Apex Specialist SuperBadge

Apex Triggers

AccountAddressTrigger:

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account: Trigger.New){
        if(account.Match_Billing_Address__c==True){
            account.ShippingPostalCode= account.BillingPostalCode;
        }
    }
}
```

ClosedOpportunityTrigger:

```
trigger ClosedOpportunityTrigger on Opportunity (before insert, after update) {
    List<Task> tasklist = new List<Task>();
    for(Opportunity opp: Trigger.New){
        if(opp.StageName=='Closed Won'){
            tasklist.add(new Task(Subject='Follow Up Test Task',WhatId=opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

Apex Testing

VerifyDate:

```
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
```

```
                return SetEndOfMonthDate(date1);
            }
        }

        private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }
        Date date30Days = date1.addDays(30);
            if( date2 >= date30Days ) { return false; }
            else { return true; }
        }

        private static Date SetEndOfMonthDate(Date date1) {
            Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
            Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
            return lastDay;
        }
}
```

TestVerifyDate :

```
@isTest
public class TestVerifyDate {
   @isTest static void test1(){
      Date d= VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('01/03/2022'));
      System.assertEquals(Date.parse('01/03/2022'),d);
   }
   @isTest static void test2(){
      Date d= VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
      System.assertEquals(Date.parse('01/31/2022'),d);
   }
}
```

RestrictContactByName :

```
trigger RestrictContactByName on Contact (before insert, before update) {

        For (Contact c : Trigger.New) {
            if(c.LastName == 'INVALIDNAME') {   //invalidname is invalid
                c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
            }
```

```
        }
}
```

TestRestrictContactByName :

```
@isTest
public class TestRestrictContactByName {
   @isTest public static void testContact(){
      Contact ct= new Contact();
      ct.LastName='INVALIDNAME';
      Database.SaveResult res= Database.insert(ct,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());
   }
}
```

RandomContactFactory :

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer num,String lastName){
      List<Contact> contactList = new List<Contact>();
      for (Integer i=1;i<=num;i++){
         Contact ct= new Contact(FirstName='Test '+i,LastName=lastName);
         contactList.add(ct);
      }
      return contactList;
   }
}
```

Asynchronous Apex

AccountProcessor :

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountId_lst) {
      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from contacts) from
```

```
account]);
        for(account a:account_lst_all) {
            account_cno.put(a.id,a.contacts.size()); //populate the map
        }

        List<account> account_lst = new List<account>(); // list of account that we will upsert

        for(Id accountId : accountId_lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c = account_cno.get(accountId);
                account_lst.add(acc);
            }
        }
        upsert account_lst;
    }
}
```

AccountProcessorTest :

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;
        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;
        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
            AccountProcessor.countContacts(acc_list);
```

```
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from account
where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }
}
```

LeadProcessor :

```
global class LeadProcessor implements Database.Batchable<sObject>, Database.Stateful {
    global Integer recordsProcessed = 0;

    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope){

        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {
                lead.LeadSource = 'Dreamforce';
                recordsProcessed = recordsProcessed + 1;
        }
        update leads;
    }

    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed.');
    }
}
```

LeadProcessorTest :

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
```

```
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }

    static testmethod void test() {
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
    }
}
```

AddPrimaryContact :

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT
200];
        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;
            lstOfConts.add(conInst);
        }
        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest :

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');
        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();
        System.assertEquals(50, [select count() from Contact]);
    }
}
```

DailyLeadProcessor :

```
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;
  }
    }
}
```

DailyLeadProcessorTest :

```
@isTest
public class DailyLeadProcessorTest {
   public static String CRON_EXP='0 0 0 15 4 ? 2033';
   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();
      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company '
+ i, Status = 'Open - Not Contacted');
         leads.add(lead);
      }
      insert leads;
      Test.startTest();
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());
      Test.stopTest();
   }
}
```

Apex Integration Services

AnimalLocator :

```
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);
        String strResp = '';
         system.debug('******response '+response.getStatusCode());
         system.debug('******response '+response.getBody());
      if (response.getStatusCode() == 200)
      {
         Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
```

```
        Map<string,object> animals = (map<string,object>) results.get('animal');
         System.debug('Received the following animals:' + animals );
         strResp = string.valueof(animals.get('name'));
         System.debug('strResp >>>>>>' + strResp );
       }
       return strResp ;
  }
}
```

AnimalLocatorTest :

```
@isTest
private class AnimalLocatorTest{
   @isTest static  void AnimalLocatorMock1() {
      Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());
      string result=AnimalLocator.getAnimalNameById(3);
      string expectedResult='chicken';
      System.assertEquals(result, expectedResult);
   }
}
```

AnimalLocatorMock :

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
       HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');
      response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
      response.setStatusCode(200);
      return response;
   }
}
```

ParkService :

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

ParkLocator :

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

ParkLocatorTest :

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

AccountManager :

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
    }
```

```
}
```

AccountManagerTest :

```
@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account  acc = AccountManager.getAccount();
      System.assert(acc != null);
   }
   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;
      Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
      Insert con;
      return acc.Id;
   }
}
```

Process Automation Specialist SuperBadge

Formula and Validations

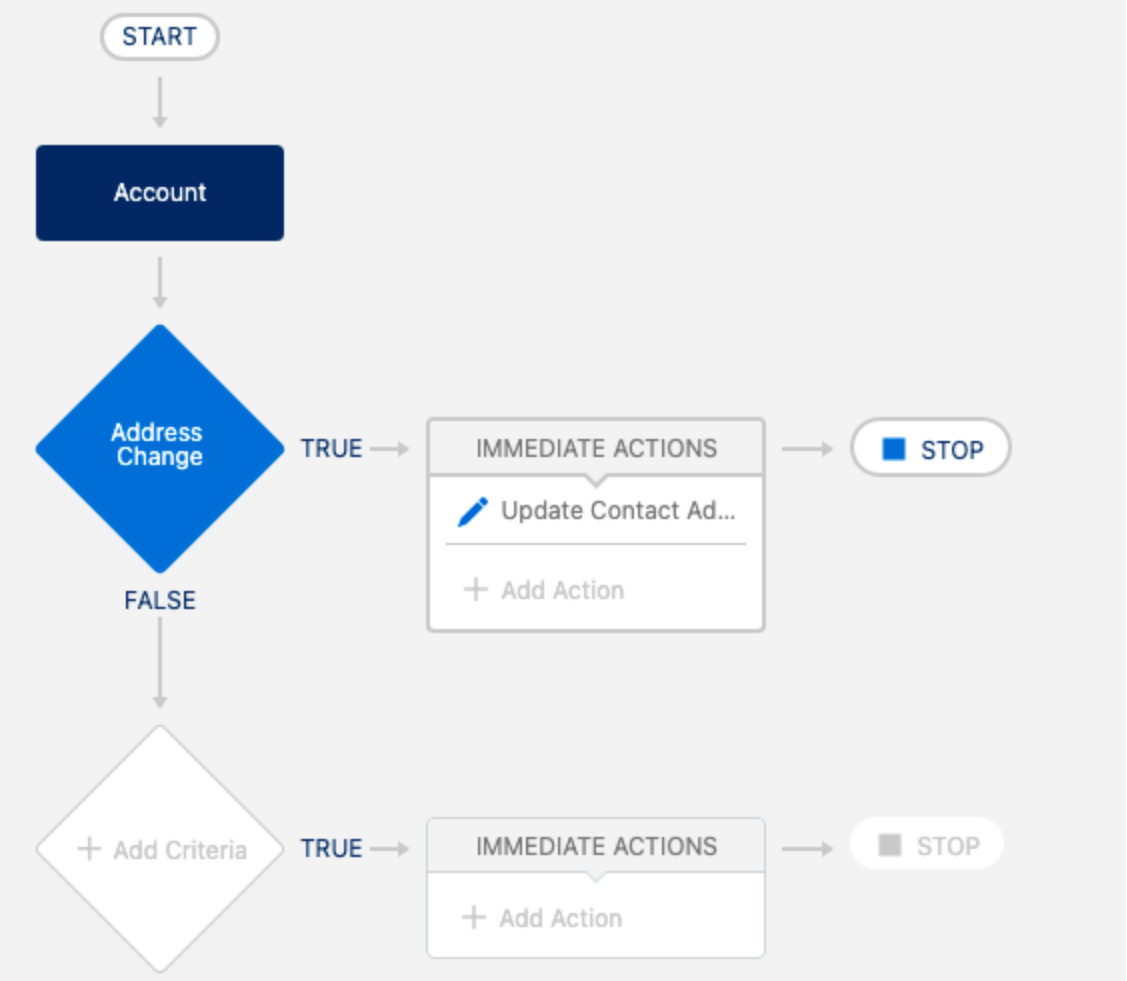
Creating a validation rule that displays an error message and prevents a user from creating or updating a contact if two conditions are both true.

Contact_must_be_in_Account_ZIP_Code :

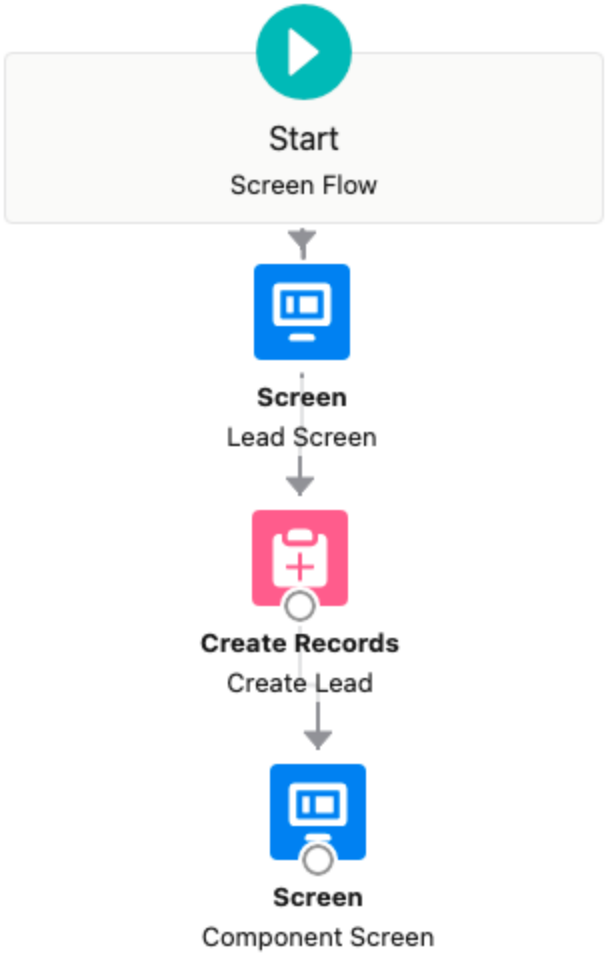
```
AND(NOT(ISBLANK( AccountId )), MailingPostalCode   <>  Account.ShippingPostalCode )
```

Salesforce Flow

Creating a process that updates child contacts' mailing addresses when the parent account's shipping address changes. If you use an existing playground to complete this challenge, deactivate any validation rules you created for the Contact or Account objects in the playground.



Building a flow that creates a lead with user-entered information and uploads a related file for the lead. Then add the flow to a Home page:


Start
Screen Flow
Screen
Lead Screen
Create Records
Create Lead
Screen
Component Screen