# Apex Specialist

Automate record creation

```
MaintenanceRequest


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


MaintenanceRequestHelper

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,
16                                                          (SELECT
    Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
17                                                          FROM
    Case WHERE Id IN :validIds]);
18              Map<Id,Decimal> maintenanceCycles = new
    Map<ID,Decimal>();
19
20              //calculate the maintenance request due dates by
    using the maintenance cycle defined on the related equipment
    records.
21              AggregateResult[] results = [SELECT
    Maintenance_Request__c,
22
    MIN(Equipment__r.Maintenance_Cycle__c)cycle
23                                          FROM
    Equipment_Maintenance_Item__c
24                                          WHERE
    Maintenance_Request__c IN :ValidIds GROUP BY
    Maintenance_Request__c];
25
26              for (AggregateResult ar : results){
27                  maintenanceCycles.put((Id)
    ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
28              }
29
30              List<Case> newCases = new List<Case>();
31              for(Case cc : closedCases.values()){
32                  Case nc = new Case (
33                      ParentId = cc.Id,
34                      Status = 'New',
35                      Subject = 'Routine Maintenance',
36                      Type = 'Routine Maintenance',
37                      Vehicle__c = cc.Vehicle__c,
38                      Equipment__c =cc.Equipment__c,
39                      Origin = 'Web',
40                      Date_Reported__c = Date.Today()
41                  );
42
43                  //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;
        }
    }
}
```

Synchronize Salesforce data with an external system

```
WarehouseCalloutService

public with sharing class WarehouseCalloutService implements
  Queueable {
```

```
4       private static final String WAREHOUSE_URL = 'https://th-

5
6       //Write a class that makes a REST callout to an external
    warehouse system to get a list of equipment that needs to be
    updated.
7       //The callout's JSON response returns the equipment records
    that you upsert in Salesforce.
8
9       @future(callout=true)
10      public static void runWarehouseEquipmentSync(){
11          System.debug('go into runWarehouseEquipmentSync');
12          Http http = new Http();
13          HttpRequest request = new HttpRequest();
14
15          request.setEndpoint(WAREHOUSE_URL);
16          request.setMethod('GET');
17          HttpResponse response = http.send(request);
18
19          List<Product2> product2List = new List<Product2>();
20          System.debug(response.getStatusCode());
21          if (response.getStatusCode() == 200){
22              List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
23              System.debug(response.getBody());
24
25              //class maps the following fields:
26              //warehouse SKU will be external ID for identifying
    which equipment records to update within Salesforce
27              for (Object jR : jsonResponse){
28                  Map<String,Object> mapJson =
    (Map<String,Object>)jR;
29                  Product2 product2 = new Product2();
30                  //replacement part (always true),
31                  product2.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
32                  //cost
33                  product2.Cost__c = (Integer) mapJson.get('cost');
34                  //current inventory
35                  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');
    }

}
```

Schedule synchronization

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

Test automation logic

```
MaintenanceRequest

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

MaintenanceRequestHelper

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);
                }
            }
        }
```

```
11
12          //When an existing maintenance request of type Repair or
    Routine Maintenance is closed,
13          //create a new maintenance request for a future routine
    checkup.
14          if (!validIds.isEmpty()){
15              Map<Id,Case> closedCases = new Map<Id,Case>([SELECT
    Id, Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,
16                                                          (SELECT
    Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
17                                                          FROM
    Case WHERE Id IN :validIds]);
18              Map<Id,Decimal> maintenanceCycles = new
    Map<ID,Decimal>();
19
20              //calculate the maintenance request due dates by
    using the maintenance cycle defined on the related equipment
    records.
21              AggregateResult[] results = [SELECT
    Maintenance_Request__c,
22
    MIN(Equipment__r.Maintenance_Cycle__c)cycle
23                                          FROM
    Equipment_Maintenance_Item__c
24                                          WHERE
    Maintenance_Request__c IN :ValidIds GROUP BY
    Maintenance_Request__c];
25
26              for (AggregateResult ar : results){
27                  maintenanceCycles.put((Id)
    ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
28              }
29
30              List<Case> newCases = new List<Case>();
31              for(Case cc : closedCases.values()){
32                  Case nc = new Case (
33                      ParentId = cc.Id,
34                      Status = 'New',
35                      Subject = 'Routine Maintenance',
36                      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 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++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equip

        }
        insert equipmentMaintenanceItemList;
```

```
        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);
    }
}
```

Test callout logic

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

5
```

```
6       //Write a class that makes a REST callout to an external
    warehouse system to get a list of equipment that needs to be
    updated.
7       //The callout's JSON response returns the equipment records
    that you upsert in Salesforce.
8
9       @future(callout=true)
10      public static void runWarehouseEquipmentSync(){
11          System.debug('go into runWarehouseEquipmentSync');
12          Http http = new Http();
13          HttpRequest request = new HttpRequest();
14
15          request.setEndpoint(WAREHOUSE_URL);
16          request.setMethod('GET');
17          HttpResponse response = http.send(request);
18
19          List<Product2> product2List = new List<Product2>();
20          System.debug(response.getStatusCode());
21          if (response.getStatusCode() == 200){
22              List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
23              System.debug(response.getBody());
24
25              //class maps the following fields:
26              //warehouse SKU will be external ID for identifying
    which equipment records to update within Salesforce
27              for (Object jR : jsonResponse){
28                  Map<String,Object> mapJson =
    (Map<String,Object>)jR;
29                  Product2 product2 = new Product2();
30                  //replacement part (always true),
31                  product2.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
32                  //cost
33                  product2.Cost__c = (Integer) mapJson.get('cost');
34                  //current inventory
35                  product2.Current_Inventory__c = (Double)
    mapJson.get('quantity');
36                  //lifespan
37                  product2.Lifespan_Months__c = (Integer)
```

```
    mapJson.get('lifespan');
38                  //maintenance cycle
39                  product2.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
40                  //warehouse SKU
41                  product2.Warehouse_SKU__c = (String)
    mapJson.get('sku');
42
43                  product2.Name = (String) mapJson.get('name');
44                  product2.ProductCode = (String)
    mapJson.get('_id');
45                  product2List.add(product2);
46              }
47
48              if (product2List.size() > 0){
49                  upsert product2List;
50                  System.debug('Your equipment was synced with the

51              }
52          }
53      }
54
55      public static void execute (QueueableContext context){
56          System.debug('start runWarehouseEquipmentSync');
57          runWarehouseEquipmentSync();
58          System.debug('end runWarehouseEquipmentSync');
59      }
60
61  }
62
63  WarehouseCalloutServiceMock
64
1   @isTest
2   global class WarehouseCalloutServiceMock implements
    HttpCalloutMock {
3       // implement http mock callout
4       global static HttpResponse respond(HttpRequest request) {
5
6           HttpResponse response = new HttpResponse();
7           response.setHeader('Content-Type', 'application/json');
8
```

```
    response.setBody('[{"_id":"55d66226726b611100aaf741","replacement
        response.setStatusCode(200);

        return response;
    }
}

WarehouseCalloutServiceTest

@IsTest
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);
    }
}
```

Test scheduling logic

```
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":false,"quantity":5,
        response.setStatusCode(200);

        return response;
    }
}

WarehouseSyncSchedule

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

WarehouseSyncScheduleTest
@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());
17          String jobId = System.schedule('Warehouse Time to
    ());
18          CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
    =: jobId];
19          System.assertEquals('WAITING', String.valueOf(c.State),
    'JobId does not match');
20
21          Test.stopTest();
22      }
23  }
```

# Process Automation Specialist

Install the Process Automation superbadge unmanaged package(package ID 04t46000001Zch4). If you have trouble installing a managed or unmanaged package or app from AppExchange, follow the steps in this article.Validation Rule

Check the function for Length.
Remember to check the NULL Values in Validation rule.
Queue Creation

This is straightforward normal Queue creation
Create Names with related to appropriate sales team.

Field Creations on Account Object

Number of deals Field should be a Roll-Up Summary take count of COUNT Opportunities
Number of won deals Field should be a Roll-Up Summary (COUNT Opportunity) with filter criteria of Closed Won
Amount of won deals Field should be  a Roll-Up Summary (SUM Opportunity) with filter criteria of Closed Won

Last won deal date Field should be  a Roll-Up Summary (MAX Opportunity)
Deal win percent Field should be  a Formula(Percentage field)  IF Number_of_deals__c greater than 0 the , Number_of_won_deals__c /Number_of_deals__c  otherwise Zero
Call for Service Field should be  a Formula (Date) IF(OR(TODAY() – 730 > Last_won_deal_date__c , TODAY() + 730 < Last_won_deal_date__c ), 'Yes','No')
Validation Rules on Account Object

For Customer – Channel
ISCHANGED( Name ) && ISPICKVAL(Type, "Customer – Channel")
For Customer – Direct
ISCHANGED( Name ) && ISPICKVAL(Type, "Customer – Direct" )

For Billing State/Province
NOT(

CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:" &

"IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:" &

"NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:" &

"WA:WV:WI:WY", BillingState))

For Billing Country
BillingCountry <> "US" && BillingCountry <> "USA" && BillingCountry <> "United States" &&  NOT( ISBLANK(BillingCountry ) )
For Billing Country
BillingCountry <> "US" && BillingCountry <> "USA" && BillingCountry <> "United States" &&  NOT( ISBLANK(BillingCountry ) )

For Shipping State/Province and Shipping Country

Create a object and make sure the object name should be Robot_Setup__c
Edit the Robot name(Standard field) switch the data type from Text to AutoNumber and make sure the display format should be ROBOT SETUP-{0000}
Create following fields with correct data type:
Date————————>Date__c————————->DATE

Notes————————-> Notes__c————————–>TEXT

Day of the Week——>Day_of_the_Week__c———>TEXT

Create Sales Process in Opportunity; the name should be RB Robotics Sales Process.
Create a record type; the name should be RB Robotics Process RT.
Add Awaiting Approval value in opportunity Stage don't forget to add RB Robotics Process RT record type.
Create a Checkbox field and Name it Approved.
Write a validation rule as below:
AND( Amount > 100000, Approved__c = False)

Approval Process Definition Detail: See the screenshot below for details

create Process Builder.

Name: Automate Opportunities

Create the flow to display products

Change the datatype for "Day of the week" field from TEXT to Formula (TEXT) and use the following the formula to get Day of the week
CASE( MOD( Date__c – DATE(1900, 1, 7), 7), 0, "Sunday", 1, "Monday", 2, "Tuesday", 3, "Wednesday", 4, "Thursday", 5, "Friday", 6, "Saturday","Error")

Or You can use this formula also instead of above formula

CASE(WEEKDAY( Date__c ),
1, "Sunday",
2, "Monday",
3, "Tuesday",
4, "Wednesday",
5, "Thursday",
6, "Friday",
7, "Saturday",
Text(WEEKDAY( Date__c )))

Create Another Process Builder (Name: Robot Setup)

Conditions are as below:

If Day of the week is Saturday , change [Robot_Setup__c].Date__c +2
If Day of the week is Saturday , change [Robot_Setup__c].Date__c +1
Activate the Process