1- APEX SPECIALIST

What I have done to complete This Superbadge

1. Automate record creation using Apex triggers
2. Synchronize Salesforce data with an external system using asynchronous REST callouts
3. Schedule synchronization using Apex code
4. Test automation logic to confirm Apex trigger side effects
5. Test integration logic using callout mocks
6. Test scheduling logic to confirm action gets queued

1-Automate record creation using Apex triggers

in this I have used two classes

(a)MaintenanceRequest

(b)MaintenanceRequestHelper

```
1  //Maintenance request
2  trigger MaintenanceRequest on Case (before update, after update)
   {
3      if(Trigger.isUpdate && Trigger.isAfter){
4          MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
   Trigger.OldMap);
5      }
6  }
7
```

```
1  //MaintenanceRequestHelper
2  public with sharing class MaintenanceRequestHelper {
3      public static void updateworkOrders(List<Case> updWorkOrders,
   Map<Id,Case> nonUpdCaseMap) {
4          Set<Id> validIds = new Set<Id>();
5          For (Case c : updWorkOrders){
6              if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
   c.Status == 'Closed'){
7                  if (c.Type == 'Repair' || c.Type == 'Routine
```

```
                    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.
                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> 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;
```

```
        }
    }
}
```

2-Synchronize Salesforce data with an external system using asynchronous REST callouts

in this module I have used a class name is:-

(a)WarehouseCalloutService

```
//WarehouseCalloutService
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-


    //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());
```

```
23
24              //class maps the following fields:
25              //warehouse SKU will be external ID for identifying
    which equipment records to update within Salesforce
26              for (Object jR : jsonResponse){
27                  Map<String,Object> mapJson =
    (Map<String,Object>)jR;
28                  Product2 product2 = new Product2();
29                  //replacement part (always true),
30                  product2.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
31                  //cost
32                  product2.Cost__c = (Integer) mapJson.get('cost');
33                  //current inventory
34                  product2.Current_Inventory__c = (Double)
    mapJson.get('quantity');
35                  //lifespan
36                  product2.Lifespan_Months__c = (Integer)
    mapJson.get('lifespan');
37                  //maintenance cycle
38                  product2.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
39                  //warehouse SKU
40                  product2.Warehouse_SKU__c = (String)
    mapJson.get('sku');
41
42                  product2.Name = (String) mapJson.get('name');
43                  product2.ProductCode = (String)
    mapJson.get('_id');
44                  product2List.add(product2);
45              }
46
47              if (product2List.size() > 0){
48                  upsert product2List;
49                  System.debug('Your equipment was synced with the

50              }
51          }
52      }
53
```

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

}
```

3-Schedule synchronization using Apex code

in this I have used a class name is:-

(a)WarehouseSyncSchedule

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

4-Test automation logic to confirm Apex trigger side effects

in this I have used three class follows are:-

(a)MaintenanceRequest

(b)MaintenanceRequestHelper

(c)MaintenanceRequestHelperTest

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

```
1  //MaintenanceRequestHelper
2  public with sharing class MaintenanceRequestHelper {
3      public static void updateworkOrders(List<Case> updWorkOrders,
   Map<Id,Case> nonUpdCaseMap) {
4          Set<Id> validIds = new Set<Id>();
5          For (Case c : updWorkOrders){
6              if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
   c.Status == 'Closed'){
7                  if (c.Type == 'Repair' || c.Type == 'Routine

8                      validIds.add(c.Id);
9                  }
10             }
11         }
12
13         //When an existing maintenance request of type Repair or
   Routine Maintenance is closed,
14         //create a new maintenance request for a future routine
   checkup.
15         if (!validIds.isEmpty()){
16             Map<Id,Case> closedCases = new Map<Id,Case>([SELECT
   Id, Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,
17                                                          (SELECT
   Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
18                                                          FROM
   Case WHERE Id IN :validIds]);
19             Map<Id,Decimal> maintenanceCycles = new
   Map<ID,Decimal>();
20
21             //calculate the maintenance request due dates by
   using the maintenance cycle defined on the related equipment
   records.
22             AggregateResult[] results = [SELECT
   Maintenance_Request__c,
23
   MIN(Equipment__r.Maintenance_Cycle__c)cycle
24                                          FROM
   Equipment_Maintenance_Item__c
25                                          WHERE
   Maintenance_Request__c IN :ValidIds GROUP BY
```

```
    Maintenance_Request__c];
26
27              for (AggregateResult ar : results){
28                  maintenanceCycles.put((Id)
    ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
29              }
30
31              List<Case> newCases = new List<Case>();
32              for(Case cc : closedCases.values()){
33                  Case nc = new Case (
34                      ParentId = cc.Id,
35                      Status = 'New',
36                      Subject = 'Routine Maintenance',
37                      Type = 'Routine Maintenance',
38                      Vehicle__c = cc.Vehicle__c,
39                      Equipment__c =cc.Equipment__c,
40                      Origin = 'Web',
41                      Date_Reported__c = Date.Today()
42                  );
43
44                  //If multiple pieces of equipment are used in the
    maintenance request,
45                  //define the due date by applying the shortest
    maintenance cycle to today's date.
46                  //If (maintenanceCycles.containskey(cc.Id)){
47                      nc.Date_Due__c =
    Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
48                  //} else {
49                  //    nc.Date_Due__c =
    Date.today().addDays((Integer)
    cc.Equipment__r.maintenance_Cycle__c);
50                  //}
51
52                  newCases.add(nc);
53              }
54
55              insert newCases;
56
57              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;
        }
    }
}
```

```
//MaintenanceRequestHelperTest
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing

        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing

                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
```

```
21      private static Case createMaintenanceRequest(id vehicleId, id
    equipmentId){
22          case cse = new case(Type='Repair',
23                              Status='New',
24                              Origin='Web',
25                              Subject='Testing subject',
26                              Equipment__c=equipmentId,
27                              Vehicle__c=vehicleId);
28          return cse;
29      }
30
31      // createEquipmentMaintenanceItem
32      private static Equipment_Maintenance_Item__c
    createEquipmentMaintenanceItem(id equipmentId,id requestId){
33          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    new Equipment_Maintenance_Item__c(
34              Equipment__c = equipmentId,
35              Maintenance_Request__c = requestId);
36          return equipmentMaintenanceItem;
37      }
38
39      @isTest
40      private static void testPositive(){
41          Vehicle__c vehicle = createVehicle();
42          insert vehicle;
43          id vehicleId = vehicle.Id;
44
45          Product2 equipment = createEquipment();
46          insert equipment;
47          id equipmentId = equipment.Id;
48
49          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
50          insert createdCase;
51
52          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
53          insert equipmentMaintenanceItem;
54
55          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;
```

```
93
94          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
95          insert createdCase;
96
97          Equipment_Maintenance_Item__c workP =
    createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
98          insert workP;
99
100          test.startTest();
101          createdCase.Status = 'Working';
102          update createdCase;
103          test.stopTest();
104
105          list<case> allCase = [select id from case];
106
107          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    [select id
108                                                  from
    Equipment_Maintenance_Item__c
109                                                  where
    Maintenance_Request__c = :createdCase.Id];
110
111          system.assert(equipmentMaintenanceItem != null);
112          system.assert(allCase.size() == 1);
113      }
114
115      @isTest
116      private static void testBulk(){
117          list<Vehicle__C> vehicleList = new list<Vehicle__C>();
118          list<Product2> equipmentList = new list<Product2>();
119          list<Equipment_Maintenance_Item__c>
    equipmentMaintenanceItemList = new
    list<Equipment_Maintenance_Item__c>();
120          list<case> caseList = new list<case>();
121          list<id> oldCaseIds = new list<id>();
122
123          for(integer i = 0; i < 300; i++){
124              vehicleList.add(createVehicle());
125              equipmentList.add(createEquipment());
```

```
126         }
127         insert vehicleList;
128         insert equipmentList;
129 
130         for(integer i = 0; i < 300; i++){
131 
    caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
    equipmentList.get(i).id));
132         }
133         insert caseList;
134 
135         for(integer i = 0; i < 300; i++){
136 
    equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(e

137         }
138         insert equipmentMaintenanceItemList;
139 
140         test.startTest();
141         for(case cs : caseList){
142             cs.Status = 'Closed';
143             oldCaseIds.add(cs.Id);
144         }
145         update caseList;
146         test.stopTest();
147 
148         list<case> newCase = [select id
149                                     from case
150                                     where status ='New'];
151 
152 
153 
154         list<Equipment_Maintenance_Item__c> workParts = [select
    id
155                                                          from
    Equipment_Maintenance_Item__c
156                                                          where
    Maintenance_Request__c in: oldCaseIds];
157 
158         system.assert(newCase.size() == 300);
```

```
159
160         list<case> allCase = [select id from case];
161         system.assert(allCase.size() == 600);
162     }
163 }
```

5-Test integration logic using callout mocks

in this I have used three classes that are as follows:-

(a)WarehouseCalloutService

(b)WarehouseCalloutServiceMock

(c)WarehouseCalloutServiceTest

```
1  //WarehouseCalloutService
2  public with sharing class WarehouseCalloutService implements
   Queueable {
3      private static final String WAREHOUSE_URL = 'https://th-

4
5      //Write a class that makes a REST callout to an external
   warehouse system to get a list of equipment that needs to be
   updated.
6      //The callout's JSON response returns the equipment records
   that you upsert in Salesforce.
7
8      @future(callout=true)
9      public static void runWarehouseEquipmentSync(){
10         System.debug('go into runWarehouseEquipmentSync');
11         Http http = new Http();
12         HttpRequest request = new HttpRequest();
13
14         request.setEndpoint(WAREHOUSE_URL);
15         request.setMethod('GET');
16         HttpResponse response = http.send(request);
17
18         List<Product2> product2List = new List<Product2>();
19         System.debug(response.getStatusCode());
```

```
20          if (response.getStatusCode() == 200){
21              List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
22              System.debug(response.getBody());
23
24              //class maps the following fields:
25              //warehouse SKU will be external ID for identifying
    which equipment records to update within Salesforce
26              for (Object jR : jsonResponse){
27                  Map<String,Object> mapJson =
    (Map<String,Object>)jR;
28                  Product2 product2 = new Product2();
29                  //replacement part (always true),
30                  product2.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
31                  //cost
32                  product2.Cost__c = (Integer) mapJson.get('cost');
33                  //current inventory
34                  product2.Current_Inventory__c = (Double)
    mapJson.get('quantity');
35                  //lifespan
36                  product2.Lifespan_Months__c = (Integer)
    mapJson.get('lifespan');
37                  //maintenance cycle
38                  product2.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
39                  //warehouse SKU
40                  product2.Warehouse_SKU__c = (String)
    mapJson.get('sku');
41
42                  product2.Name = (String) mapJson.get('name');
43                  product2.ProductCode = (String)
    mapJson.get('_id');
44                  product2List.add(product2);
45              }
46
47              if (product2List.size() > 0){
48                  upsert product2List;
49                  System.debug('Your equipment was synced with the
```

```
            }
        }
    }

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

}
```

```
//WarehouseCalloutServiceMock
@isTest
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









        response.setStatusCode(200);

        return response;
    }
}
```

```
1  //WarehouseCalloutServiceTest
2  @IsTest
3  private class WarehouseCalloutServiceTest {
4      // implement your mock callout test here
5    @isTest
6      static void testWarehouseCallout() {
7          test.startTest();
8          test.setMock(HttpCalloutMock.class, new
   WarehouseCalloutServiceMock());
9          WarehouseCalloutService.execute(null);
10         test.stopTest();
11
12         List<Product2> product2List = new List<Product2>();
13         product2List = [SELECT ProductCode FROM Product2];
14
15         System.assertEquals(3, product2List.size());
16         System.assertEquals('55d66226726b611100aaf741',
   product2List.get(0).ProductCode);
17         System.assertEquals('55d66226726b611100aaf742',
   product2List.get(1).ProductCode);
18         System.assertEquals('55d66226726b611100aaf743',
   product2List.get(2).ProductCode);
19     }
20 }
```

6-Test scheduling logic to confirm action gets queued

in this module I have used three classes that are as follows

(a)WarehouseCalloutServiceMock

(b)WarehouseSyncSchedule

(c)WarehouseSyncScheduleTest

```
1  // WarehouseCalloutServiceMock
2  @isTest
3  global class WarehouseCalloutServiceMock implements
   HttpCalloutMock {
4      // implement http mock callout
5      global static HttpResponse respond(HttpRequest request) {
6
```

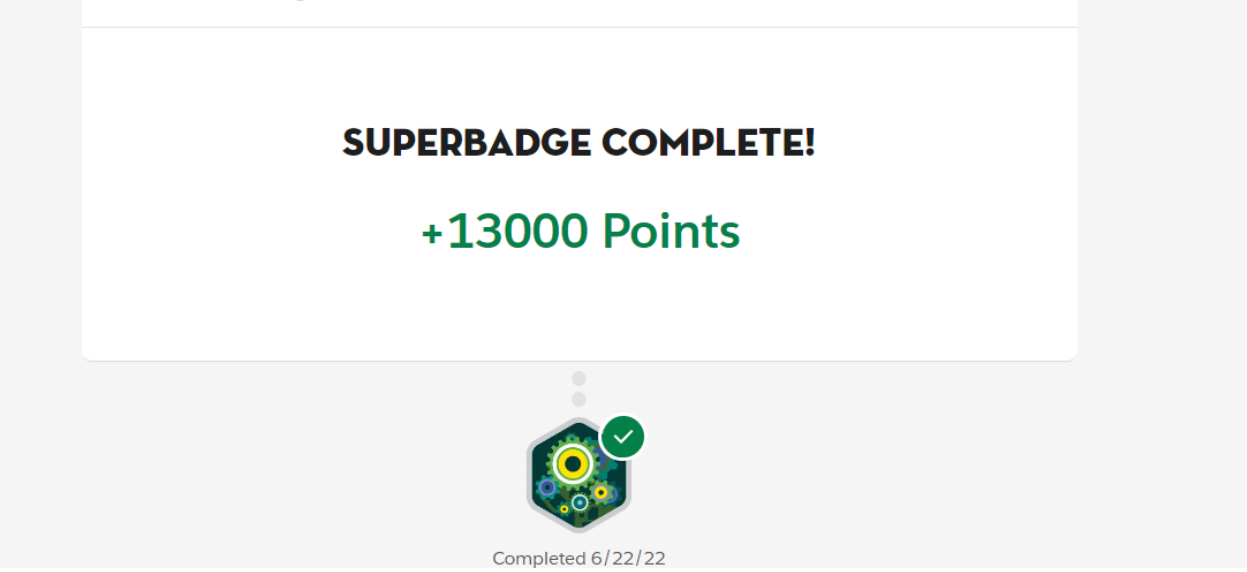
```
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement
        response.setStatusCode(200);

        return response;
    }
}
```

```
//WarehouseSynSchedule
global with sharing class WarehouseSyncSchedule implements
Schedulable {
    // implement scheduled code here
    global void execute (SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

```
//WarehouseSynScheduleTest
@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());
```

```
10          String jobId = System.schedule('Warehouse Time to
    ());
11          CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
    =: jobId];
12          System.assertEquals('WAITING', String.valueOf(c.State),
    'JobId does not match');
13
14          Test.stopTest();
15      }
16  }
```

In this way I have completed this superbadge
[this is my trailhead profile to check the completion status]



2- Process Automation Specialist

What I have Done to Earn This Superbadge

1. Automate lead ownership using assignment rules
2. Enforce data integrity with formula fields and validation rules

3. Create a custom object in a master-detail relationship to a standard object
4. Define an opportunity sales process using stages, record types, and validation rules
5. Automate business processes to send emails, create related records, and submit opportunities for approval
6. Create a flow to display dynamic information on a Lightning record page
7. Create a process to evaluate and update records

[This is my trailhead profile link to check the completion status]

SUPERBADGE COMPLETE!

+10000 Points

Completed 6/23/22