

# Apex Specialist Super badge

## Challenge 1:

This is the first challenge where we attend a quiz answering some general questions regarding the super badge challenge that we are doing.

## Challenge 2:

It is all about preparing my organization with the necessary package installations and customizations as per given in the Prepare Your Organization section to complete the Apex Specialist Super badge.

## Challenge 3:

In this challenge we automate record creation using apex class and apex trigger by creating an apex class called MaintenanceRequestHelper and an apex trigger called MaintenanceRequest.

### Apex Trigger code:

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

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

### Apex Class code:

```
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;

```

```

    }
}
}

```

## Challenge 4:

In challenge three we synchronize Salesforce data with an external system using apex class of name WarehouseCalloutService which is already given and after writing code in it and executing it anonymously in a separate window, the process will be successful.

### Apex Class code:

```

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');
}

}
```

### **Execute anonymous window:**

```
WarehouseCalloutService.runWarehouseEquipmentSync();
```

## **Challenge 5:**

In challenge four we will be scheduling our synchronization using WarehouseSyncSchedule in the apex class and execute a code in an anonymous window

### **Apex Class code:**

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

### **Execute anonymous window:**

```
WarehouseSyncSchedule.scheduleInventoryCheck()
```

## **Challenge 6:**

In this challenge we are testing our automation logic using apex trigger class MaintenanceRequest and three apex classes where two are used for testing and one is used for sharing and those classes are given below.

### **Apex Trigger code:**

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

## Apex Class code:

```
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;
}
}
}

```

### **Apex Class code:**

```

@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 equipmentMaintenanceltemList;

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

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

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

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

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

```

## Challenge 7:

In challenge six we are testing our call out logic by using two apex classes which are used for testing where one of the classes implements HTTPCalloutMock.

## Apex Class code:

```
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');
}

}

```

### **Apex Class code:**

```

@IsTest
private class WarehouseCalloutServiceTest {
    / implement your mock call out test here@isTest
    static void testWareHouseCallout(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync(); } }

```

### **Apex Class code:**

```

@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":"55d66226726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}');
    response.setStatusCode(200);

    return response;
}
}
```

## Challenge 8:

In this challenge we are testing our Scheduling logic by using a apex test class to testour scheduling logic and the code is given below.

## Apex Class code:

```
@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();  
}  
}
```

## Process Automation Specialist Super badge

### Challenge 1:

It is the same as the previous super badge challenge 1 where we answer a quiz before moving into the actual Super badge challenges.

### Challenge 2:

This challenge is all about automating leads where we create a Validation rule under leads and you can give any Rule Name and the Error condition formula will be given below for validating leads. After this we have to create two Queues with the given name as per in the instruction of the challenge and then create an assignment rule. If all these things are done properly, the challenge will be completed without any problems.

#### Error Condition Formula:

```
OR(AND(LEN(State)>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", State))),  
NOT(OR(Country ="US",Country ="USA",Country ="United States", ISBLANK(Country))))
```

## Lead Assignment Rule

## Trailhead

Add rule entries that specify the criteria used to route leads. You can reorder rule entries on this page after you create them.

## Rule Detail

[Edit](#)Rule Name **Trailhead**Active ☒Created By [Nidhi Gupta](#), 2020-06-03, 8:33 a.m.Modified By [Nidhi Gupta](#), 2020-06[Edit](#)

## Rule Entries

[New](#)[Reorder](#)

Action	Order	Criteria	Assign To
<a href="#">Edit</a>   <a href="#">Del</a>	<input type="text" value="1"/>	Lead: Lead Source EQUALS Web	<a href="#">Rainbow Sales</a>
<a href="#">Edit</a>   <a href="#">Del</a>	<input type="text" value="2"/>	Lead: Lead Source NOT EQUAL TO Web	<a href="#">Assembly System Sales</a>

## Rule Entry Edit

## Trailhead

## Enter the rule entry


[Save](#)[Save & New](#)[Cancel](#)

## Step 1: Set the order in which this rule entry will be processed

Sort Order  

## Step 2: Select the criteria for this rule entry

Run this rule if the  :

Field	Operator	Value	
<input type="text" value="Lead: Lead Source"/>	<input type="text" value="not equal to"/>	<input type="text" value="Web"/>	 AND
<input type="text" value="--None--"/>	<input type="text" value="--None--"/>	<input type="text"/>	AND
<input type="text" value="--None--"/>	<input type="text" value="--None--"/>	<input type="text"/>	AND
<input type="text" value="--None--"/>	<input type="text" value="--None--"/>	<input type="text"/>	AND
<input type="text" value="--None--"/>	<input type="text" value="--None--"/>	<input type="text"/>	

[Add Filter Logic...](#)

## Step 3: Select the user or queue to assign the Lead to

Queue ☐ Do Not Reassign Owner

Email Template

## Challenge 3:

In this challenge we are given the task of automating accounts by creating Roll Up Summary fields as it is given in the instructions and after that by creating two Error Condition Formulas we automate our accounts and the code will be given below for these two formulas

### Error Condition Formula1:

```
OR(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)) ),  
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)) ),  
NOT(OR(BillingCountry = "US",BillingCountry="USA",BillingCountry = "United States",  
ISBLANK(BillingCountry))),  
NOT(OR(ShippingCountry = "US",ShippingCountry = "USA",ShippingCountry = "United States",ISBLANK(ShippingCountry))))
```

### Error Condition Formula2:

```
ISCHANGED( Name ) && ( OR( ISPICKVAL( Type , 'Customer - Direct' ) ,ISPICKVAL( Type  
, 'Customer - Channel' ) ) )
```

## Challenge 4:

It is the easiest challenge in this superbadge where we don't have to do a lot of things, we only have to create Robot Setup object with a master-detail relationship with the opportunity and then create a few fields as per given in the challenge instructions.

## Challenge 5:

In this challenge we are creating a Sales Process and Validating its opportunities. First we have to create a field with checkbox type with the name Approval where it can only be viewed by System Administrators and Sales Managers. Then we have to add a picklist value as Awaiting Approval to the field Stage. Lastly we have to add the desired fields and then add a Validation rule in the Opportunity object.

### Validation Rule:

```
IF(( Amount > 100000 && Approved c <> True && ISPICKVAL( StageName,'Closed Won')  
,True,False)
```

## Challenge 6:

In this challenge we are Automating Opportunities, First we have to create three Email Templates upon reading instructions and create a approval process by selecting opportunity object in the approval process with the necessary field updates in the process and set a criteria where this process will only run if the criteria is met.

Then go to the process builder and start building a process by selecting a object first and by setting four criteria where each criteria will do a action upon meeting the criteria.

**Process Definition Detail** [Edit](#) [Clone](#) [Deactivate](#)

Process Name: prospect Active ☒

Unique Name: prospect Next Automated Approver Determined By

Description:

Entry Criteria: (Opportunity: Stage EQUALS Negotiation/Review) AND (Opportunity: Amount GREATER THAN 100000)

Record Editability: Administrator ONLY Allow Submitters to Recall Approval Requests ☐

Approval Assignment Email Template: [SALES Opportunity Needs Approval](#)

Initial Submitters: Opportunity Owner

Created By: [Nishi Gupta](#), 2020-06-04, 9:42 a.m. Modified By: [Nishi Gupta](#), 2020-06-05, 3:36 a.m.

---

**Initial Submission Actions** [Add Existing](#) [Add New](#)

Action	Type	Description
<a href="#">Edit</a>	Record Lock	Lock the record from being edited
<a href="#">Edit</a>   <a href="#">Remove</a>	Field Update	<a href="#">approval update</a>

---

**Approval Steps** [Add Existing](#) [Add New](#)

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
<a href="#">Show Actions</a>   <a href="#">Edit</a>	1	Step 1			<a href="#">User Nishi Davoud</a>	Final Rejection

---

**Final Approval Actions** [Add Existing](#) [Add New](#)

Action	Type	Description
<a href="#">Edit</a>	Record Lock	Lock the record from being edited
<a href="#">Edit</a>   <a href="#">Remove</a>	Field Update	<a href="#">won deal</a>

---

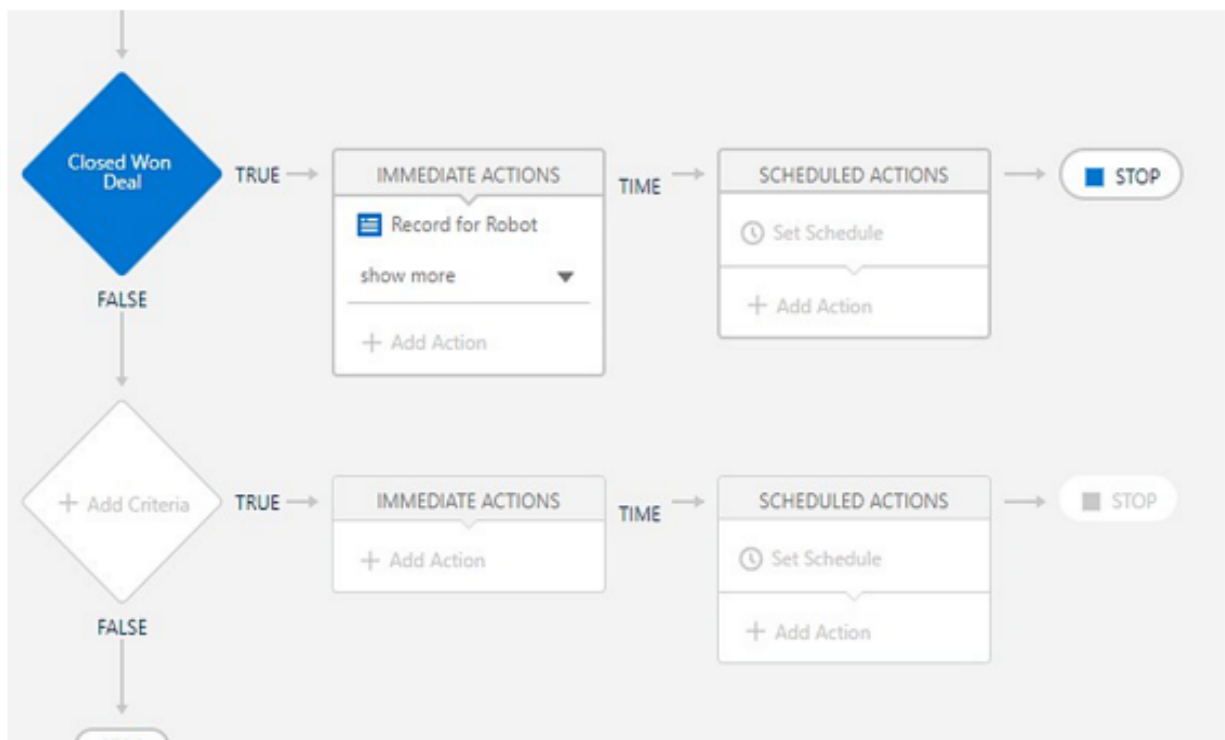
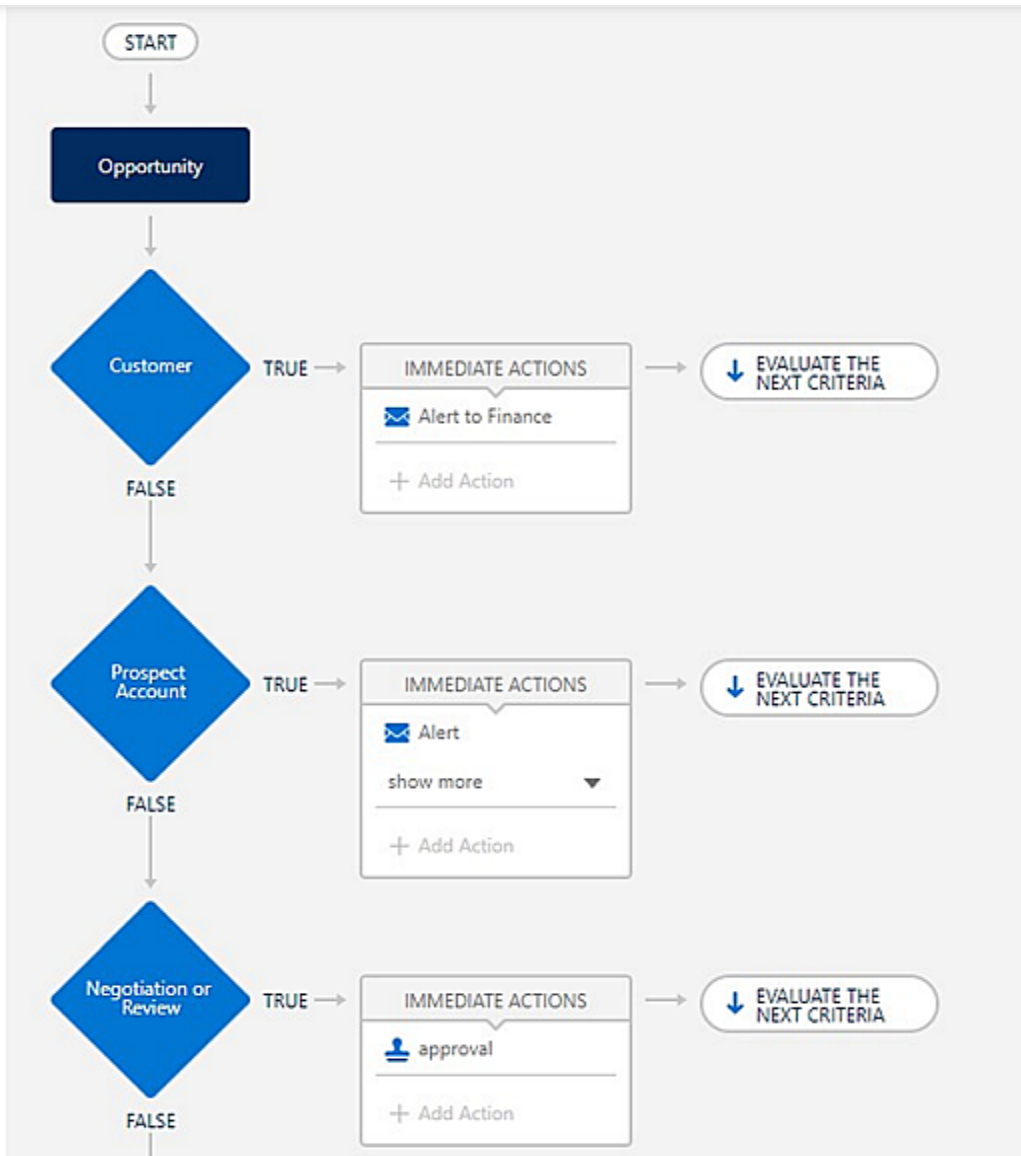
**Final Rejection Actions** [Add Existing](#) [Add New](#)

Action	Type	Description
<a href="#">Edit</a>	Record Lock	Unlock the record for editing
<a href="#">Edit</a>   <a href="#">Remove</a>	Field Update	<a href="#">open</a>

---

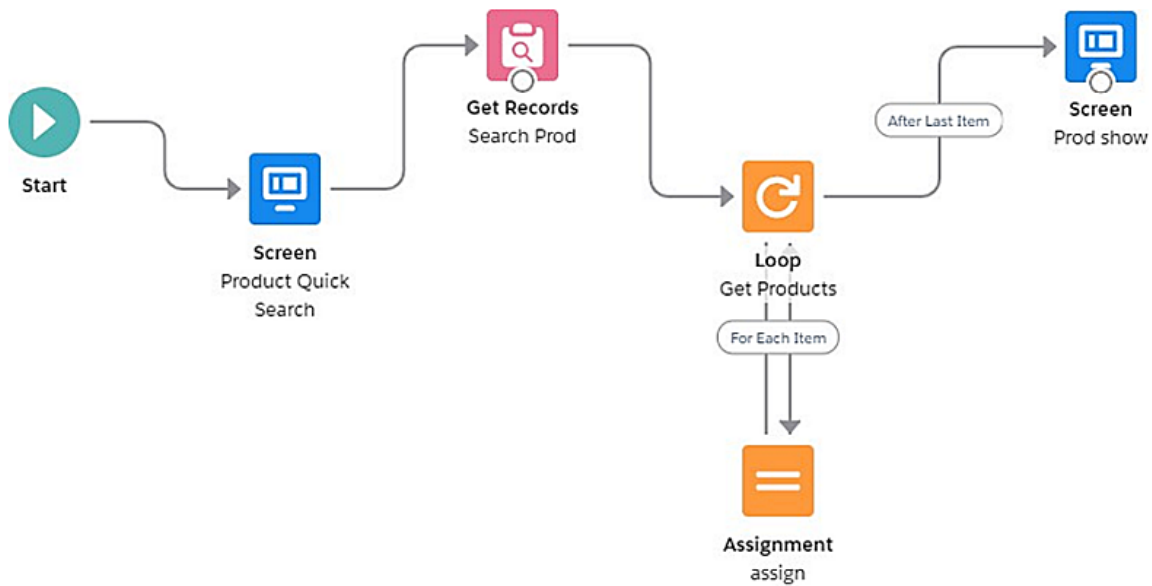
**Recall Actions** [Add Existing](#) [Add New](#)

Action	Type	Description
--------	------	-------------



## **Challenge 7:**

In this challenge we are creating Flow for Opportunities, First with a Start element then Screen element where it then gets Records and there's a loop to get each record and after that the process ends with a screen element where it shows the products. The products are created as per given in the challenge instructions to successfully complete the challenge.



Create Flow for Opportunities

## Challenge 8:

It is the last challenge of the super badge where we Automate Setups, First we have to change the formula in one of the fields of the Robot object where the Formula will be given below and then we have go to the flows process that we created previously and clone it to make changes where we change the formula for the last criteria to Automate setups according to dates.

### Formula 1:

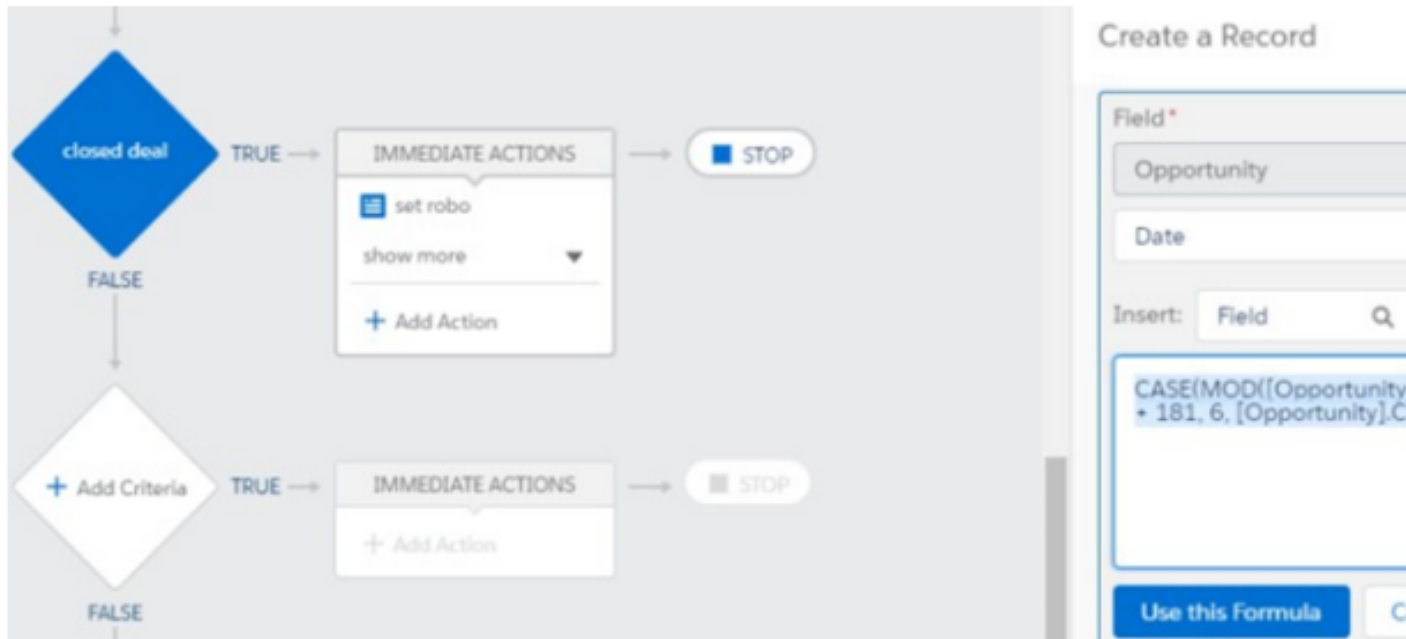
```

Case ( WEEKDAY(Date c ),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
5,"Thursday",
6,"Friday",
7,"Saturday",
Text(WEEKDay(Date c)))
  
```

### Formula 2:

```

CASE(MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7), 0, [Opportunity].CloseDate + 181, 6, [Opportunity].CloseDate + 182, [Opportunity].CloseDate + 180)
  
```



And with this you will have successfully completed this Super badge.

## Apex Triggers

### Get Started with Apex Triggers:

#### Apex trigger:

```
trigger AccountAddressTrigger on Account(before insert,before update){
    List acclst=new List();
    for(account a:trigger.new){
        if(a.Match_Billing_Address c==true && a.BillingPostalCode!=null){
            a.ShippingPostalCode=a.BillingPostalCode;
        }
    }
}
```



```

    }
  }
}

```

## Bulk Apex Triggers:

### Apex Trigger:

```

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
  List tasks = new List();
  for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE
  StageName='ClosedWon' AND Id IN :Trigger.new]){
    tasks.add(new Task(Subject = 'Follow Up Test Task' , WhatId = opp.Id));
    } if(tasks.size() > 0){
  Insert tasks;
  }
}

```

## Apex Testing

### Get Started with Apex Unit Tests:

#### Apex class:

```

@isTest private class TestVerifyDate {
  @isTest static void testWithin30Days() {
    Date Datetest = VerifyDate.CheckDates(System.today(),
    System.today()+10);System.assertEquals(System.today()+10, Datetest);
  }
  @isTest static void testSetEndOfMonth() {
    Date Datetest = VerifyDate.CheckDates(System.today(), System.today()+52);
    System.assertEquals(System.today()+27, Datetest);
  }
}

```

## Test Apex Triggers:

### Apex class:

```
@isTest private class TestRestrictContactByName {
    static testMethod void metodoTest() {
        List listContact= new List();
        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 e) {}
        Test.stopTest();
    }
}
```

## Create Test Data for Apex Tests:

### Apex class:

```
public with sharing class RandomContactFactory {
    public static List generateRandomContacts( Integer noOfContacts, String lastName ) {
        List contacts = new List(); for( Integer i = 0; i < noOfContacts; i++ ) {
            Contact con = new Contact( FirstName = 'Test '+i, LastName = lastName );
            contacts.add( con );
        }
        return contacts;
    }
}
```

## Asynchronous Apex

## Use Future Methods:

### Apex class:

```
public class AccountProcessor{
    @future public static void countContacts(List accountIds){
        List vAccountList = new List();
        List acc = [SELECT Id,Name, (SELECT Id,Name FROM Contacts) FROM Account WHERE Id IN
:accountIds];
        System.debug('total contact in Account: ' + acc);
        if(acc.size() > 0){
            for(Account a: acc){
                List con = [SELECT Id,Name FROM Contact WHERE accountId = :a.Id];
                a.Number_of_Contacts c = con.size();
                vAccountList.add(a);
            }
            if(vAccountList.size()>0) {
                update vAccountList;
            }
        }
    }
}
```

## Test Class:

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account(Name = 'Acme1');
        Insert a; Account b = new Account(Name = 'Acme2');
        insert b; Contact c = new Contact(FirstName = 'Gk', LastName = 'Gupta', accountId= a.Id);
        insert c; Contact c1 = new Contact(FirstName = 'Gk1',LastName = 'Gupta1', accountId =
b.Id);insert c1;
        List acnt = [SELECT Id FROM Account WHERE Name = :a.Name OR Name = :b.Name];
        System.debug('size of acnt: ' + acnt);
        List acntIDLST = new List();
        for(Account ac: acnt){
```

```

        acctIDLST.add(ac.Id);
    }
    Test.startTest();
    AccountProcessor.countContacts(acntIDLST);
    Test.stopTest();
}
}

```

## Use Batch Apex:

### Apex class:

```

global class LeadProcessor implements Database.Batchable {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    global void execute(Database.BatchableContext bc, List scope) {
        for (LeadLeads : scope) {
            Leads.LeadSource = 'Dreamforce';
        } update scope;
    }
    global void finish(Database.BatchableContext bc){}
}

@isTest
public class LeadProcessorTest {
    static testMethod void testMethod1() {
        List lstLead = new List();
        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:

### Apex class:

```
public class AddPrimaryContact implements Queueable {
    private Contact c;
    private String state;
    public AddPrimaryContact(Contact c, String state) {
        this.c = c; this.state = state;
    }
    public void execute(QueueableContext context) {
        List ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM
ACCOUNTWHERE BillingState = :state LIMIT200];
        List lstContact = new List();
        for (Account acc:ListAccount) {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId = acc.id; lstContact.add( cont );
        }
        if(lstContact.size() >0) {
            insert lstContact;
        }
    }
}

@isTest
public class AddPrimaryContactTest {
    @isTest
    static void TestList() {
        List Test e = new List ();
        for(Integer i=0;i<50;i++) {
            Test e.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++) {
            Test e.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
        Contact co = new Contact();
        co.FirstName='demo';
    }
}
```

```

        co.LastName = 'demo';
        insert co;
        String state = 'CA';
        AddPrimaryContact apc = new AddPrimaryContact(co, state);
        Test.startTest();
        System.enqueueJob(apc);
        Test.stopTest();
    }
}

```

## Schedule Jobs Using the Apex Scheduler:

### Apex class:

```

global class DailyLeadProcessor implements Schedulable {global void
execute(SchedulableContext ctx) {
    List lList = [Select Id, LeadSource from Lead where LeadSource = null];
    if(!lList.isEmpty()){
        for(Lead l: lList) {
            l.LeadSource = 'Dreamforce';
        }
        update lList;
    }
}
}
@isTest
public class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
    static testMethod void testDailyLeadProcessorTest() {
        List listLead = new List();
        for (Integer i=0; i<200; i++) {
            Lead ll = new Lead();
            ll.LastName = 'Test' + i; ll.Company = 'Company'+ i; ll.Status = 'Open - Not Contacted';
listLead.add(ll);
        }
        insert listLead;
        Test.startTest();
        DailyLeadProcessor daily = new DailyLeadProcessor();
        String jobId = System.schedule('Update LeadSource to Dreamforce', CRON_EXP, daily);
        List liss = new List([SELECT Id, LeadSource FROM Lead WHERE LeadSource !=

```

```

'Dreamforce']);
    Test.stopTest();
}
}

```

## Apex Integration Services

### Apex Rest Callouts:

#### Apex class:

```

public class AnimalLocator {
    public static String getAnimalNameById(Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        /*Map results = (Map)JSON.deserializeUntyped(response.getBody()); system.debug('---
>results'+results); List animals = (List) results.get('animal'); system.debug('----- Map
mapAnimal = new Map();
        Integer varId; String varName;
        JSONParser parser1= JSON.createParser(response.getBody());
        while (parser1.nextToken() != null) {
            if ((parser1.getCurrentToken() == JSONTOKEN.FIELD_NAME) && (parser1.getText() ==
'id')){ }
            / Get the value. parser1.nextToken();
            / Fetch the ids for all animals in JSON Response.
            varId=parser1.getIntegerValue();
            System.debug('--- >varId-->'+varId);
            parser1.nextToken();
            if ((parser1.getCurrentToken() == JSONTOKEN.FIELD_NAME) && (parser1.getText() ==
'name')) {
                parser1.nextToken();

```

```

        / Fetch the names for all animals in JSON Response.
        varName=parser1.getText();
        System.debug('---->varName-->'+varName);
    }
    mapAnimal.put(varId,varName);
}
system.debug('---->mapAnimal-->'+mapAnimal); return mapAnimal.get(id);
}
} Mock Test Class:
@Test
global class AnimalLocatorMock implements HttpCalloutMock {
    / Implement this interface method global
    HTTPResponse respond(HTTPRequest request)
    / Create a fake response
    HTTPResponse response = new HTTPResponse();
    response.setHeader('Content-Type','application/json'); response.setBody('{ "animal": [{ "id": 1, "name":
    "chicken", "eats": "chicken food", "says": "cluck cluck"}, { "id": 2, "name": "duck", "eats": "worms", "says": "pek
    pek"} ] }');
    response.setStatusCode(200);
    return response;
}
}

```

Test Class:

```

@Test
private class AnimalLocatorTest {
    @Test
    static void testGetCallout() {
        / Set mock callout class
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        / This causes a fake response to be sent / from the class that implements HttpCalloutMock.
        String response = AnimalLocator.getAnimalNameById(1);
        system.debug('Test Response1---->'+response);
        String expectedValue = 'chicken';
        System.assertEquals(expectedValue, response);
        String response2 = AnimalLocator.getAnimalNameById(2);
        system.debug('Test Response2---->'+response2);
        String expectedValue2 = 'duck';
        System.assertEquals(expectedValue2, response2);
    }
}

```



```
}
```

## Apex SOAP Call outs:

### Apex class:

Service:

/ 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'};
        privateString[] apex_schema_type_info = new String[]{'http:/ parks.services/',false,false};
        privateString[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        publicString arg0;
        private String[] arg0_type_info = new String[]{'arg0','http:/ parks.services/',null,'0','1','false'};
        privateString[] apex_schema_type_info = new String[]{'http:/ parks.services/',false,false};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        publicString endpoint_x = 'https:/ th-apex-soap-service.herokuapp.com/service/parks';
        public Map inputHttpHeaders_x; publicMap outputHttpHeaders_x;
        public StringclientCertName_x;
        public String clientCert_x;
        publicString clientCertPasswd_x;public Integer timeout_x;
        privateString[] 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 response_map_x = new Map();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke( this, reque st_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;
    }
}

```

Class:

```

public class ParkLocator {
    public static String[]country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country); return parksname;
    }
}

```

Test:

```

@Test
private class ParkLocatorTest{
    @Test
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[]arrayOfParks = ParkLocator.country('India');
        System.assertEquals('Park1', arrayOfParks[0]);
    }
}

```

Mock Test:

```

@Test
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke( Object stub, Object request, Map response,String endpoint, String
soapAction, String requestName, String responseNS, String responseName, String
responseType) {
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        List lstOfDummyParks = new List {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;
        response.put('response_x', response_x);
    }
}

```

## Apex Web Services:

### Apex class:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        String accountId = request.requestURI.substringBetween('Accounts/', '/contacts');
        system.debug(accountId);
        Account objAccount = [SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM Account
WHERE Id = :accountId LIMIT 1];
        return objAccount;
    }
}

/ Test class

@isTest

private class AccountManagerTest{
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'test Account'); insert objAccount; Contact
objContact = new Contact(LastName = 'test Contact', AccountId = objAccount.Id);
        insert objContact; Id recordId = objAccount.Id; RestRequest request = new RestRequest();
        request.requestUri = 'https://sandeepidentity-dev-
ed.my.salesforce.com/services/apexrest/Accounts/' + recordId + '/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        / Call the method to test Account
        thisAccount = AccountManager.getAccount();
        / Verify
        resultsSystem.assert(thisAccount != null);
        System.assertEquals('test Account', thisAccount.Name);
    }
}
```

# Lightning Web Components

## Deploy Lightning Web ComponentFiles:

### bikeCard.html:

```
<template>
  <div>
    <div>Name: {name}</div>
    <div>Description: {description}</div>
    <lightning-badge label={material}></lightning-badge>
    <lightning-badge label={category}></lightning-badge>
    <div>Price: {price}</div>
    <div><img src={pictureUrl}/></div>
  </div>
</template>
```

### bikeCard.js

```
import { LightningElement } from 'lwc';
export default class BikeCard extends LightningElement {
  name = 'ElectraX4';
  description = 'A sweet bike built for comfort.';
  category = 'Mountain';
  material = 'Steel';
  price = '$2,700';
  pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';
}
```

### bikeCard.js-meta.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <!-- The apiVersion may need to be increased for the current release -->
  <apiVersion>52.0</apiVersion>
  <isExposed>true</isExposed>
  <masterLabel>Product Card</masterLabel>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```