## *APPEX TRIGGERS*

GET STARTED WITH APEX TRIGGERS:

**1.AccountAddressTrigger.apxt**

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:

**1.ClosedOpportunityTrigger.apxt**

trigger ClosedOpportunityTrigger on Opportunity (before 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;

}

}

## *APPEX TESTING*

GET STARTED WITH APEX UNIT TEST:

1. **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

public 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('12/01/2019'));

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:

1. **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');

}

}

}

1. 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());

}

}

CREATE TEST DATA FOR APEX TESTS:

**1.RandomContactFactory.apxc**

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:

1. **AccountProcessor.apxc**

public class AccountProcessor { @future

public static void countContacts(List<Id> accountIds)

{

List<Account> accountsToUpdate=new List<Account>();

List<Account> accounts=[Select Id,Name,(Select Id from Contacts) from Account Where Id in:accountIds];

For(Account acc:accounts){

List<Contact> contactList=acc.Contacts; acc.Number\_Of\_Contacts c=contactList.size(); accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

# AccountProcessorTest.apxc

@IsTest

private class AccountProcessorTest { @IsTest

private static void testCountContacts(){

Account newAccount = new Account(Name='Test 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<Id> accountIds=new List<Id>(); accountIds.add(newAccount.Id);

Test.startTest(); AccountProcessor.countContacts(accountIds); Test.stopTest()

}

}

USE BATCH APEX:

1. **LeadProcessor.apxc**

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);

}

}

# LeadProcessorTest.apxc

@isTest

public class LeadProcessorTest { @isTest

public static void testit(){

List<lead> L\_list = new List<Lead>();

for(Integer i=0;i<200;i++){ Lead L=new Lead(); L.LastName= 'name'+i; L.Company='Company'; L.Status='Random Status'; L\_list.add(L);

}

insert L\_list;

Test.startTest();

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

}

}

CONTROL PROCESSES WITH QUEUEABLE APEX:

1. **AddPrimaryContact.apxc**

public class AddPrimaryContact implements Queueable { private Contact con;

private String state;

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

this.state=state;

}

public void execute(QueueableContext context){

List<Account> accounts=[select Id,Name,(Select FirstName,LastName,Id from contacts) from Account where BillingState= :state Limit 200];

List<Contact> primaryContacts= new List<Contact>();

for(Account acc:accounts){ Contact c=con.clone(); c.AccountId=acc.Id; primaryContacts.add(c);

}

if(primaryContacts.size() > 0){ insert primaryContacts;

}

}

}

# AddPrimaryContactTest.apxc

@isTest

public class AddPrimaryContactTest {

static testmethod void testQueueable(){

List<Account> testAccounts=new List<Account>(); for(Integer i=0;i<50;i++)

{

testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));

}

for(Integer j=0;j<50;j++)

{

testAccounts.add(new Account(Name='Account' +j,BillingState='NY'));

}

insert testAccounts;

Contact testContact=new Contact(FirstName='john',LastName='Doe'); insert testContact;

AddPrimaryContact addit=new AddPrimaryContact(testContact,'CA');

Test.startTest();

system.enqueueJob(addit); Test.stopTest();

System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from Account where BillingState='CA')]);

}

}

SCHEDULE JOBS USING APEX SCHEDULER:

1. **DailyLeadProcessor.apxc**

public without sharing class DailyLeadProcessor implements schedulable{ public void execute(SchedulableContext ctx)

{

List<lead> leads=[SELECT Id,LeadSource FROM Lead WHERE Leadsource = null LIMIT 200]; for(Lead l: leads)

{

l.LeadSource='Dreamforce';

}

update leads;

}

}

# DailyLeadProcessorTest.apxc

@isTest

public class DailyLeadProcessorTest{

private static String CRON\_EXP='0 0 0 ? \* \* \*';

@isTest

private static void testschedulabelClass(){ List<Lead> leads=new List<Lead>(); for(Integer i=0;i<500;i++){

if(i<250){

leads.add(new Lead(LastName='connock',Company='Salesforce'));

}

else{

leads.add(new Lead(LastName='Connock',Company='Salesforce',LeadSource='Other'));

}

}

insert leads;

Test.startTest();

String jobId=System.schedule('Process Leads',CRON\_EXP,new DailyLeadProcessor()); Test.stopTest();

List<lead> updatedLeads=[select Id,LeadSource from Lead where LeadSource='Dreamforce'];

System.assertEquals(200,updatedLeads.size(),'ERROR: at least 1 record not updated correctly');

List<CronTrigger> cts=[select Id, TimesTriggered ,NextFireTime from CronTrigger where Id=

:jobId];

System.debug('Next Fire Time '+cts[0].NextFireTime);

}

}

**APEX INTEGRATION SERVICES**

APEX REST CALLOUTS:

1. **AnimalLocator.apxc**

public class AnimalLocator {

public static String getAnimalNameById (Integer i) { Http http=new Http();

HttpRequest request=new HttpRequest();

request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i); request.setMethod('GET');

HttpResponse response=http.send(request);

Map<String,Object> result=(Map<String,Object>)JSON.deserializeUntyped(response.getBody());

Map<String,Object> animal=(Map<String,Object>)result.get('animal'); System.debug('name: '+string.valueOf(animal.get('name')));

return string.valueOf(animal.get('name'));

}

}

# AnimalLocatorMock.apxc

@isTest

global class AnimalLocatorMock implements HttpCalloutMock{ global HttpResponse respond(HttpRequest request){

HttpResponse response=new HttpResponse(); response.setHeader('contentType','application/jason'); response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"}}'); response.setStatusCode(200);

return response;

}

}

# AnimalLocatorTest.apxc

@isTest

private class AnimalLocatorTest{ @isTest

static void animalLocatorTest1(){ Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock()); String actual=AnimalLocator.getAnimalNameById(1);

String expected='moose';

System.assertEquals(actual, expected);

}

}

APEX SOAP CALLOUTS:

1. **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'};](http://parks.services/%27%2Cnull%2C%270%27%2C%27-1%27%2C%27false%27)

private String[] apex\_schema\_type\_info = new String[][{'h](http://parks.services/%27%2C%27false%27%2C%27false%27)tt[p://parks.services/','false','false'}](http://parks.services/%27%2C%27false%27%2C%27false%27); 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'};](http://parks.services/%27%2Cnull%2C%270%27%2C%271%27%2C%27false%27) private String[] apex\_schema\_type\_info = new String[][{'h](http://parks.services/%27%2C%27false%27%2C%27false%27)tt[p://parks.services/','false','false'}](http://parks.services/%27%2C%27false%27%2C%27false%27); private String[] field\_order\_type\_info = new String[]{'arg0'};

}

public class ParksImplPort {

public String endpoint\_x = ['https://th-apex-soap-service.herokuapp.com/service/parks](https://th-apex-soap-service.herokuapp.com/service/parks)'; public Map<String,String> inputHttpHeaders\_x;

public Map<String,String> outputHttpHeaders\_x; public String clientCertName\_x;

public String clientCert\_x;

public String clientCertPasswd\_x; public Integer timeout\_x;

private String[] ns\_map\_type\_info = new String[][{'h](http://parks.services/%27)tt[p://parks.services/',](http://parks.services/%27) 'ParkService'}; public String[] byCountry(String arg0) {

ParkService.byCountry request\_x = new ParkService.byCountry(); request\_x.arg0 = arg0;

ParkService.byCountryResponse response\_x;

Map<String, ParkService.byCountryResponse> response\_map\_x = new Map<String, ParkService.byCountryResponse>();

response\_map\_x.put('response\_x', response\_x); WebServiceCallout.invoke(

this, request\_x,

response\_map\_x,

new String[]{endpoint\_x, '',

['http://parks.services/](http://parks.services/)', 'byCountry', ['http://parks.services/](http://parks.services/)', 'byCountryResponse', 'ParkService.byCountryResponse'}

);

response\_x = response\_map\_x.get('response\_x'); return response\_x.return\_x;

}

}

}

# 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'};](http://parks.services/%27%2Cnull%2C%270%27%2C%27-1%27%2C%27false%27)

private String[] apex\_schema\_type\_info = new String[][{'h](http://parks.services/%27%2C%27false%27%2C%27false%27)tt[p://parks.services/','false','false'}](http://parks.services/%27%2C%27false%27%2C%27false%27); 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'};](http://parks.services/%27%2Cnull%2C%270%27%2C%271%27%2C%27false%27) private String[] apex\_schema\_type\_info = new String[][{'h](http://parks.services/%27%2C%27false%27%2C%27false%27)tt[p://parks.services/','false','false'}](http://parks.services/%27%2C%27false%27%2C%27false%27); private String[] field\_order\_type\_info = new String[]{'arg0'};

}

public class ParksImplPort {

public String endpoint\_x = ['https://th-apex-soap-service.herokuapp.com/service/parks](https://th-apex-soap-service.herokuapp.com/service/parks)'; public Map<String,String> inputHttpHeaders\_x;

public Map<String,String> outputHttpHeaders\_x; public String clientCertName\_x;

public String clientCert\_x;

public String clientCertPasswd\_x; public Integer timeout\_x;

private String[] ns\_map\_type\_info = new String[][{'h](http://parks.services/%27)tt[p://parks.services/',](http://parks.services/%27) 'ParkService'}; public String[] byCountry(String arg0) {

ParkService.byCountry request\_x = new ParkService.byCountry(); request\_x.arg0 = arg0;

ParkService.byCountryResponse response\_x;

Map<String, ParkService.byCountryResponse> response\_map\_x = new Map<String, ParkService.byCountryResponse>();

response\_map\_x.put('response\_x', response\_x); WebServiceCallout.invoke(

this, request\_x,

response\_map\_x,

new String[]{endpoint\_x, '',

['http://parks.services/](http://parks.services/)', 'byCountry', ['http://parks.services/](http://parks.services/)', 'byCountryResponse',

'ParkService.byCountryResponse'}

);

response\_x = response\_map\_x.get('response\_x'); return response\_x.return\_x;

}

}

}

# ParkLocatorTest.apxc

@isTest

public class ParkLocatorTest { @isTest static void testCallout(){

Test.setMock(WebServiceMock.class, new ParkServiceMock()); String country='United States';

List<String> expectedParks=new List<String>{'Yosemite','Sequoia','Crater Lake'}; System.assertEquals(expectedParks,ParkLocator.country(country));

}

}

1. ParkServiceMock.apxc

@isTest

global class ParkServiceMock implements webServiceMock{

global void doInvoke( Object stub, Object request,

Map<String,Object> response, String endpoint,

String soapAction, String requestName, String responseNS, String responseName, String responseType)

{

parkService.byCountryResponse response\_x=new parkService.byCountryResponse(); response\_x.return\_x=new List<String>{'Yosemite','Sequoia','Crater Lake'}; 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 request=RestContext.request;

String accountId=request.requestURI.substringBetween('Accounts/','/contacts'); Account result=[select ID,Name,(select ID,FirstName,LastName from Contacts)

from Account

where Id= :accountId];

return result;

}

}

# AccountManagerTest.apxc

@isTest

private class AccountManagerTest { @isTest

static void testGetAccount(){

Account a=new Account(Name='TestAccount'); insert a;

Contact c=new Contact(AccountId=a.Id, FirstName='Test',LastName='Test'); insert c;

RestRequest request=new RestRequest();

request.requestUri='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/contacts'; request.httpMethod='GET';

RestContext.request=request;

Account myAcct=AccountManager.getAccount(); System.assert(myAcct!=null); System.assertEquals('TestAccount', myAcct.Name);

}

}

## *APEX SPECIALIST*

AUTOMATE RECORD CREATION:

1. **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<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;

}

}

}

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-superbadge-apex.herokuapp.com/equipment](https://th-superbadge-apex.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 with sharing 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 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);

}

}

# 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));

}

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;

}

}

}

# MaintenanceRequest.apxt

trigger MaintenanceRequest on Case (before update, after update) { if(Trigger.isUpdate && Trigger.isAfter){

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

}

}

TEST CALLOUT LOGIC:

1. **WarehouseCalloutService.apxc**

public with sharing class WarehouseCalloutService { private static final String WAREHOUSE\_URL =

['https://th-superbadge-apex.herokuapp.com/equipment](https://th-superbadge-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<Product2> warehouseEq = new List<Product2>();

if (response.getStatusCode() == 200){ List<Object> jsonResponse =

(List<Object>)JSON.deserializeUntyped(response.getBody()); System.debug(response.getBody());

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 = (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 { @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]);

}

}

# 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":"Ge nerator 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) {

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 ');

}

}