Apex Specialist

2)Automate record creation
MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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.
                //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;
        }
    }
}
```

3)Synchronize Salesforce data with an external system

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());

            //class maps the following fields:
            //warehouse SKU will be external ID for identifying
```

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

49              }
50          }
51      }
52
53      public static void execute (QueueableContext context){
54          System.debug('start runWarehouseEquipmentSync');
55          runWarehouseEquipmentSync();
```

```
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

4) **Schedule synchronization class;**`WarehouseCalloutService`

```
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:
```

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

```
            }
        }
    }

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

}
```

Schedule synchronization.cls

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

5) Test automation logic

MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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

                    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;
        }
    }
}
```

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
    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();

```

```
59          Case newCase = [Select id,
60                          subject,
61                          type,
62                          Equipment__c,
63                          Date_Reported__c,
64                          Vehicle__c,
65                          Date_Due__c
66                         from case
67                         where status ='New'];
68
69          Equipment_Maintenance_Item__c workPart = [select id
70                                                    from
    Equipment_Maintenance_Item__c
71                                                    where
    Maintenance_Request__c =:newCase.Id];
72          list<case> allCase = [select id from case];
73          system.assert(allCase.size() == 2);
74
75          system.assert(newCase != null);
76          system.assert(newCase.Subject != null);
77          system.assertEquals(newCase.Type, 'Routine Maintenance');
78          SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
79          SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
80          SYSTEM.assertEquals(newCase.Date_Reported__c,
    system.today());
81      }
82
83      @isTest
84      private static void testNegative(){
85          Vehicle__C vehicle = createVehicle();
86          insert vehicle;
87          id vehicleId = vehicle.Id;
88
89          product2 equipment = createEquipment();
90          insert equipment;
91          id equipmentId = equipment.Id;
92
93          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
94          insert createdCase;
```

```
95
96          Equipment_Maintenance_Item__c workP =
    createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
97          insert workP;
98
99          test.startTest();
100          createdCase.Status = 'Working';
101          update createdCase;
102          test.stopTest();
103
104          list<case> allCase = [select id from case];
105
106          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    [select id
107                                                      from
    Equipment_Maintenance_Item__c
108                                                      where
    Maintenance_Request__c = :createdCase.Id];
109
110          system.assert(equipmentMaintenanceItem != null);
111          system.assert(allCase.size() == 1);
112      }
113
114      @isTest
115      private static void testBulk(){
116          list<Vehicle__C> vehicleList = new list<Vehicle__C>();
117          list<Product2> equipmentList = new list<Product2>();
118          list<Equipment_Maintenance_Item__c>
    equipmentMaintenanceItemList = new
    list<Equipment_Maintenance_Item__c>();
119          list<case> caseList = new list<case>();
120          list<id> oldCaseIds = new list<id>();
121
122          for(integer i = 0; i < 300; i++){
123              vehicleList.add(createVehicle());
124              equipmentList.add(createEquipment());
125          }
126          insert vehicleList;
127          insert equipmentList;
128
```

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

136         }
137         insert equipmentMaintenanceItemList;
138 
139         test.startTest();
140         for(case cs : caseList){
141             cs.Status = 'Closed';
142             oldCaseIds.add(cs.Id);
143         }
144         update caseList;
145         test.stopTest();
146 
147         list<case> newCase = [select id
148                                     from case
149                                     where status ='New'];
150 
151 
152 
153         list<Equipment_Maintenance_Item__c> workParts = [select
    id
154                                                          from
    Equipment_Maintenance_Item__c
155                                                          where
    Maintenance_Request__c in: oldCaseIds];
156 
157         system.assert(newCase.size() == 300);
158 
159         list<case> allCase = [select id from case];
160         system.assert(allCase.size() == 600);
161     }
```

```
162 }
```

6) TEST callout Logic

WarehouseCalloutService.cls

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

3
4      //Write a class that makes a REST callout to an external
   warehouse system to get a list of equipment that needs to be
   updated.
5      //The callout's JSON response returns the equipment records
   that you upsert in Salesforce.
6
7      @future(callout=true)
8      public static void runWarehouseEquipmentSync(){
9          System.debug('go into runWarehouseEquipmentSync');
10         Http http = new Http();
11         HttpRequest request = new HttpRequest();
12
13         request.setEndpoint(WAREHOUSE_URL);
14         request.setMethod('GET');
15         HttpResponse response = http.send(request);
16
17         List<Product2> product2List = new List<Product2>();
18         System.debug(response.getStatusCode());
19         if (response.getStatusCode() == 200){
20             List<Object> jsonResponse =
   (List<Object>)JSON.deserializeUntyped(response.getBody());
21             System.debug(response.getBody());
22
23             //class maps the following fields:
24             //warehouse SKU will be external ID for identifying
   which equipment records to update within Salesforce
25             for (Object jR : jsonResponse){
26                 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

            }
        }
    }

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

}
```

WarehouseCalloutServiceMock.cls

```
@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;
    }
}
```

WarehouseCalloutServiceTest.cls

```
@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;
    }
}
```

7)

WarehouseSyncSchedule

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

WarehouseCalloutServiceMock.cls

```
@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;
    }
}
```

WarehouseSyncScheduleTest.cls

```
@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
());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
=: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State),
'JobId does not match');

        Test.stopTest();
    }
}
```