**1. Get Started with Apex Triggers**

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

    for(Account account:Trigger.New){

        if(account.Match\_Billing\_Address\_\_c == True){

            account.ShippingPostalcode = account.BillingPostalCode;

        }

    }

}

**Create a Bulk Apex trigger**

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

    List<task> tasklist =new List<Task>();

    for(Opportunity opp: Trigger.New){

        if(opp.StageName == 'Closed Won'){

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

        }

    }

    if(tasklist.size()>0){

        insert tasklist;

    }

}

2. **Get Started with Apex Unit Tests**

**public class VerifyDate {**

**//method to handle potential checks against two dates**

**public static Date CheckDates(Date date1, Date date2) {**

**//if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month**

**if(DateWithin30Days(date1,date2)) {**

**return date2;**

**} else {**

**return SetEndOfMonthDate(date1);**

**}**

**}**

**//method to check if date2 is within the next 30 days of date1**

**@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {**

**//check for date2 being in the past**

**if( date2 < date1) { return false; }**

**//check that date2 is within (>=) 30 days of date1**

**Date date30Days = date1.addDays(30); //create a date 30 days away from date1**

**if( date2 >= date30Days ) { return false; }**

**else { return true; }**

**}**

**//method to return the end of the month of a given date**

**@TestVisible private static Date SetEndOfMonthDate(Date date1) {**

**Integer totalDays = Date.daysInMonth(date1.year(), date1.month());**

**Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);**

**return lastDay;**

**}**

**}**

**TestVerifyDate**

**@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 a Unit Test for a Simple Apex Trigger**

**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');**

**}**

**}**

**}**

**Create a Contact Test Factory**

**public class RandomContactFactory {**

**public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){**

**List<Contact> contacts = new List<Contact>();**

**for(Integer i=0;i<numcnt;i++){**

**Contact cnt = new Contact(FirstName = 'Test' +i, Lastname = lastname);**

**contacts.add(cnt);**

**}**

**return contacts;**

**}**

**}**

**Asynchronous apex**

**Use future Methods**

**public class AccountProcessor {**

**@future**

**public static void countContacts(List<Id> accountIds){**

**List<Account> 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;**

**}**

**}**

**@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:**

**global class LeadProcessor implements Database.Batchable<sObject> {**

**global Integer count=0;**

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

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

**}**

**global void execute (Database.BatchableContext bc, List<Lead> L\_list){**

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

**for(lead L:L\_list){**

**L.leadsource = 'Dreamforce';**

**L\_list\_new.add(L);**

**count+=1;**

**}**

**update L\_list\_new;**

**}**

**global void finish(Database.BatchableContext bc){**

**system.debug('count=' + count);**

**}**

**}**

**@isTest**

**public class LeadProcessorTest {**

**@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 PROCESS WITH QUEUABLE APEX:**

**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;**

**}**

**}**

**}**

**@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:**

**global class DailyLeadProcessor implements Schedulable{**

**global void execute(SchedulableContext ctcx){**

**List<Lead> leadstoupdate = new List<lead>();**

**List<Lead> leads = [Select id From Lead Where Leadsource = NULL Limit 200];**

**for(Lead l:leads){**

**l.LeadSource = 'Dreamforce';**

**leadstoupdate.add(l);**

**}**

**update leadstoupdate;**

**}**

**}**

**@isTest**

**private class DailyLeadProcessorTest {**

**public static String CRON\_EXP = '0 0 0 15 4 ? 2023';**

**static testmethod void testScheduledJob(){**

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

**for(Integer i=0;i<200;i++){**

**Lead l = new Lead(**

**FirstName = 'First ' +i,**

**LastName = 'LastName',**

**Company = 'The Inc'**

**);**

**leads.add(l);**

**}**

**insert leads;**

**Test.startTest();**

**String jobId = System.schedule('ScheduledApexTest',CRON\_EXP,new DailyLeadProcessor());**

**Test.stopTest();**

**List<Lead> checkleads = new List<Lead>();**

**checkleads = [Select Id From Lead Where LeadSource = 'Dreamforce' and Company = 'The Inc'];**

**System.assertEquals(200,checkleads.size(), 'Leads were not created');**

**}**

**}**

**APEX INTEGRATION SERVICES**

**APEX REST CALLOUTS:**

public class AnimalLocator{

    public static String getAnimalNameById(Integer x){

        Http http = new Http();

        HttpRequest req = new HttpRequest();

        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);

        req.setMethod('GET');

        Map<String, Object> animal= new Map<String, Object>();

        HttpResponse res = http.send(req);

            if (res.getStatusCode() == 200) {

        Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());

      animal = (Map<String, Object>) results.get('animal');

        }

return (String)animal.get('name');

    }

}

@isTest

private class AnimalLocatorTest{

    @isTest static void AnimalLocatorMock1() {

        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

        string result = AnimalLocator.getAnimalNameById(3);

        String expectedResult = 'chicken';

        System.assertEquals(result,expectedResult );

    }

}

@isTest

global class AnimalLocatorMock implements HttpCalloutMock

{

     // Implement this interface method

    global HTTPResponse respond(HTTPRequest request)

    {

        // Create a fake response

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type', 'application/json');

        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');

        response.setStatusCode(200);

        return response;

    }

}

APEX SOAP CALLOUTS

public class ParkLocator {

    public static string[] country(string theCountry) {

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

        return parkSvc.byCountry(theCountry);

    }

}

@isTest

private class ParkLocatorTest {

    @isTest static void testCallout() {

        Test.setMock(WebServiceMock.class, new ParkServiceMock ());

        String country = 'United States';

        List<String> result = ParkLocator.country(country);

        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

         System.assertEquals(parks, result);

    }

}

@isTest

global class ParkServiceMock implements WebServiceMock {

   global void doInvoke(

           Object stub,

           Object request,

           Map<String, Object> response,

           String endpoint,

           String soapAction,

           String requestName,

           String responseNS,

           String responseName,

           String responseType) {

        // start - specify the response you want to send

        ParkService.byCountryResponse response\_x = new ParkService.byCountryResponse();

        response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

        // end

        response.put('response\_x', response\_x);

   }

}

APEX WEB SERVICES

@RestResource(urlMapping='/Accounts/\*/contacts')

global 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;

    }

}

@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

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;

        }

    }

}

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

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

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

    }

}

CHALLENGE 2

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();

    }

}

CHALLENGE 3

global with sharing class WarehouseSyncSchedule implements Schedulable{

    global void execute(SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

CHALLENG 4

@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);

    }

}

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;

        }

    }

}

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

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

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

    }

}

CHALLENGE-5

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);

            }

        }

    }

}

@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]);

    }

}

@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;

    }

}

CHALLENGE-6

global class WarehouseSyncSchedule implements Schedulable {

    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();

    }

}

@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 ');

    }

}