APEX SPECIALIST SUPERBADGE

Challenge 1:

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));
                }

                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);
    }
}
```

Challenge 2:

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';

    @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 = (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();
    }

}
```

Challenge 3:

Schedule synchronization using Apex code

WarehouseSyncShedule.apxc :-

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

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

Challenge 4:

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);
    }
}
```

Challenge 5:

Test callout logic

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge
apex.herokuapp.com/equipment';

    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();
    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 {
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        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"}]')
;         response.setStatusCode(200);
        return response;
    }
}
```

Challenge 6:

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();
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

APEX TRIGGERS

Get Started With Apex Triggers:

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account : Trigger.new)
    {
        if((account.Match_Billing_Address__c==true)&&(account.BillingPostalCode !=
NULL))
            account.ShippingPostalCode = account.BillingPostalCode;
    }

}
```

Bulk Apex Triggers:

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after 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

Get Started with Apex Unit Test:

VerifyDate.apxc

```
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.apxc

```
@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);
   }


}
```

Test Apex Triggers:

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (before insert, before update) {
   For (Contact c : Trigger.New) {
if(c.LastName == 'INVALIDNAME') {
c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
}
}
}
```

TestRestrictContactByName.apxc

```
@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());
   }

}
```

Create Test data for Apex Tests:

RandomContactFactory.apxc

```
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

Use Future Method:

AccountProcessor.apxc

```
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds){

List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id from Contacts)
from Account where Id in :accountIds];

        for(Account acc : accList){

            acc.Number_Of_Contacts__c = acc.Contacts.size();
         }

        update accList;
    }
}
```

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
```

```
        Account a= new Account();
        a.Name='Test Account';
        insert a;

        Contact con= new Contact();
        con.FirstName='Vyshnavi';
        con.LastName = 'Priya';
        con.AccountId=a.Id;

        insert con;

        List<Id> accListId =new List<Id>();
        accListId.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(accListId);
        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.apxc

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

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

    global void execute (Database.BatchableContext bc, List<lead> L_list){
```

```
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource='Dreamforce';
            L_list_new.add(L);
            count+=1;
        }
        update L_list_new;
    }

    global void finish(Database.BatchableContext bc){
        system.debug('count='+count);
    }

}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list=new List<lead>();
        for(Integer i=0;i<200;i++){
            Lead L= new lead();
            L.LastName='name'+i;
            L.Company='Company';
            L.Status='Random Status';
            L_list.add(L);
        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp= new LeadProcessor();
```

```
        Id batchId= Database.executeBatch(lp);
        Test.stopTest();
    }

}
```

Control Processes with Queueable Apex:

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {

    private Contact con;
    private String state;

    public AddPrimaryContact(Contact con,String state){
        this.con=con;
        this.state=state;
    }

    public void execute(QueueableContext context){
        List<Account> accounts=[Select Id,Name,(Select FirstName,LastName,Id from
contacts)
                                from Account where BillingState=: state Limit 200];
        List<Contact> primaryContacts=new List<Contact>();

        for(Account acc:accounts){
            contact c= con.clone();
            c.AccountId=acc.Id;
            primaryContacts.add(c);
        }
        if(primaryContacts.size()>0){
            insert primaryContacts;
        }
    }
```

```
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts=new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account'+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));
        }
        insert testAccounts;

        Contact testContact =new Contact(FirstName='Vyshnavi',LastName='Priya');
        insert testContact;

        AddPrimaryContact addit = new addPrimaryContact(testContact,'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();
        System.assertEquals(50,[Select count() from Contact where accountId in (Select Id
from Account where BillingState='CA')]);
    }


}
```

Schedule Jobs Using the Apex Scheduler:

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements schedulable{

   public void execute(schedulableContext sc) {
      List<lead> l_lst_new = new List<lead>();
      List<lead> l_lst = new List<lead>([select id, leadsource from lead where leadsource
= null]);
      for(lead l : l_lst) {
         l.leadsource = 'Dreamforce';
         l_lst_new.add(l);
      }
      update l_lst_new;
   }

}
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {

   @isTest
   public static void testing() {

      List<lead> l_lst = new List<lead>();
      for(Integer i=0;i<200;i++) {
         lead l = new lead();
         l.lastname = 'lastname'+i;
         l.Company = 'company'+i;
         l_lst.add(l);
```

```
        }
        insert l_lst;

        Test.startTest();
        DailyLeadProcessor dlp = new DailyLeadProcessor ();
        String jobId = System.Schedule('dailyleadprocessing','0 0 0 1 12 ? *',dlp);
        Test.stopTest();

        List<lead> l_lst_chk = new List<lead>([select id,leadsource from lead where
leadsource != 'Dreamforce']);
        System.assertequals(0,l_lst_chk.size());
    }

}
```

Apex Integration Services

Apex REST Callouts:

AnimalLocator.apxc

```
public class AnimalLocator {

    public static String getAnimalNameById (Integer id) {
        String AnimalName = '';
        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);
        if (response.getStatusCode() == 200) {
            Map<String,Object> results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String, Object>) results.get('animal');
            animalName = (String) animal.get('name');
```

```
        }
        return animalName;
    }
}
```

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest {
@isTest static  void testGet() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String result = AnimalLocator.getAnimalNameById (7);
        System.assertNotEquals(null,result, 'The callout returned a null response.');
        System.assertEquals('panda', result,
                        'The animal name should be \'panda\");          }    }
```

Apex SOAP Callouts:

ParkService.apxc

```
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.apxc

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

ParkLocatorTest.apxc

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

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

Apex Web Services:

AccountManager.apxc

```
@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.apxc

```
@Istest(SeeAllData=true)
public class AccountManagerTest {
  @isTest static void testGetAccount() {
    Id recordId = createTestRecord();
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://resourceful-badger-76636-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts
'        + recordId;
    request.httpMethod = 'GET';
    RestContext.request = request;
    Account thisAcc = AccountManager.getAccount();
    System.assert(thisAcc != null);
    System.assertEquals('Test record', thisAcc.Name);
}
static Id createTestRecord()
{
```

```
    Account accTest = new Account(
        Name='Test record');
    insert accTest;
    return accTest.Id;
}


}
```