

Project Doc: 2 Superbadges

Process Automation-Super-badge:

1. Here we have automated leads for the company where according to the validation rules and assignment rules for whom the leads have to be sent have been set
2. Accounts have been automated and a new object named "Robot setup" have been created which checks for the leads and availability according to the validation rules and the flows.
3. A Sales process have been created which checks for the sales and automates the process of sales
4. Opportunities have also been automated which is all monitored by a user "Nushi Davoud" where he is the one who makes the decision.
5. A flow named "Product quick search" have been created which automates products according to different category.
6. A process have been created which sets up the date and describes about the business requirements.
7. Finally all the Automation for taking in leads to changing, creating, sending email alerts and updating records are all have been done.

Apex Specialist Super-Badge:

1. This is a trigger that automates a Maintenance request when any new record is

created or updated.

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

2. This is a Helper class which gets activated by the Request trigger which creates a time slot for the maintenance.

```
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);  
                }  
            }  
        }  
    }  
}
```

//When an existing maintenance request of type Repair or Routine Maintenance is closed,

//create a new maintenance request for a future routine checkup.

```
if (!validIds.isEmpty()) {  
    Map<Id, Case> closedCases = 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>();
```

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

```
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'));
}
```

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

```
for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
//define the due date by applying the shortest maintenance cycle to today's date.
```

```
nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
```

```
newCases.add(nc);
}
```

```
insert newCases;
```

```
List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
```

```

        Equipment_Maintenance_Item__c item = clonedListItem.clone();
        item.Maintenance_Request__c = nc.Id;
        clonedList.add(item);
    }
}
insert clonedList;
}
}
}

```

3.This helper class checks if all the conditions in the maintenance helper class satisfies or not. This helper checks by creating Equipment maintenance item and maintenanceRequest in which 3 test cases positive negative and bulk have been checked.

```

@Test
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
            lifespan_months__c = 10,
            maintenance_cycle__c = 10,
            replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
            Status='New',
            Origin='Web',

```

```

        Subject='Testing subject',
        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cse;
}

// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
    Equipment__c = equipmentId,
    Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}

@isTest
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;

    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
    update createdCase;
    test.stopTest();
}

```

```

Case newCase = [Select id,
                  subject,
                  type,
                  Equipment__c,
                  Date_Reported__c,
                  Vehicle__c,
                  Date_Due__c
                from case
                where status ='New'];

```

```

Equipment_Maintenance_Item__c workPart = [select id
                                           from Equipment_Maintenance_Item__c
                                           where Maintenance_Request__c =:newCase.Id];

```

```

list<case> allCase = [select id from case];
system.assert(allCase.size() == 2);

```

```

system.assert(newCase != null);
system.assert(newCase.Subject != null);
system.assertEquals(newCase.Type, 'Routine Maintenance');
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}

```

@isTest

```

private static void testNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

```

```

product2 equipment = createEquipment();
insert equipment;
id equipmentId = equipment.Id;

```

```

case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
insert createdCase;

```

```

Equipment_Maintenance_Item__c workP =

```

```

createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
    insert workP;

    test.startTest();
    createdCase.Status = 'Working';
    update createdCase;
    test.stopTest();

    list<case> allCase = [select id from case];

    Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                                                from Equipment_Maintenance_Item__c
                                                                where Maintenance_Request__c = :createdCase.Id];

    system.assert(equipmentMaintenanceItem != null);
    system.assert(allCase.size() == 1);
}

@isTest
private static void testBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
    list<case> caseList = new list<case>();
    list<id> oldCaseIds = new list<id>();

    for(integer i = 0; i < 300; i++){
        vehicleList.add(createVehicle());
        equipmentList.add(createEquipment());
    }
    insert vehicleList;
    insert equipmentList;

    for(integer i = 0; i < 300; i++){
        caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
    }
}

```

```

insert caseList;

for(integer i = 0; i < 300; i++){

equipmentMaintenanceltemList.add(createEquipmentMaintenanceltem(equipmentList.
get(i).id, caseList.get(i).id));
}
insert equipmentMaintenanceltemList;

test.startTest();
for(case cs : caseList){
    cs.Status = 'Closed';
    oldCaseIds.add(cs.Id);
}
update caseList;
test.stopTest();

list<case> newCase = [select id
                    from case
                    where status ='New'];

list<Equipment_Maintenance_Item__c> workParts = [select id
                                                from Equipment_Maintenance_Item__c
                                                where Maintenance_Request__c in: oldCaseIds];

system.assert(newCase.size() == 300);

list<case> allCase = [select id from case];
system.assert(allCase.size() == 600);
}
}

```

4. This is a sharing class which implements queueable apex which makes the REST callout form the external system and recieves the responce.


```
public with sharing class WarehouseCalloutService implements Queueable {  
    private static final String WAREHOUSE_URL = 'https://th-superbadge-  
apex.herokuapp.com/equipment';
```

//Write a 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(){  
    System.debug('go into runWarehouseEquipmentSync');  
    Http http = new Http();  
    HttpRequest request = new HttpRequest();  
  
    request.setEndpoint(WAREHOUSE_URL);  
    request.setMethod('GET');  
    HttpResponse response = http.send(request);  
  
    List<Product2> product2List = new List<Product2>();  
    System.debug(response.getStatusCode());  
    if (response.getStatusCode() == 200){  
        List<Object> jsonResponse =  
(List<Object>)JSON.deserializeUntyped(response.getBody());  
        System.debug(response.getBody());  
  
        //class maps the following fields:  
        //warehouse SKU will be external ID for identifying which equipment records to  
update within Salesforce  
        for (Object jR : jsonResponse){  
            Map<String,Object> mapJson = (Map<String,Object>)jR;  
            Product2 product2 = new Product2();  
            //replacement part (always true),  
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');  
            //cost  
            product2.Cost__c = (Integer) mapJson.get('cost');  
            //current inventory
```

```

        product2.Current_Inventory__c = (Double) mapJson.get('quantity');
        //lifespan
        product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
        //maintenance cycle
        product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
        //warehouse SKU
        product2.Warehouse_SKU__c = (String) mapJson.get('sku');
        product2.Name= (String) mapJson.get('name');
        product2.ProductCode = (String) mapJson.get('_id');
        product2List.add(product2);
    }

    if (product2List.size() > 0){
        upsert product2List;
        System.debug('Your equipment was synced with the warehouse one');
    }
}
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}

}

```

5.The callout service mock takes in the response form the external source and returns the Http response

```

@Test
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

        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"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]);
        response.setStatusCode(200);

        return response;
    }
}

```

6.The Test checks if the callout Is working or not by raising the WarehouseCalloutServiceMock and checking the response with the builtin response

```

@Test
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}

```

```

    }
}

```

7. This is a sharing global class where the class is scheduled to be done at particular instance of the day which acts as a raising function for the WarehouseCalloutService in an enqueued process.

```

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

```

8.

This is a schedule test which tests if the Sync schedule has been performed or not by using a test class by setting a time and calling the CalloutService function. Here Job Id has been stored when the function returns it.

```

@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}

```