CHINNAM RAVITEJA
https://trailblazer.me/id/raviteja2002

Salesforce Developer Catalyst

Apex Specialist Superbadge

Automated Record Creation

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);


        }
      }
    }

    if (!validIds.isEmpty()){
      List<Case> newCases = new List<Case>();
      Map<Id,Case> closedCasesM = 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>();
      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'));
    }

      for(Case cc : closedCasesM.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 (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> clonedWPs = new List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
          for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c wpClone = wp.clone();
            wpClone.Maintenance_Request__c = nc.Id;
            ClonedWPs.add(wpClone);

          }
        }
        insert ClonedWPs;
      }
    }
}
```

MaitenanceRequest.apxt :

```
trigger MaintenanceRequest on Case (before update, after update) {

  if(Trigger.isUpdate && Trigger.isAfter){

    MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

  }

}
```

Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :

```
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();
  }

}
```

Schedule synchronization using Apex code

WarehouseSyncShedule.apxc :

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

Test automation logic

MaintenanceRequestHelperTest.apxc :-

```
@istest
public with sharing class MaintenanceRequestHelperTest {

  private static final string STATUS_NEW = 'New';
  private static final string WORKING = 'Working';
  private static final string CLOSED = 'Closed';
  private static final string REPAIR = 'Repair';
  private static final string REQUEST_ORIGIN = 'Web';
  private static final string REQUEST_TYPE = 'Routine Maintenance';
  private static final string REQUEST_SUBJECT = 'Testing subject';

  PRIVATE STATIC Vehicle__c createVehicle(){
    Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
    return Vehicle;
  }

  PRIVATE STATIC Product2 createEq(){
    product2 equipment = new product2(name = 'SuperEquipment',
                        lifespan_months__C = 10,
                        maintenance_cycle__C = 10,
                        replacement_part__c = true);
    return equipment;
  }

  PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cs = new case(Type=REPAIR,
              Status=STATUS_NEW,
              Origin=REQUEST_ORIGIN,
              Subject=REQUEST_SUBJECT,
```

```
            Equipment__c=equipmentId,
            Vehicle__c=vehicleId);
    return cs;
  }

  PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
    Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c =
equipmentId,
                                        Maintenance_Request__c = requestId);
    return wp;
  }


  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c, Date_Due__c
          from case
          where status =:STATUS_NEW];

    Equipment_Maintenance_Item__c workPart = [select id
                        from Equipment_Maintenance_Item__c
                        where Maintenance_Request__c =:newReq.Id];

    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
  }
```

```
    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

        list<case> allRequest = [select id
                                 from case];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
          vehicleList.add(createVehicle());
           equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
           requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
```

```
    }
    insert requestList;

    for(integer i = 0; i < 300; i++){
      workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
    }
    insert workPartList;

    test.startTest();
    for(case req : requestList){
      req.Status = CLOSED;
      oldRequestIds.add(req.Id);
    }
    update requestList;
    test.stopTest();

    list<case> allRequests = [select id
                  from case
                  where status =: STATUS_NEW];

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

    system.assert(allRequests.size() == 300);
  }
}
```

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);


        }
      }
    }

    if (!validIds.isEmpty()){
      List<Case> newCases = new List<Case>();
      Map<Id,Case> closedCasesM = 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>();
        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'));
      }

        for(Case cc : closedCasesM.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 (maintenanceCycles.containskey(cc.Id)){
            nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
          }

          newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
          for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c wpClone = wp.clone();
            wpClone.Maintenance_Request__c = nc.Id;
            ClonedWPs.add(wpClone);

          }
        }
        insert ClonedWPs;
      }
    }
}
```

MaintenanceRequest.apxt :

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

```
    }
}
```

Test callout logic

WarehouseCalloutService.apxc :

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

    //@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());

            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 = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the warehouse one');
                System.debug(warehouseEq);
            }

        }
    }
}
```

WarehouseCalloutServiceTest.apxc :

```
@isTest
private class WarehouseCalloutServiceTest {
  @isTest
  static void testWareHouseCallout(){
    Test.startTest();
    // implement mock callout test here
    Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.runWarehouseEquipmentSync();
    Test.stopTest();
    System.assertEquals(1, [SELECT count() FROM Product2]);
  }
}
```

WarehouseCalloutServiceMock.apxc :

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
  // implement http mock callout
  global static HttpResponse respond(HttpRequest request){

    System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
    System.assertEquals('GET', request.getMethod());

    // Create a fake response
    HttpResponse response = new HttpResponse();
    response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generat
or 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
    response.setStatusCode(200);
    return response;
  }
}
```

Test scheduling logic

WarehouseSyncSchedule.apxc :

```
global class WarehouseSyncSchedule implements Schedulable {
  global void execute(SchedulableContext ctx) {

    WarehouseCalloutService.runWarehouseEquipmentSync();
  }
}
```

WarehouseSyncScheduleTest.apxc :

```
@isTest
public class WarehouseSyncScheduleTest {
```

```
  @isTest static void WarehousescheduleTest(){
    String scheduleTime = '00 00 01 * * ?';
    Test.startTest();
    Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
    String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
    Test.stopTest();
    //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX
systems.
    // This object is available in API version 17.0 and later.
    CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
    System.assertEquals(jobID, a.Id,'Schedule ');
```

Apex Triggers

Get Started with Apex Triggers

AccountAddressTrigger

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

Bulk Apex Triggers

ClosedOpportunityTrigger

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

    List<Opportunity> relatedOpps = [SELECT Id,OwnerId,StageName FROM Opportunity  WHERE id in
:Trigger.New];

   List<Task> tasks = new List<Task>();
   for(Opportunity opp : relatedOpps)
```

```
    {
      if(opp.StageName == 'Closed Won')
       {
          Task tsk = new Task(whatID = Opp.ID, Ownerid = Opp.OwnerId, Subject='Follow Up   Test Task');
tasks.add(tsk);

       }
      }
  insert tasks;

}
```

Apex Testing

Get Started with Apex Unit Tests

VerifyDate

```
public class VerifyDate {

        //method to handle potential checks against two dates
        public static Date CheckDates(Date date1, Date date2) {
                //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }

        //method to check if date2 is within the next 30 days of date1
        private static Boolean DateWithin30Days(Date date1, Date date2) {
                //check for date2 being in the past
           if( date2 < date1) { return false; }

           //check that date2 is within (>=) 30 days of date1
           Date date30Days = date1.addDays(30); //create a date 30 days away from date1
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }


        //method to return the end of the month of a given date
        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
class TestVerifyDate {

   static testMethod void TestVerifyDate() {
    VerifyDate.CheckDates(System.today(),System.today()+10);
     VerifyDate.CheckDates(System.today(),System.today()+78);
   }
}
```

Test Apex Triggers

RestrictContactByName

```
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');
                }

        }




}
```

TestRestrictContactByName

```
@istest
private class TestRestrictContactByName {
   @istest static void testname(){
     contact c = new contact(firstname='Satya',lastname='INVALIDNAME');
     test.startTest();
     database.SaveResult result = database.insert(c,false);
     test.stopTest();
     system.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests

RandomContactFactory

```
public class RandomContactFactory {

  Public Static List<Contact> generateRandomContacts(integer noOfContact, String lastName)
  {
     List<Contact> con=New list<Contact>();
     for(Integer i=0;i<noOfContact;i++)
     {
        Contact c = new Contact(FirstName='Ank' + i,LastName=lastName);
        Con.add(c);
      }



     // insert con;

      Return con;
  }


}
```

Asynchronous Apex

Use Future Methods

AccountProcessor

```
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();
       system.debug(' acc.Number_of_Contacts__c ');
    }
    update lstAccount;
 }
}
```

AccountProcessorTest

```
@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> setAccId = new Set<ID>();
    setAccId.add(a.id);

    Test.startTest();
      AccountProcessor.countContacts(setAccId);
    Test.stopTest();

    Account ACC = [select Number_of_Contacts__c from Account where id = :a.id];
    System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
 }

}
```

Use Batch Apex

LeadProcessor

```
public class LeadProcessor  implements
  Database.Batchable<sObject>, Database.Stateful {
  // instance member to retain state across transactions
  public Integer recordsProcessed = 0;
  public Database.QueryLocator start(Database.BatchableContext bc) {
    return Database.getQueryLocator('SELECT ID, LeadSource from Lead');
  }
public void execute(Database.BatchableContext bc, List<Lead> scope){
    // process each batch of records
   //  List<Lead> lList = new List<Lead>();
    for (Lead lList : scope) {
         lList.leadsource='Dreamforce';



        }
```

```
    update scope;
  }
  public void finish(Database.BatchableContext bc){

  }
}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest {
@testSetup
  static void setup() {
    List<Lead> llist = new List<Lead>();
      // insert 10 accounts
    for (Integer i=0;i<200;i++) {
      llist.add(new Lead(FirstName='Lead '+i,LastName='last', Company ='demo'+i));
    }
    insert llist;
    // find the account just inserted. add contact for each

  }
  @isTest static void test() {
    Test.startTest();
    LeadProcessor lpt = new LeadProcessor();
    Id batchId = Database.executeBatch(lpt);
    Test.stopTest();
    // after the testing stops, assert records were updated properly
    System.assertEquals(200, [select count() from lead where Leadsource = 'Dreamforce']);
  }
}
```

Control Processes with Queueable Apex

AddPrimaryContact

```
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<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM
ACCOUNT WHERE BillingState = :state LIMIT 200];
     List<Contact> lstContact = new List<Contact>();
     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;
     }

  }

}
```

AddPrimaryContactTest

```
@isTest

public class AddPrimaryContactTest
{
   @isTest static void TestList()
   {
     List<Account> Teste = new List <Account>();
     for(Integer i=0;i<50;i++)
     {
       Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
     }
     for(Integer j=0;j<50;j++)
     {
       Teste.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

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
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
          for (Integer i = 0; i < 200; i++) {
                        lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));
                }
                insert lList;

                Test.startTest();
                String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
        }
}
```

Apex Integration Services

Apex REST Callouts

AnimalLocator

```
public with sharing class AnimalLocator {

  public static String getAnimalNameById(Integer animalNameId) {
    String animalName = '';
    //New Http 'GET' Request
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/:id');
    request.setHeader('Content-Type', 'application/json;charset=UTF-8');
    request.setMethod('GET');
    //Get response
    HttpResponse response = Http.send(request);
    //Parse JSON from the response body
    JSONParser parser = JSON.createParser(response.getBody());
    while (parser.nextToken() != null) {
      // Read entire JSON object
      if (parser.getCurrentToken() == JSONToken.START_OBJECT) {
        AnimalLocator.AnimalList animalList = (AnimalLocator.AnimalList)
parser.readValueAs(AnimalLocator.AnimalList.class);
        System.debug(animalList.animal.size());
        //Sort through the list of animals to find one with the matching ID
        //Set the animal name
        for (Integer i = 0; i < animalList.animal.size() ; i++) {
          if (animalList.animal[i].id == animalNameId){
            animalName = animalList.animal[i].name;
            break;
          } else{
            animalName = 'Could not find an Animal with a matching ID';
          }
        }

      }
    }
    return animalName;
  }

  public class AnimalList {
    public List<animal> animal; //This has to be the same name thats in the JSON file.
  }

  //animal Object Wrapper
  public class animal {
    public Integer id;
    public String name;
    public String eats;
    public String says;
  }

}
```

AnimalLocatorTest

```
@isTest
public with sharing class AnimalLocatorTest {
  @isTest
   static void testGetCallout() {
     Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
     String result = AnimalLocator.getAnimalNameById(1);
     String expectedResult = 'Chicken';
     System.assertEquals(result,expectedResult);
     result = AnimalLocator.getAnimalNameById(4);
     expectedResult = 'Could not find an Animal with a matching ID';
     System.assertEquals(result,expectedResult);
  }
}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock{
  global HttpResponse respond(HttpRequest request){
     //Create Fake Response
     HttpResponse response = new HttpResponse();
     response.setHeader('Content-Type', 'application/json;charset=UTF-8');
     response.setStatusCode(200);
     response.setBody('
{"animal":[{"id":1,"name":"Chicken","eats":"Grain","says":"Cluck"},{"id":2,"name":"Dog","eats":"Chicken","says":"
Woof"}]} ');
     return response;
  }
}
```

Apex SOAP Callouts

ParkLocator

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

ParkLocatorTest

```
@isTest
private class ParkLocatorTest {
  @isTest 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);
    }
}
```

ParkServiceMock

```
@isTest
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

AccountManager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {


  @HttpGet
  global static account getAccount() {
```

```
    RestRequest request = RestContext.request;

    String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
      request.requestURI.lastIndexOf('/'));
    List<Account> a = [select id, name, (select id, name from contacts) from account where id = :accountId];
    List<contact> co = [select id, name from contact where account.id = :accountId];
    system.debug('** a[0]= '+ a[0]);
    return a[0];

  }

}
```

AccountManagerTest

```
@istest
public class AccountManagerTest {
@istest static void testGetContactsByAccountId() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+ recordId+'/Contacts';
request.httpMethod = 'GET';
RestContext.request = request;

Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount!= null);
System.assertEquals('Test record', thisAccount.Name);
}

// Helper method
static Id createTestRecord() {

// Create test record
Account accountTest = new Account(
Name='Test record');
insert accountTest;
Contact contactTest = new Contact(
FirstName='John',
LastName='Doe',
AccountId=accountTest.Id
);
return accountTest.Id;
}
}
```

PROCESS AUTOMATION SPECIALIST SUPERBADGE

Automata leads

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))))

Automate accounts

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 formula1
ISCHANGED(Name) && (OR(ISPICKVAL(Type ,'Customer - Direct') ,ISPICKVAL(Type ,'Customer - Channel')))

Automata steps

formula

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