**Trailhead-Salesforce Developer Catalyst:**

**Apex Triggers:**

1. [**Get Started with Apex Triggers**](https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_intro?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst) **:**

**AccountAddressTrigger:**

trigger AccountAddressTrigger on Account (before insert , before update) {

for (Account a : Trigger.new){

if(a.Match\_Billing\_Address\_\_c == true){

a.ShippingPostalCode = a.BillingPostalCode;

}

}

}

1. [**Bulk Apex Triggers**](https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_bulk?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst) **:**

**ClosedOpportunityTrigger:**

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

list<Task> newTask= new list<Task>();

for(Opportunity oppWon :[Select Id from Opportunity where StageName='Closed Won'

and Id in: Trigger.new]){

newTask.add(new Task (Subject ='Follow Up Test Task',WhatId=oppWon.Id));

}

if(newTask.size()>0){

upsert newTask;

}

}

# 

# **Apex Testing:**

1. [**Get Started with Apex Unit Tests**](https://trailhead.salesforce.com/content/learn/modules/apex_testing/apex_testing_intro?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**VerifyDate:**

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

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

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:**

@istest

public class TestVerifyDate {

@istest Static Void test1(){

Date d = Verifydate.Checkdates(date.parse('01/01/2022'),date.parse('01/03/2022'));

System.assertEquals(date.parse('01/03/2022'),d);

}

@istest Static Void test2(){

Date d = Verifydate.Checkdates(date.parse('01/01/2022'),date.parse('03/03/2022'));

System.assertEquals(date.parse('01/31/2022'),d);

}

}

1. **Test Apex Triggers:**

**RestrictContactByName:**

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:**

@isTest

public class TestRestrictContactByName {

@isTest

public static void testcontact(){

contact ct = new contact();

ct.LastName = 'INVALIDNAME';

database.Saveresult res = Database.insert(ct,false);

System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',res.getErrors()[0].getMessage());

}

}

1. **Create Test Data for Apex Tests:**

**RandomContactFactory:**

public class RandomContactFactory {

Public static List<contact> generateRandomContacts (integer num, string lastName){

List<Contact> contactlist = new list<contact>();

for(integer i=1;i<=num;i++){

contact ct = new contact(FirstName = 'Test' + i,LastName= lastName);

contactlist.add(ct);

}

return contactlist;

}

}

[**Asynchronous Apex**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

1. [**Use Future Methods**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_future_methods?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AccountProcessor:**

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;

}

}

**AccountProcessorTest:**

@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();

}

}

b) [**Use Batch Apex**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_batch?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**LeadProcessor:**

global class LeadProcessor implements Database.Batchable<sObject> {

global Integer count = 0;

global Database.QueryLocator start(Database.BatchableContext bc){

return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');

}

global void execute (Database.BatchableContext bc, List<lead> L\_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:**

@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();

}

}

c) [**Control Processes with Queueable Apex**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_queueable?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**AddPrimaryContact:**

public class AddPrimaryContact implements Queueable {

public contact c;

public String state;

public AddPrimaryContact(Contact c, String state) {

this.c = c;

this.state = state;

}

public void execute(QueueableContext qc) {

system.debug('this.c = '+this.c+' this.state = '+this.state);

List<Account> accList = new List<account>([select id, name, BillingState from account where account.BillingState = :this.state limit 200]);

List<contact> insertContact = new List<contact>();

for(account a: accList) {

contact c = new contact();

c = this.c.clone(false, false, false, false);

c.AccountId = a.Id;

insertContact.add(c);

}

insert insertContact;

}

}

**AddPrimaryContactTest:**

@isTest

public class AddPrimaryContactTest {

@testSetup

static void setup() {

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

for(integer i=0; i<=100; i++) {

if(i <=50) {

insertAccount.add(new Account(Name='Acc'+i, BillingState = 'NY'));

} else {

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

}

}

insert insertAccount;

}

static testMethod void testAddPrimaryContact() {

Contact con = new Contact(LastName = 'LastName');

AddPrimaryContact addPC = new AddPrimaryContact(con, 'CA');

Test.startTest();

system.enqueueJob(addPC);

Test.stopTest();

system.assertEquals(50, [select count() from Contact]);

}

}

1. [**Schedule Jobs Using the Apex Scheduler**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_scheduled?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**DailyLeadProcessor:**

global class DailyLeadProcessor implements Schedulable {

global void execute(SchedulableContext ctx) {

//Retrieving the 200 first leads where lead source is in blank.

List<Lead> 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:**

@isTest

private class DailyLeadProcessorTest {

@isTest

public static void testDailyLeadProcessor(){

//Creating new 200 Leads and inserting them.

List<Lead> leads = new List<Lead>();

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<Lead> 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 Integration Services:**

1. [**Apex REST Callouts**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_rest_callouts?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AnimalLocator:**

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');

}

}

**AnimalLocatorTest:**

@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 );

}

}

**AnimalLocatorMock:**

@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;

}

}

1. [**Apex SOAP Callouts**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_soap_callouts?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**ParkLocator:**

public class ParkLocator {

public static string[] country(string theCountry) {

ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space

return parkSvc.byCountry(theCountry);

}

}

**ParkLocatorTest:**

@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);

}

}

**ParkServiceMock:**

@isTest

global class ParkServiceMock implements WebServiceMock {

global void doInvoke(

Object stub,

Object request,

Map<String, Object> response,

String endpoint,

String soapAction,

String requestName,

String responseNS,

String responseName,

String responseType) {

// 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);

}

}

1. [**Apex Web Services**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_webservices?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst):

**AccountManager:**

@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:**

@isTest

private class AccountManagerTest {

private static testMethod void getAccountTest1() {

Id recordId = createTestRecord();

// Set up a test request

RestRequest request = new RestRequest();

request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts' ;

request.httpMethod = 'GET';

RestContext.request = request;

// Call the method to test

Account thisAccount = AccountManager.getAccount();

// Verify results

System.assert(thisAccount != null);

System.assertEquals('Test record', thisAccount.Name);

}

// Helper method

static Id createTestRecord() {

// Create test record

Account TestAcc = new Account(

Name='Test record');

insert TestAcc;

Contact TestCon= new Contact(

LastName='Test',

AccountId = TestAcc.id);

return TestAcc.Id;

}

}

# **Apex Specialist Superbadge:**

1. **Automated Record Creation**

**MaintenanceRequestHelper.apxc**

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

Set<Id> validIds = new Set<Id>();

For (Case c : updWorkOrders){

if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

validIds.add(c.Id);

}

}

}

if (!validIds.isEmpty()){

List<Case> newCases = new List<Case>();

Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle\_\_c, Equipment\_\_c, Equipment\_\_r.Maintenance\_Cycle\_\_c,(SELECT Id,Equipment\_\_c,Quantity\_\_c FROM Equipment\_Maintenance\_Items\_\_r)

FROM Case WHERE Id IN :validIds]);

Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

AggregateResult[] results = [SELECT Maintenance\_Request\_\_c, MIN(Equipment\_\_r.Maintenance\_Cycle\_\_c)cycle FROM Equipment\_Maintenance\_Item\_\_c WHERE Maintenance\_Request\_\_c IN :ValidIds GROUP BY Maintenance\_Request\_\_c];

for (AggregateResult ar : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));

}

for(Case cc : closedCasesM.values()){

Case nc = new Case (

ParentId = cc.Id,

Status = 'New',

Subject = 'Routine Maintenance',

Type = 'Routine Maintenance',

Vehicle\_\_c = cc.Vehicle\_\_c,

Equipment\_\_c =cc.Equipment\_\_c,

Origin = 'Web',

Date\_Reported\_\_c = Date.Today()

);

If (maintenanceCycles.containskey(cc.Id)){

nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

} else {

nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);

}

newCases.add(nc);

}

insert newCases;

List<Equipment\_Maintenance\_Item\_\_c> clonedWPs = new List<Equipment\_Maintenance\_Item\_\_c>();

for (Case nc : newCases){

for (Equipment\_Maintenance\_Item\_\_c wp : closedCasesM.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){

Equipment\_Maintenance\_Item\_\_c wpClone = wp.clone();

wpClone.Maintenance\_Request\_\_c = nc.Id;

ClonedWPs.add(wpClone);

}

}

insert ClonedWPs;

}

}

}

**MaitenanceRequest.apxt**

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

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

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

}

}

**2. Synchronize Salesforce data with an external system**

**WarehouseCalloutService.apxc :-**

public with sharing class WarehouseCalloutService implements Queueable {

private static final String WAREHOUSE\_URL = '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();

}

}

**3.Schedule synchronization using Apex code**

**WarehouseSyncShedule.apxc :-**

global with sharing class WarehouseSyncSchedule implements Schedulable{

global void execute(SchedulableContext ctx){

System.enqueueJob(new WarehouseCalloutService());

}

}

**4.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<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);

}

}

**5.Test callout logic**

**WarehouseCalloutService.apxc :-**

public with sharing class WarehouseCalloutService {

private static final String WAREHOUSE\_URL = '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":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');

response.setStatusCode(200);

return response;

}

}

**6.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 ');

}

}