1-DailyLead Processor- Class Code and isTest Code.

```
global class DailyLeadProcessor implements Schedulable{

   global void execute (SchedulableContext ctx){
      List<lead> leadstoupdate = new List<lead>();
      List<Lead> leads= [Select id From Lead where LeadSource= Null Limit 200 ];

      for(Lead l:leads){
         l.LeadSource = 'Dreamforce';
         leadstoupdate.add(l);
      }

      update leadstoupdate;
   }

}

@isTest

private class DailyLeadProcessorTest {

   public static String CRON_EXP ='0 0 0 15 3 ? 2025';
   static testmethod void testScheduledJob()
   {
      List<Lead> leads = new List<lead>();
      for(Integer i=0; i<200; i++){
         Lead l = new Lead(
            FirstName= 'First'+ i,
            LastName= 'LastName',
            Company= 'The Inc');
         leads.add(l);
      }
      insert leads;


      Test.startTest();

      String jobId = System.Schedule('ScheduledApexTest'+
string.valueOf(system.currentTimeMillis()), CRON_EXP, new DailyLeadProcessor());
      Test.stopTest();
```

```
        List<Lead> Checkleads = new List<Lead>();
        checkleads = [Select Id From Lead Where LeadSource = 'Dreamforce' and Company= 'The
Inc'];

        System.assertEquals(200, checkleads.size(), 'Leads were not created ');
    }

}
```

ContactsTodayControllers

```
public class ContactsTodayController {

    @AuraEnabled
    public static List<Contact> getContactsForToday() {

        List<Task> my_tasks = [SELECT Id, Subject, WhoId FROM Task WHERE OwnerId =
:UserInfo.getUserId() AND IsClosed = false AND WhoId != null];
        List<Event> my_events = [SELECT Id, Subject, WhoId FROM Event WHERE OwnerId =
:UserInfo.getUserId() AND StartDateTime >= :Date.today() AND WhoId != null];
        List<Case> my_cases = [SELECT ID, ContactId, Status, Subject FROM Case WHERE
OwnerId = :UserInfo.getUserId() AND IsClosed = false AND ContactId != null];

        Set<Id> contactIds = new Set<Id>();
        for(Task tsk : my_tasks) {
            contactIds.add(tsk.WhoId);
        }
        for(Event evt : my_events) {
            contactIds.add(evt.WhoId);
        }
        for(Case cse : my_cases) {
            contactIds.add(cse.ContactId);
        }

        List<Contact> contacts = [SELECT Id, Name, Phone, Description FROM Contact WHERE
Id IN :contactIds];

        for(Contact c : contacts) {
            c.Description = '';
            for(Task tsk : my_tasks) {
                if(tsk.WhoId == c.Id) {
                    c.Description += 'Because of Task '"'+tsk.Subject+'"'\n';
```

```
            }
        }
        for(Event evt : my_events) {
            if(evt.WhoId == c.Id) {
                c.Description += 'Because of Event "'+evt.Subject+'"\n';
            }
        }
        for(Case cse : my_cases) {
            if(cse.ContactId == c.Id) {
                c.Description += 'Because of Case "'+cse.Subject+'"\n';
            }
        }
    }

    return contacts;
  }

}


@IsTest
public class ContactsTodayControllerTest {

  @IsTest
  public static void testGetContactsForToday() {

    Account acct = new Account(
        Name = 'Test Account'
    );
    insert acct;

    Contact c = new Contact(
        AccountId = acct.Id,
        FirstName = 'Test',
        LastName = 'Contact'
    );
    insert c;

    Task tsk = new Task(
        Subject = 'Test Task',
        WhoId = c.Id,
        Status = 'Not Started'
    );
    insert tsk;
```

```
        Event evt = new Event(
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(5),
            EndDateTime = Date.today().addDays(6)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(1, contacts.size());
        System.assert(contacts[0].Description.containsIgnoreCase(tsk.Subject));
        System.assert(contacts[0].Description.containsIgnoreCase(evt.Subject));
        System.assert(contacts[0].Description.containsIgnoreCase(cse.Subject));

    }

    @IsTest
    public static void testGetNoContactsForToday() {

        Account acct = new Account(
            Name = 'Test Account'
        );
        insert acct;

        Contact c = new Contact(
            AccountId = acct.Id,
            FirstName = 'Test',
            LastName = 'Contact'
        );
        insert c;

        Task tsk = new Task(
            Subject = 'Test Task',
            WhoId = c.Id,
            Status = 'Completed'
        );
        insert tsk;

        Event evt = new Event(
```

```
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(-6),
            EndDateTime = Date.today().addDays(-5)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id,
            Status = 'Closed'
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(0, contacts.size());

    }

}
```

ApexSpecialistSuperbadgePackage Code:-

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

```
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,
```

```
@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;
```

WarehouseCalloutService:-

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';

    //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(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
```

```
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        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":"55d66226726b
611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b61110
0aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        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);
    }
}
```

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());
        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 Builder SuperBadge:-

Important Pre-works before you start doing this Superbadge:

- Create a new Trailhead Playground for this superbadge. Your new org will have all the special data you need. (Be sure to create a Trailhead Playground, and not a regular Developer Edition org. Only Trailhead Playgrounds have the correct data for these challenges.) Using this org for any other reason might create problems when validating the challenges.
- Use Lightning Experience.
- 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.
- Don't use Workflow to solve any challenges.

These three are very important to avoid any error while doing challenges in the superbadge.

Challenge 1

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.

Assignment Rule

- Create new Assignment rule for this scenario(Do not use the standard rule).
- Make sure that you rule is Active before you validate this step.

Challenge 2

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() + 7 < 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 Shipping State/Province and Shipping Country

Don't forget replicate For Shipping State/Province and Shipping Country same as Billing State/Province and Billing Country validation which I have mentioned above.

Challenge 3

It can be done easily:

- 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

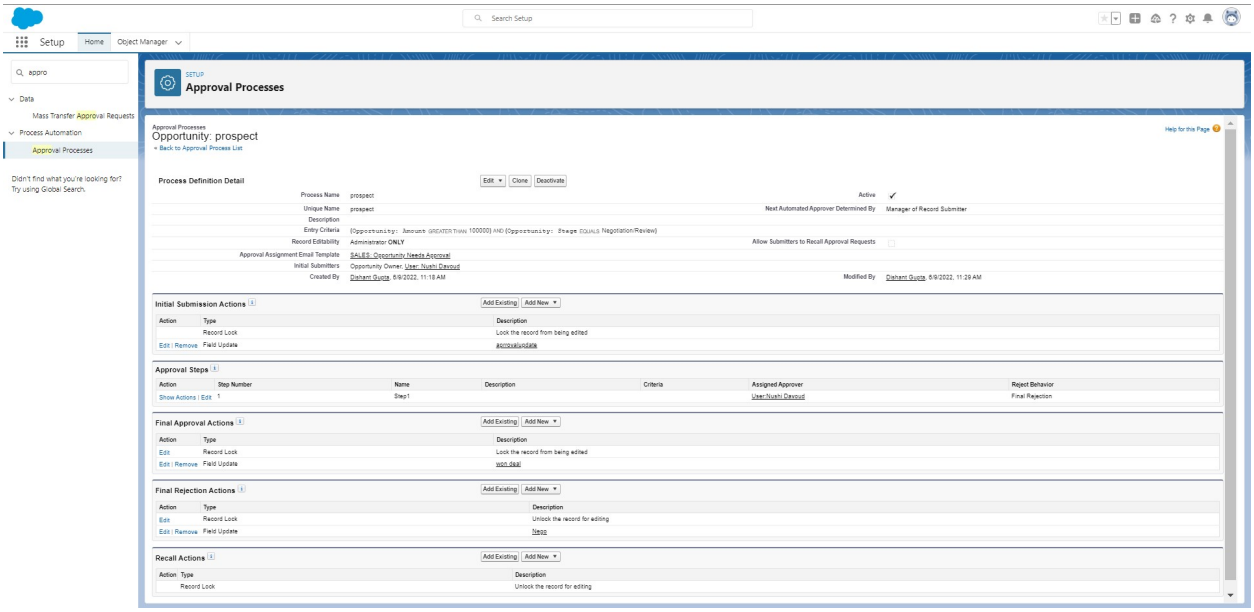
Challenge 4

- 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)

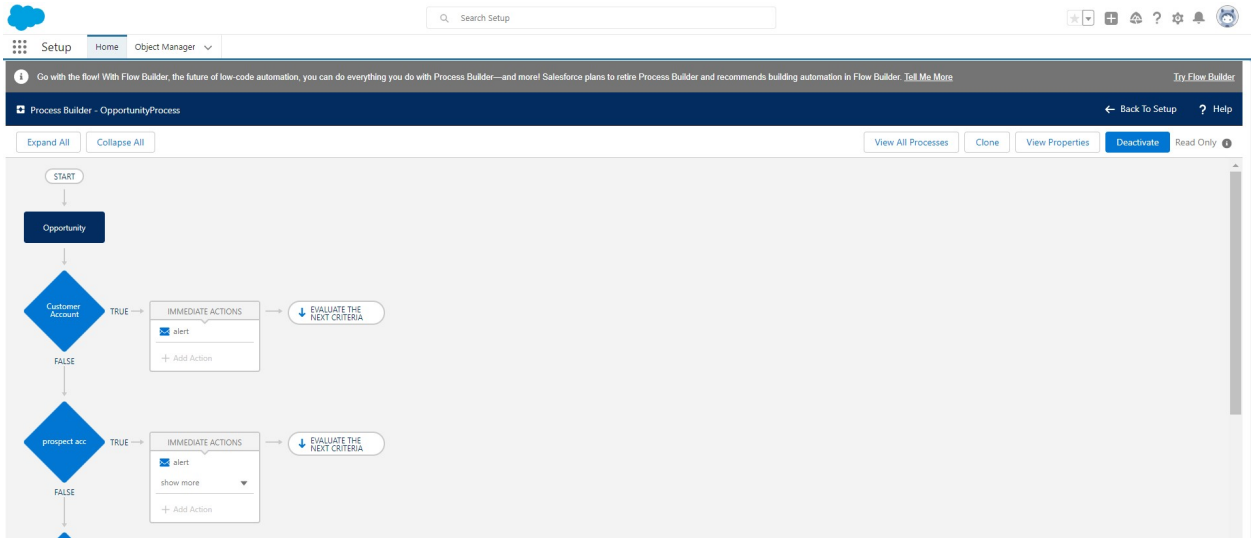
Challenge 5

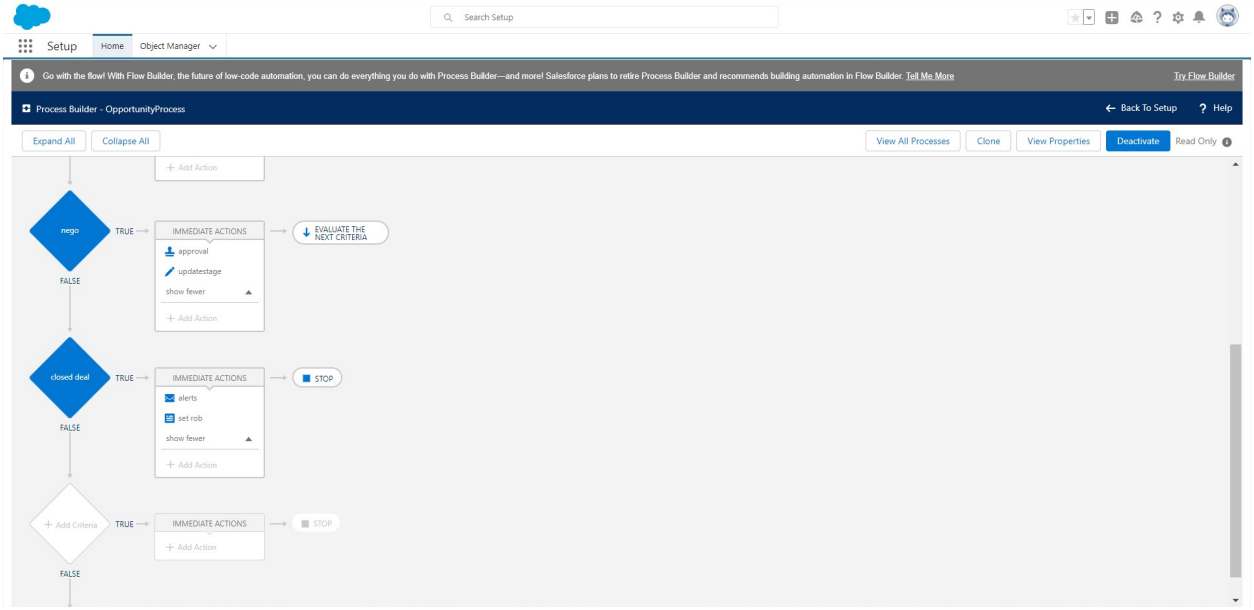
Approval Process Definition Detail: See the screenshot below for details



It’s time to create **Process Builder**.

Name: OpportunityProcess

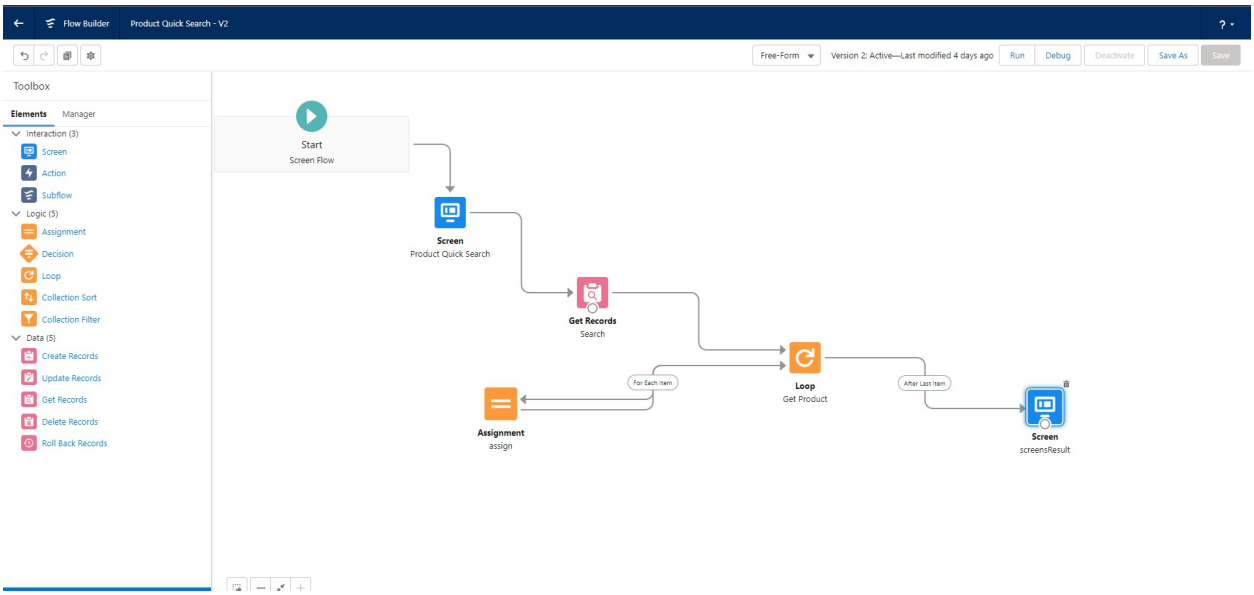




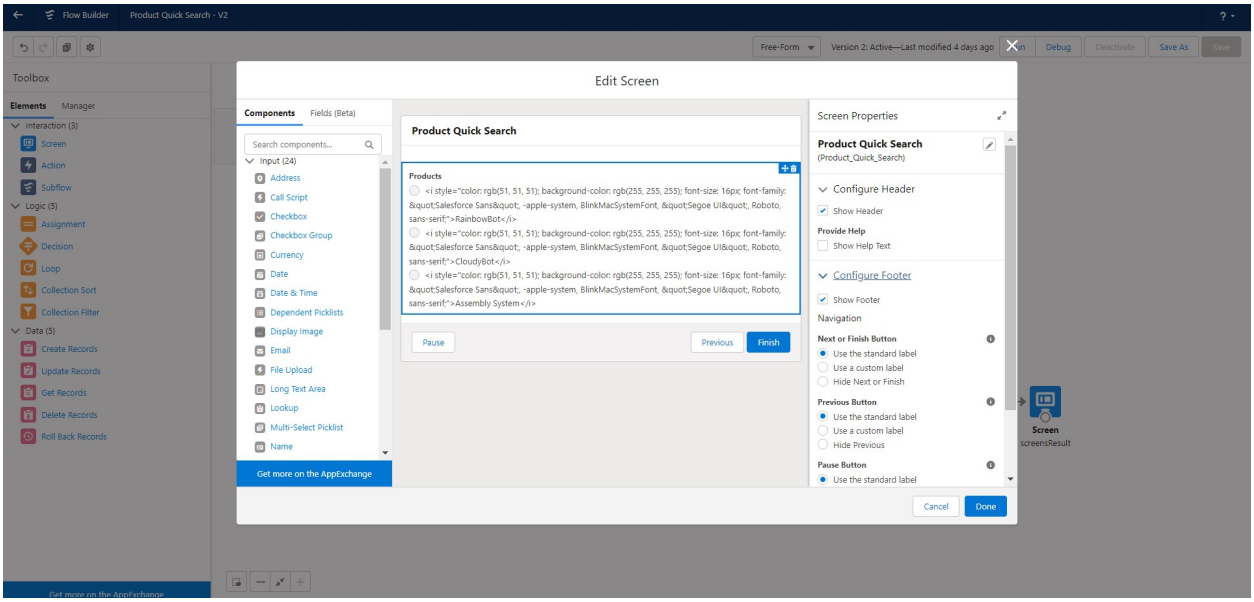
Note: If you have trouble in creating process builder, comment the errors you are getting, so that I will guide you to process it.

Challenge 6

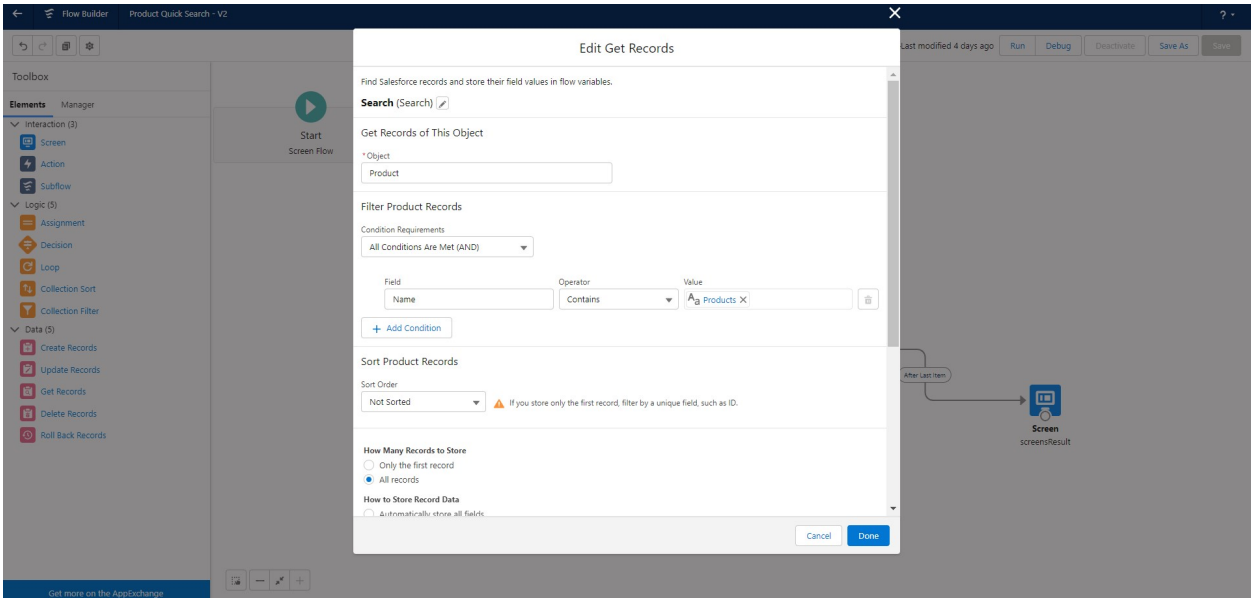
Create the flow to display products.

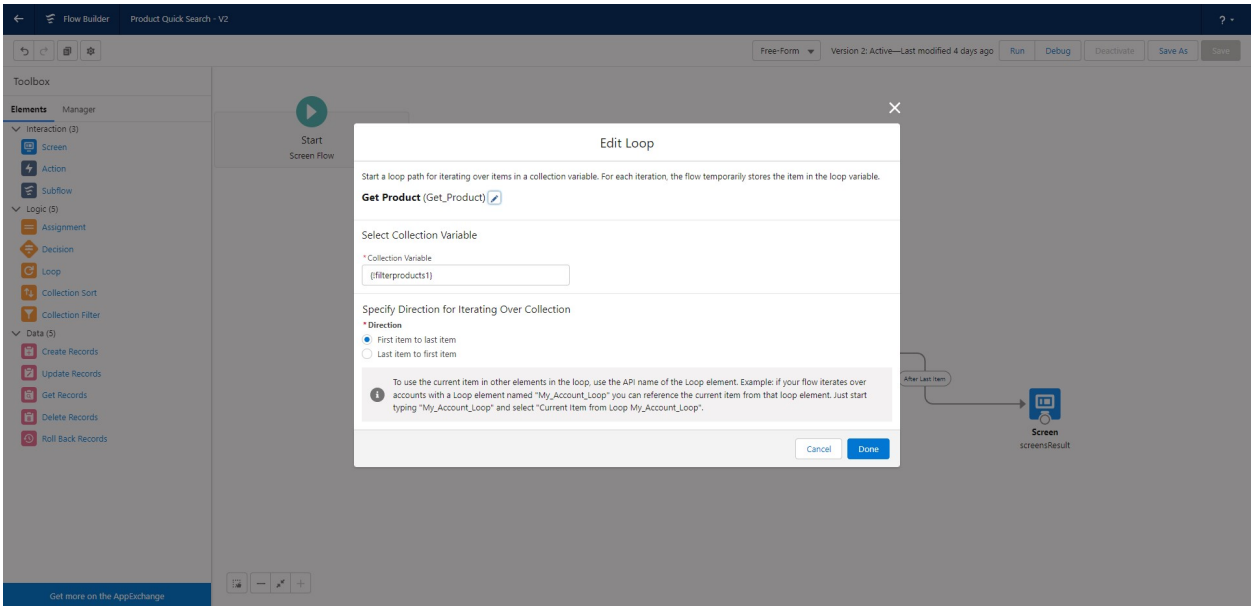


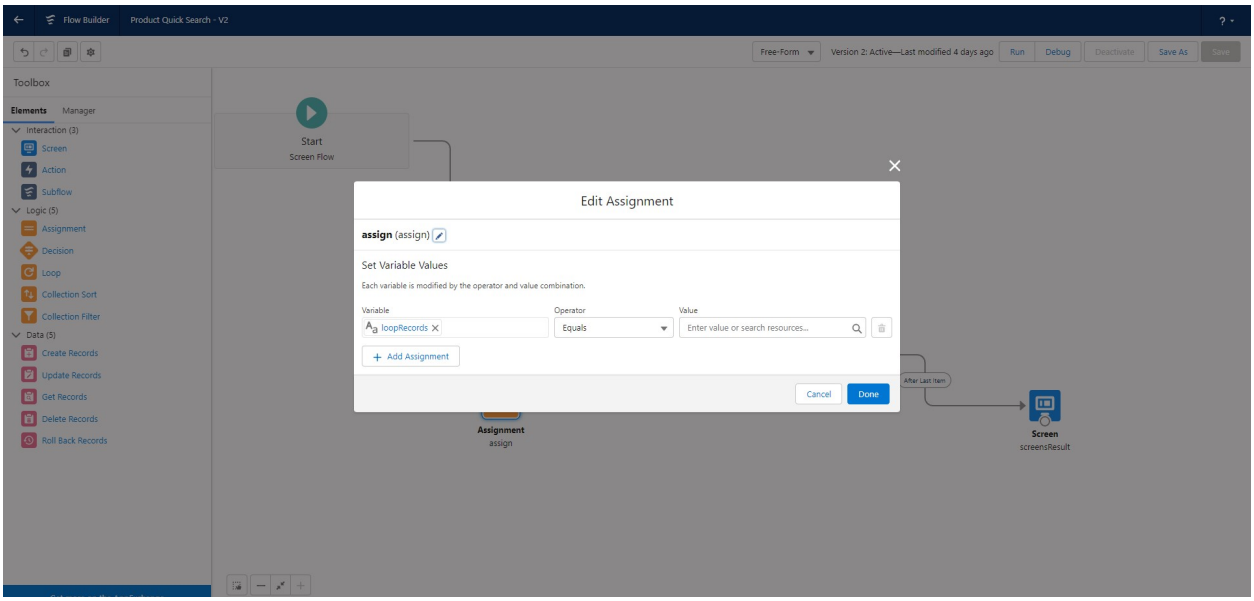
Screen (Product Type Search) Properties:

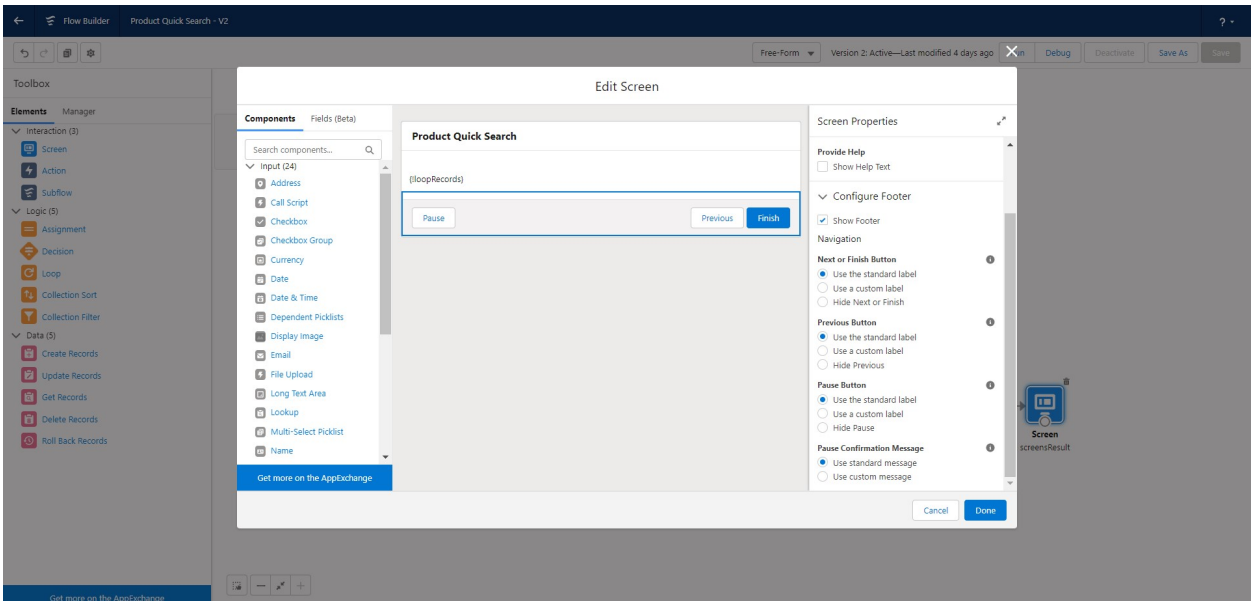


Get Records (Product Name Lookup) Properties:





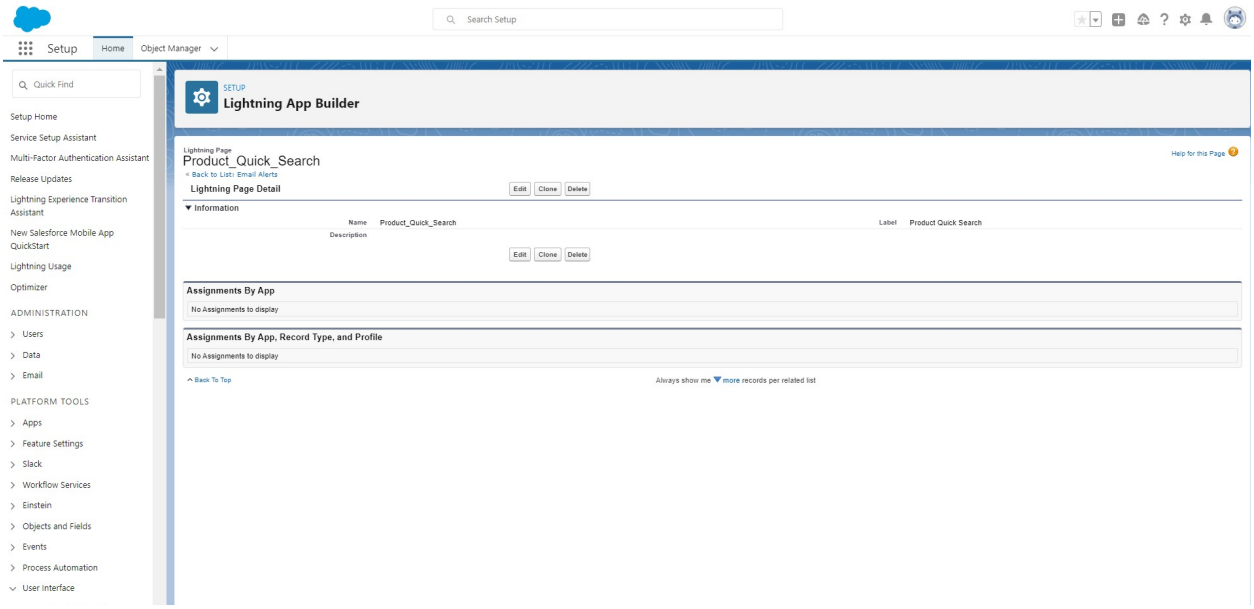




- Activate the flow
- Add the flow to the opportunity screen using app builder.

Create a Record Page on Opportunity Object:

Go to Lightning App Builder page and click new. Record Page Properties are as follows:-



- Add the component on newly created Opportunity Record Page.
- Please don't forgot to Activate the page.

Challenge 7

- 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

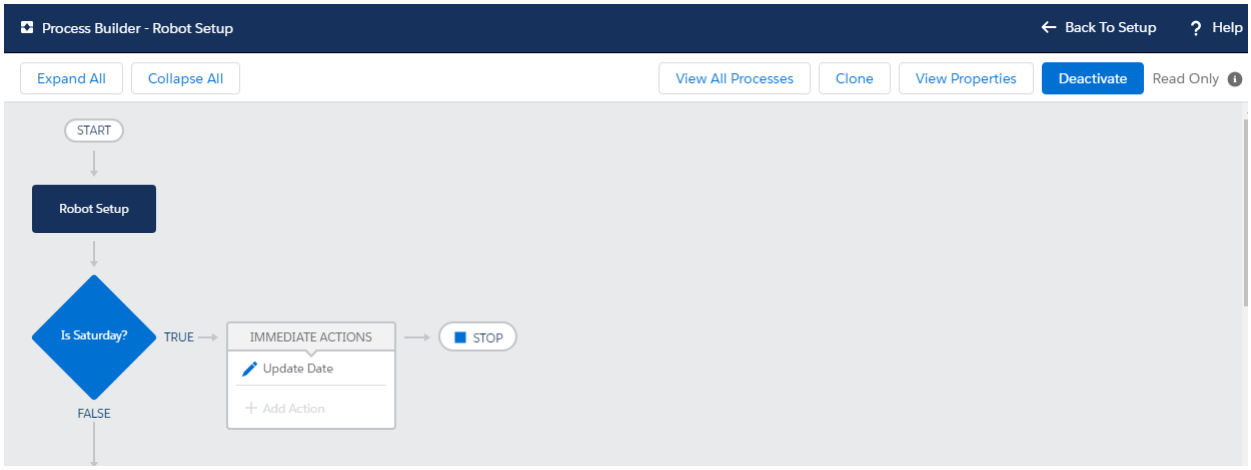
CASE(WEEKDAY(Date__c),
1, "Sunday",
2, "Monday",
3, "Tuesday",
4, "Wednesday",
5, "Thursday",
6, "Friday",
7, "Saturday",
Text(WEEKDAY(Date__c)))

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 and you are done!

Hooray.... Done the Superbadge Successfully..