Salesforce Virtual Internship Program

Developer Console Codes

-Pasham Akshatha Sai

APEX SPECIALIST SUPERBADGE CODES

MaintenanceRequestHelper Class

```
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;
      }
   }
}
```

MaintenanceRequest Trigger

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

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

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

WarehouseCalloutService Class

```
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 = 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();
}

}
```

WarehouseSyncSchedule Class

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

MaintenanceRequestHelperTest Class

```
@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 Class

```
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 Trigger

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

WarehouseCalloutService Class

```
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 = 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 Class

```
@isTest
```

```
private class WarehouseCalloutServiceTest {
@isTest
static void WarehouseCalloutServiceTest(){
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 Class

```
@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,"qua
ntity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
response.setStatusCode(200);
return response;
}
}
```

WarehouseSyncSchedule Class

```
global class WarehouseSyncSchedule implements Schedulable { global void
execute(SchedulableContext ctx) {
WarehouseCalloutService.runWarehouseEquipmentSync(); } }
```

WarehouseSyncScheduleTest Class

```
@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 ');


}
}
```

AnimalLocator Class

```
public class AnimalLocator {

    public static String getAnimalNameById(Integer i){

        Http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);

        request.setMethod('GET');

        HttpResponse response = http.send(request);



        //If the request is successful,parse the JSON response

        Map<String, Object> result = (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());

        Map<String, Object> animal = (Map<String, Object>)result.get('animal');

        System.debug('name: '+string.valueOf(animal.get('name')));

        return string.valueOf(animal.get('name'));

    }
```

}

SELF-LEARNING MODULES:-

APEX TRIGGERS

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account(before insert, before update){
//Get the List of accounts
 //List<Account> newAccts = new List<Account>(
//[SELECT
Id,Match_Billing_Address__c,BillingPostalCode,ShippingPostalCode
FROM Account WHERE Id IN :newAccts]);

    for(Account alice : Trigger.New) {
        if (alice.Match_Billing_Address__c == true) {
           alice.ShippingPostalCode = alice.BillingPostalCode;

        }
    }
```

ClosedOpportunityTrigger

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

   //Grab the Opportunity Id's from Opps that are
Closed Won from the Context Variable and store
them in opp
for(Opportunity opp : [SELECT Id FROM Opportunity
WHERE StageName = 'Closed Won' IN
:Trigger.New]){

//Create a Follow Up Task against Id's that are stored
in the variable opp

newTask.add(new Task(Subject = 'Follow Up Test
Task',
Priority = 'High',
WhatId = opp.Id));

//Insert new Tasks
{insert newTask;
```

```
    }
    }
    }
```

APEX TESTING

VerifyDate Class

```
public class VerifyDate {
    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 Class

```
@isTest
    private class TestVerifyDate {
```

```
    @isTest static void testCheckDates() {
        Date test_date1 =
    VerifyDate.CheckDates(Date.newInstance(2018, 3, 19),
    System.today());
        Date test_date2 =
    VerifyDate.CheckDates(Date.newInstance(2018, 3, 19),
    System.today() + 100);
        Date test_date3 = VerifyDate.CheckDates(System.today(),
    System.today()-1);
    }
    }
```

RestrictContactByName Trigger

```
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 Class

```
@isTest
    public class TestRestrictContactByName{
    @isTest static void testRestrictContactByName () {
        Contact c = new Contact(LastName='INVALIDNAME');
        try{
          insert c;
        }
        catch(DMLException e){
              System.assert(e.getMessage().contains('The Last Name
    "'+c.LastName+'" is not allowed for DML'));
          }
        }
```

```
        }
```

RandomContactFactory Class

```
public class RandomContactFactory {
  public static List<Contact> generateRandomContacts(Integer
numOfContacts, String lName) {
      List<Contact> cList = new List<Contact>();
      for(Integer i=0; i<numOfContacts; i++) {
         Contact c = new Contact(Firstname = 'Test' + i, Lastname =
lName);
         conList.add(c);
      }
      return cList;
   }

}
```

ASYNCHRONOUS APEX

AccountProcessor Class

```
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 Class

```
@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 Class

```
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_lst) {
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
            count+=1;
        }
        update l_lst_new;
    }

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

LeadProcessorTest Class

```
@isTest
        public class LeadProcessorTest {

            @isTest
            public static void testit() {
                List<lead> l_lst = new List<lead>();
                for (Integer i = 0; i<200; i++) {
                    Lead l = new lead();
                    l.LastName = 'name'+i;
                    l.company = 'company';
                    l.Status =  'somestatus';
                    l_lst.add(l);
                }
                insert l_lst;

                test.startTest();

                Leadprocessor lp = new Leadprocessor();
                Id batchId = Database.executeBatch(lp);
                Test.stopTest();

            }

        }
```

AddPrimaryContact Class

```
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 Class

```
@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 Class

```
public without sharing class DailyLeadProcessor implements Schedulable {
 public void execute(SchedulableContext ctx) {
        //System.debug('Context ' + ctx.getTriggerId()); // Returns the ID of
the CronTrigger scheduled job

                    // Get 200 Lead records and modify the LeadSource field
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = null LIMIT 200];
        for ( Lead l : leads) {
           l.LeadSource = 'Dreamforce';
        }

                    // Update the modified records
        update leads;
    }
}
```

DailyLeadProcessorTest Class

```
  @isTest
            private class DailyLeadProcessorTest {

               private static String CRON_EXP = '0 0 0 ? * * *'; // Midnight every
            day

               @isTest
               private static void testSchedulableClass() {
```

```
        // Load test data
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<500; i++) {
            if ( i < 250 ) {
                leads.add(new Lead(LastName='Connock',
Company='Salesforce'));
            } else {
                leads.add(new Lead(LastName='Connock',
Company='Salesforce', LeadSource='Other'));
            }
        }
        insert leads;

        // Perform the test
        Test.startTest();
        String jobId = System.schedule('Process Leads', CRON_EXP,
new DailyLeadProcessor());
        Test.stopTest();

        // Check the result
        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM
Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At
least 1 record not updated correctly');

        // Check the scheduled time
        List<CronTrigger> cts = [SELECT Id, TimesTriggered,
NextFireTime FROM CronTrigger WHERE Id = :jobId];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);

        // Not sure this works for all timezones
                //Datetime midnight =
Datetime.newInstance(Date.today(), Time.newInstance(0,0,0,0));
        //System.assertEquals(midnight.addHours(24),
cts[0].NextFireTime, 'ERROR: Not scheduled for Midnight local time');

    }
}
```

APEX INTEGRATION MODULE

AnimalLocatorTest Class

```
@isTest
private class AnimalLocatorTest {

    @isTest
    static void animalLocatorTest1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }
}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

   global HttpResponse respond(HttpRequest request){
       HttpResponse response = new HttpResponse();
       response.setHeader('contentType', 'application/json');

response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"}}');
       response.setStatusCode(200);
       return response;

   }
}
```

ParkLocator Class

```
public class ParkLocator {


    public static List < String > country(String country) {
        ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
        return prkSvc.byCountry(country);
    }
    }
```

ParkLocatorTest Class

```
@isTest
public class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        List<String> expectedParks = new List<String>{'Yosemite', 'Sequoia', 'Crater
Lake'};
        System.assertEquals(expectedParks,ParkLocator.country(country));


    }
}
```

ParkServiceMock Class

```
@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) {
            parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
                response_x.return_x = new List<String>{'Yosemite', 'Sequoia', 'Crater
Lake'};
                    response.put('response_x', response_x);
        }
}
```

AccountManager Class

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

    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        String accountId =
request.requestURI.substringBetween('Accounts/','/contacts');
        Account result = [SELECT ID,Name,(SELECT ID, FirstName, LastName FROM
Contacts)
                                FROM Account
                                WHERE Id = :accountId];
        return result;
    }
```

AccountManagerTest Class

```
@isTest
private class AccountManagerTest {
```

```
    @isTest
    static void testGetAccount(){
        Account a = new Account(Name='TestAccount');
        insert a;
        Contact c = new Contact(AccountId=a.Id, FirstName='Test', LastName='Test');
        insert c;

        RestRequest request = new RestRequest();
        request.requestUri
='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        Account myAcct = AccountManager.getAccount();
        System.assert(myAcct !=null);
        System.assertEquals('TestAccount', myAcct.Name);
    }

}
```