

MaintenanceRequestHelper - class

```
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; }
}}
```

MaintenanceRequestHelperTest - class

```

@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); }
}

```

WarehouseCalloutService - class

```

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);
}
}}
}

```

WarehouseCalloutServiceMock - class

```

@Test
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; }
}

```

WarehouseCalloutServiceTest - class

```

@Test
private class WarehouseCalloutServiceTest { @Test
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]); }
}

```

WarehouseSyncSchedule - class

```

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

```

WarehouseSyncScheduleTest - class

```

@isTest
public class WarehouseSyncScheduleTest {
    @isTest static void testScheduler() {
        Test.SetMock(HttpCallOutMock.class, new WarehouseCalloutServiceMock());
        String CRON_EXP = '0 0 0 1 1/1 ? *'; // To be executed monthly at day one Integer runDate = 1;
        DateTime firstRunTime = System.now(); DateTime nextDateTime;
        if(firstRunTime.day() < runDate) {
            nextDateTime = firstRunTime;
        } else {
            nextDateTime = firstRunTime.addMonths(1); }
        Datetime nextRunTime = Datetime.newInstance(nextDateTime.year(), nextDateTime.month(),
        runDate);
        Test.startTest();
        WarehouseSyncSchedule warehouseSyncSchedule = new WarehouseSyncSchedule();
        String jobId = System.schedule('Test Scheduler', CRON_EXP,
        warehouseSyncSchedule);
        Test.stopTest();
        // Get the information from the CronTrigger API object
        CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime FROM CronTrigger
        WHERE Id = :jobId];
        // Verify the expressions are the same
        System.assertEquals(CRON_EXP, ct.CronExpression);
        // Verify the job has not run System.assertEquals(0, ct.TimesTriggered);
        // Verify the next time the job will run System.assertEquals(String.valueOf(nextRunTime),
        String.valueOf(ct.NextFireTime));
    }
}

```

MaintenanceRequest - Trigger

```

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

```

WarehouseSyncSchedule - class

```

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

```