

# Apex Specialist Superbadge

**These are the key points that we are doing in 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

The codes used in this superbadges are as follows:-

## **Automate Creation**

//MaintenanceRequest.apxt

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

//MaintenanceRequestHelper.apxc

```
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;
}
}

}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;
}
}
}

```

### **Synchronize Salesforce data with an external system**

```

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');
}

```

```

}

```

### **Schedule Synchronization**

```

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

```

### **Test automatic 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 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;
  }
}

```

### **//MaintenanceRequestHelperTest**

@isTest

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

system.assert(equipmentMaintenanceltem != 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> equipmentMaintenanceltemList = 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 equipmentMaintenanceItem__c;

    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**

#### **//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:
        //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');
}

}

//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, "name": "Generator 1000 kW", "maintenanceperiod": 365, "lifespan": 120, "cost": 5000, "sku": "100003"}, { "_id": "55d66226726b611100aaf742", "replacement": true, "quantity": 183, "name": "Cooling Fan", "maintenanceperiod": 0, "lifespan": 0, "cost": 300, "sku": "100004"}, { "_id": "55d66226726b611100aaf743", "replacement": true, "quantity": 143, "name": "Fuse 20A", "maintenanceperiod": 0, "lifespan": 0, "cost": 22, "sku": "100005"}]');
    }
}

```



```

        response.setStatusCode(200);

        return response;
    }
}

```

### **//Warehouse Sync Schedule**

```

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

```

### **Test Scheduling Logic**

#### **//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());
        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();
    }
}

```

#### **//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
        ,"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;
    }
}

```

### //WarehouseSyncSchedule

```

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

```

## **Process Automation Specialist Superbadge**

**These are the key points that we are doing in 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

## Automate Leads

Step-1 :

Validation rule on Lead to verify Country and State field. Lead Country must be either **US**, **USA** or **United State** or it can be blank. Lead State must be valid 2-digit US state.

Steps-2 :

Create Two Queues

1. Rainbow Sales.
2. Assembly System Sales.

Create an lead assignment rule and make 2 rules entries and give order accordingly. Based on LeadSource value lead should be assign to correct sales team (Queue).

```
If(LeadSource == 'Web'){
    Assign lead to Rainbow Sales Queue;
}
else if(LeadSource == 'Partner' || LeadSource == 'Purchased List'){
    Assign lead to Assembly System Sales Queue.
}
```

## Automate Accounts

Step-1

*Validation rule on ShippingCountry and BillingCountry* i.e. Shipping and Billing Country must be either **US**, **USA**, **United State** or it can be blank.

```
NOT (OR ( BillingCountry = 'US', BillingCountry = 'USA', BillingCountry = 'United State',
ISBLANK (BillingCountry) ),
    ShippingCountry = 'US', ShippingCountry = 'USA', ShippingCountry = 'United State',
ISBLANK(ShippingCountry) )
)
```

Step-2

*Validation rule on ShippingState and BillingState* i.e. ShippingState and BillingState must be a valid 2-digit US state.

```
OR( AND(
    LEN ( ShippingState ) > 2 ,
    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", ShippingState))),
    AND(LEN ( BillingState ) > 2 ,
    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)
    ) )
```

### Step-3

*Validation Rule on Name and Type Field* i.e. Restrict user to change Account Name if Account Type picklist field value is either Customer - Direct or Customer - Channel.

```
IF(AND
    (OR( ISPICKVAL(Type, 'Customer - Direct'), ISPICKVAL(Type, 'Customer - Channel')),
    NOT(ISNEW()))
    ), ISCHANGED(Name), null
)
```

### Step-4

Create some Roll-up Summary and formula field on Account Object.

Note:- Roll-up summary fields can be created on master side of master-detail relationship.

**Number of deals** This field display number of all deals(Opportunities) under an account. Count(Opportunities).

**Number of won deals** This field display number of opportunities with stage as Closed Won under an account.Count(Opportunities) with criteria as StageName equals to 'Closed Won'.

**Last won deal date** This field display date of last won opportunity  
MAX(ClosedDate) with criteria as StageName = 'Closed Won'.

**Deal win percent** Formula Field (Number of won deals / Number of deals)

**Amount of won deals** This field display sum of all won opportunities.  
SUM (Amount) with criteria as StageName = 'Closed Won'.

**Call for Service** Formula (Text) display Yes/No based on formula.  
Return 'Yes', if last won date is 2 year ago otherwise 'No'.  
IF( (TODAY() - Last\_won\_deal\_date\_\_c) / 365 <=2 , 'No',  
'Yes')

## **Create Robot Setup Object**

This challenge is all about setting-up Robot Setup Object and create some fields on it.

<b>Name</b>	:- ROBOT SETUP-{0000} Starts with 0
<b>Date</b>	:- Date type field
<b>Notes</b>	:- TextArea (255)
<b>Day of the Week</b>	:- Formula (Text)
<b>Opportunity</b>	:- Master-Detail Relationship with Opportunity.

## **Sales Process and Validate Opportunities**

## Steps-1

Configure Stage Picklist field :-

Prospecting  
Qualification  
Proposal/Price Quote  
Negotiation/Review  
Awaiting Approval  
Closed Won  
Closed Lost

## Step-2

*Validation rule on Amount and Approved\_\_c* i.e. prevent opportunity record being saved if amount is greater than \$10,00,00 and approved is set to false.

AND( Amount > 100000, ISPICKVAL(StageName, 'Closed Won'), Approved\_\_c = false )

## Step-3

New record type ***RB Robotics Process RT***

New Sales Process ***RB Robotics Sales Process***

## Automate Opportunities

Whenever an opportunity is created for Prospect type account

- Send Email alert to Finance Group. (Integration User)
- Create a Task for Account Owner.

Whenever an Opportunity is created for Customer type account

- Send an Email alert to Finance Group. (Integration User)

Create an approval process with entry criteria as followed :-

- Opportunity.Amount > 100000
- Email Template :- SALES: Opportunity Needs Approval
- Create Initial Submission Actions (Field Update) => Set Opportunity Stage to Waiting Approval.
- Create Final Approval Actions (Field Update) => Set Opportunity Stage to Closed Won

Whenever Opportunity is in Negotiation/Review Stage

- Submit opportunity for Approval

When deal is won i.e Opportunity stage is Closed Won

- Create Robot Setup Record
- Email Alert to Finance Group

Note :- Use Process Builder(PB) to setup all these automation and obviously in one PB.

## **Create Flow for Opportunities**

In this part of challenge, we gonna create a flow which takes some input from user and will return list of Product record.

Create a User-Guided Lightning Flow.

- Create three Choice type **Resources** and use them into Flow Screen  
Choice Value:-  
*RainbowBot*  
*CloudyBot*  
*Assembly System*
- Use **Get Record** data element to get all products where name contains the options which you have selected from flow screen.

## **Automate Setups**

In this part of challenge we need to setup Robot Setup Date

If Robot Setup Date / Day Of Week is either on Saturday or Sunday, then make it to Monday.

Robot Setup Date +2, for Saturday

Robot Setup Date +1, for Sunday

**Below are the codes that are used in the self paced learning modules related to Apex:-**

### **Apex Triggers**

#### **//Get Started with Apex Triggers**

```
trigger AccountAddressTrigger on Account (before insert, before update) {
```

```
    for(Account a : Trigger.new){
        If (a.Match_Billing_Address__c == true) {
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

#### **//Bulk Apex Triggers**

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
```

```
    List<Task> taskList = new List<Task>();
    for(Opportunity opp : Trigger.new) {
```

```
        //Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on Create
```



```

        if(Trieger.isInsert) {
            if(Opp.StageName == 'Closed Won') {
                taskList.add(new Task(Subject = 'Follow Up Test Task',
WhatId = opp.Id));
            }
        }

        //Only create Follow Up Task only once when Opp StageName changed to
        'Closed Won' on Update
        if(Trieger.isUpdate) {
            if(Opp.StageName == 'Closed Won'
            && Opp.StageName != Trieger.oldMap.get(opp.Id).StageName) {
                taskList.add(new Task(Subject = 'Follow Up Test Task',
WhatId = opp.Id));
            }
        }
    }

    if(taskList.size()>0) {
        insert taskList;
    }
}

```

## **Apex Testing**

### **//Get Started with Apex Unit Tests**

```

@isTest
private class TestVerifyDate {
    static testMethod void TestVerifyDate() {
        VerifyDate.CheckDates(System.today(),System.today().addDays(10));
        VerifyDate.CheckDates(System.today(),System.today().addDays(78));
    }
}

```

## **//Test Apex Triggers**

```
//class
trigger RestrictContactByName on Contact (before insert, before update) {
    //check contacts prior to insert or update for invalid data
    For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
            c.AddError('The Last Name "' + c.LastName + '" is not allowed for DML');
        }
    }
}

//Test class
@isTest
private class TestRestrictContactByName {

    static testMethod void metodoTest()
    {
        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
        Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
        listContact.add(c1);
        listContact.add(c2);
        Test.startTest();
        try
        {
            insert listContact;
        }
        catch(Exception ee)
        {
        }
        Test.stopTest();
    }
}
```

## **//Create Test Data for Apex Testing**

```

public class RandomContactFactory{

    public static List<Contact> generateRandomContacts(integer n,string LastName){
        integer n1=n;
        List<contact> c1 = new list<contact>();
        list<contact> c2 =new list<contact>();
        c1 = [select FirstName from Contact Limit : n1];
        integer i=0;
        for(contact cnew : c1){
            contact cnew1 = new contact();
            cnew1.firstname = cnew.firstname + i;
            c2.add(cnew1);
            i++;
        }
        return c2;
    }
}

```

## **Asynchronous Apex**

### **//Use Future Methods**

```

//Apex class
public class AccountProcessor
{
    @future
    public static void countContacts(Set<id> setId)
    {
        List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from
contacts ) from account where id in :setId ];
        for( Account acc : lstAccount )
        {
            List<Contact> lstCont = acc.contacts ;
            acc.Number_of_Contacts__c = lstCont.size();
        }
    }
}

```

```

        update lstAccount;
    }
}
//Test class
@IsTest
public class AccountProcessorTest {
    public static testmethod void TestAccountProcessorTest()
    {
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact cont = New Contact();
        cont.FirstName = 'Bob';
        cont.LastName = 'Masters';
        cont.AccountId = a.Id;
        Insert cont;
        set<Id> setAcId = new Set<ID>();
        setAcId.add(a.id);

        Test.startTest();
        AccountProcessor.countContacts(setAcId);
        Test.stopTest();
        Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT
1];
        System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
    }
}

```

### **//Use Batch Apex**

```

//Batch job
global class LeadProcessor implements Database.Batchable<Sobject>
{
    global Database.QueryLocator start(Database.BatchableContext bc)
    {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
}

```

```

global void execute(Database.BatchableContext bc, List<Lead> scope)
{
    for (Lead Leads : scope)
    {
        Leads.LeadSource = 'Dreamforce';
    }
    update scope;
}

global void finish(Database.BatchableContext bc){ }
}

//Test class
@isTest
public class LeadProcessorTest
{
    static testMethod void testMethod1()
    {
        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            Lead led = new Lead();
            led.FirstName ='FirstName';
            led.LastName ='LastName'+i;
            led.Company ='demo'+i;
            lstLead.add(led);
        }
        insert lstLead;
        Test.startTest();

        LeadProcessor obj = new LeadProcessor();
        DataBase.executeBatch(obj);
        Test.stopTest();
    }
}

```

**//Control processes with Queueable Apex**

```

global class LeadProcessor implements Database.Batchable<Subject>
{
    global Database.QueryLocator start(Database.BatchableContext bc)
    {
        return Database.getQueryLocator([Select LastName From Lead ]);
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope)
    {
        for (Lead Leads : scope)
        {
            Leads.LeadSource = 'Dreamforce';
        }
        update scope;
    }

    global void finish(Database.BatchableContext bc){ }
}

@isTest
public class LeadProcessorTest
{
    static testMethod void testMethod1()
    {
        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            Lead led = new Lead();
            led.FirstName ='FirstName';
            led.LastName ='LastName'+i;
            led.Company ='demo'+i;
            lstLead.add(led);
        }
        insert lstLead;
        Test.startTest();

        LeadProcessor obj = new LeadProcessor();
        DataBase.executeBatch(obj);
    }
}

```

```

        Test.stopTest();
    }
}

```

### **//Schedule jobs using the Apex Scheduler**

```

global class DailyLeadProcessor implements Schedulable {

    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];
        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
            }
            update lList;
        }
    }
}

```

### **Apex Integration Service**

#### **//Apex REST Callouts**

```

public class AnimalLocator{
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        HttpResponse res = h.send(req);
        Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
        Map<String, Object> animal = (Map<String, Object>) results.get('animal');
        return (String)animal.get('name');
    }
}

```

#### **//Apex SOAP Callouts**

```

public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new parkService.parksImplPort();
        return park.byCountry(country);
    }
}

```

```

    }
}
//Generated by wsdl2apex
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();

```



```

        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
            this,
            request_x,
            response_map_x,
            new String[]{endpoint_x,
                "",
                'http://parks.services/',
                'byCountry',
                'http://parks.services/',
                'byCountryResponse',
                'ParkService.byCountryResponse'}
        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
    }
}
}
}

@Test
private class ParkLocatorTest {
    @Test static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        //Double x = 1.0;
        //Double result = AwesomeCalculator.add(x, y);
        String country = 'Germany';
        String[] result = ParkLocator.Country(country);
        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park',
'Hainich National Park', 'Bavarian Forest National Park'}, result);
    }
}

@Test
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke(
        Object stub,

```

```

    Object request,
    Map<String, Object> response,
    String endpoint,
    String soapAction,
    String requestName,
    String responseNS,
    String responseName,
    String responseType) {
    // start - specify the response you want to send
    parkService.byCountryResponse response_x = new
    parkService.byCountryResponse();
    response_x.return_x = new List<String>{'Hamburg Wadden Sea National Park',
    'Hainich National Park', 'Bavarian Forest National Park'};
    //calculatorServices.doAddResponse response_x = new
    calculatorServices.doAddResponse();
    //response_x.return_x = 3.0;
    // end
    response.put('response_x', response_x);
}
}

```

### **//Apex Web Services**

```

@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
            FROM Account WHERE Id = :accId];
        return acc;
    }
}

//TEST CLASS
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {

```

```

    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId + '/contacts' ;
    request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount != null);
    System.assertEquals('Test record', thisAccount.Name);

}

// Helper method
static Id createTestRecord() {
    // Create test record
    Account TestAcc = new Account(
        Name='Test record');
    insert TestAcc;
    Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
    return TestAcc.Id;
}
}

```