManager(){ Id recordId =  
getTestAccountId();  
// Set up a test request  
RestRequest request = new RestRequest(); request.requestUri =  
'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId  
+'/contacts';  
request.httpMethod = 'GET'; RestContext.request = request;  
// Call the method to test  
Account acc = AccountManager.getAccount(); // Verify results  
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 SPECIALIST SUPERBADGE

•AUTOMATE RECORD CREATION:

1.MaintenanceRequest.apxt

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

```
MaintenanceRequestHelper.updateWorkOrders(Trieger.New,
Trigger.OldMap);
}}
```

2.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, 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>();
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<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; }
}}
```

•SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

1.WarehouseCalloutService.apxc

```

public with sharing class WarehouseCalloutService implements Queueable
{
    private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.herokuapp.com/equipment';
    //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(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint(WAREHOUSE_URL); request.setMethod('GET');
    HttpResponse response = http.send(request); List<Product2>
    warehouseEq = new List<Product2>(); if (response.getStatusCode() ==
    200){
        List<Object> jsonResponse =
        (List<Object>)JSON.deserializeUntyped(response.getBody());
        System.debug(response.getBody());
        //class maps the following fields: replacement part (always true), cost,
        current
        inventory, lifespan, maintenance cycle, and warehouse SKU
        //warehouse SKU will be external ID for identifying which equipment
        records to
        update within Salesforce
        for (Object eq : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)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 = (Integer) mapJson.get('cost'); myEq.Warehouse_SKU__c =
            (String) mapJson.get('sku'); myEq.Current_Inventory__c = (Double)
            mapJson.get('quantity'); myEq.ProductCode = (String) mapJson.get('_id');
            warehouseEq.add(myEq);
        }
        if (warehouseEq.size() > 0){
            upsert warehouseEq;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

```

```

}}
}
public static void execute (QueueableContext context){
runWarehouseEquipmentSync(); }
}

```

•SCHEDULE SYNCHRONIZATION USING APEX CODE:

1.WarehouseSyncSchedule.apxc

```

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

```

•TEST AUTOMATION LOGIC:

1.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<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);
}}

```

2.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, 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>();
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<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; }
}

```

3.MaintenanceRequest.apxt

```

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

```

•TEST CALLOUT LOGIC:

1.WarehouseCalloutService.apxc

```

public with sharing class WarehouseCalloutService implements Queueable
{
    private static final String WAREHOUSE_URL = 'https://th-
        superbadgeapex.herokuapp.com/equipment';
    //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(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
    }
}

```

```

request.setEndpoint(WAREHOUSE_URL); request.setMethod('GET');
    HttpResponse response = http.send(request); List<Product2>
warehouseEq = new List<Product2>(); if (response.getStatusCode() ==
200){
    List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
    System.debug(response.getBody());
    //class maps the following fields: replacement part (always true), cost,
    current
    inventory, lifespan, maintenance cycle, and warehouse SKU
    //warehouse SKU will be external ID for identifying which equipment
    records to
    update within Salesforce
    for (Object eq : jsonResponse){
        Map<String,Object> mapJson = (Map<String,Object>)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 = (Integer) mapJson.get('cost'); myEq.Warehouse_SKU__c =
        (String) mapJson.get('sku'); myEq.Current_Inventory__c = (Double)
        mapJson.get('quantity'); myEq.ProductCode = (String) mapJson.get('_id');
        warehouseEq.add(myEq);
    }
    if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse
        one');
    }
    }}

```

```

public static void execute (QueueableContext context){
runWarehouseEquipmentSync();
}}

```

2.WarehouseCalloutServiceTest.apxc

```

@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
// implement mock callout test here
Test.setMock(HTTPCalloutMock.class, new
WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync(); Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}}

```

3.WarehouseCalloutServiceMock.apxc

```

@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock
{
// implement http mock callout
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
System.assertEquals('GET', request.getMethod());
// Create a fake response
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"}');
}
}

```

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

•TEST SCHEDULING LOGIC:

1.WarehouseSyncSchedule.apxc

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

2.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 ');
}}
```