

## **APEX SPECIALIST SUPERBADGE**

\*AUTOMATE RECORD CREATION:

### **1)MaintenanceRequest.apxt**

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

### **2)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;
}
}
}

```

\*SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

**1)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);
            }
        }
    }
}
```

```

    }

    }

}

```

\*SCHEDULE SYNCHRONIZATION USING APEX CODE:

### 1)WarehouseSyncSchedule.apxc

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

\*TEST AUTOMATION LOGIC:

### 1)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);
```

```
}
}
```

## 2)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;
}
}
}

```

### 3)MaintenanceRequest.apxt

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

```

    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}

```

\*TEST CALLOUT LOGIC:

### 1)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);
        }
    }
}

```

## 2) 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]);
    }
}

```

## 3) 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":
"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}');
        response.setStatusCode(200);

```

```

        return response;
    }
}

```

\*TEST SCHEDULING LOGIC:

### 1)WarehouseSyncSchedule.apxc

```

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

```

```

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

### 2)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 ');

```

```

    }
}

```

## **APPEX TRIGGERS**

\*GET STARTED WITH APEX TRIGGERS:

### **1.AccountAddressTrigger.apxt**

trigger AccountAddressTrigger on Account (before insert,before update) {

```
    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

\*BULK APEX TRIGGERS:

### **1.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;
    }
}
```

## **APPEX TESTING**

\*GET STARTED WITH APEX UNIT TEST:

### **1.VerifyDate.apxc**

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; }
}
```

## **2.TestVerifyDate.apxc**

@isTest

public class TestVerifyDate

```
{
    static testMethod void testMethod1()
    {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
    }
}
```

\*TEST APEX TRIGGERS:

### **1.RestrictContactByName.apxt**

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');
    }
}

}
```

\*CREATE TEST DATA FOR APEX TESTS:

### **1.RandomContactFactory.apxc**

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numOfContacts, String
lastName) {
        List<Contact> contacts = new List<Contact>();
```

```

        for(Integer i=0;i<numOfContacts;i++) {
            Contact c = new Contact(FirstName= 'Test ' + i, LastName=lastname);
            contacts.add(c);

        }
        //insert contactList;
        system.debug(contacts);
        return contacts;
    }

}

ASYNCHRONOUS APEX
*USE FUTURE METHODS:
1.AccountProcessor.apxc
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){

        List<Account> accountsToUpdate = new List<Account>();

        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where
        Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);

        }
        update accountsToUpdate;
    }

}

2.AccountProcessorTest.apxc
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts(){
        Account newaccount = new Account(Name='Test Account');
        insert newAccount;
    }
}

```

```
    Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
    insert newContact1;
```

```
    Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
    insert newContact2;
```

```
List<Id> accountIds = new List<Id>();
accountIds.add(newAccount.Id);
```

```
Test.startTest();
AccountProcessor.countContacts(accountIds);
Test.stopTest();
}
```

```
}
```

\*USE BATCH APEX:

### **1.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);
    }
}
```



```
}
```

## **2.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:

## **1.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;
        }

    }

}

```

## 2.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 = 'John', LastName = 'Doe');
        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 APEX SCHEDULER:
1.DailyLeadProcessor.apxc
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;
        }

    }

}

}

2.DailyLeadProcessorTest.apxc
@isTest
public class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 2 6 ? 2023';
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();
        for(Integer i=0; i<200; i++){
            Lead lead = new Lead(
                LastName = 'Test'+i,
                Company = 'Test Company' +i,
                Status= 'Open -Not Contacted');

            leads.add(lead);
        }
        insert leads;

        Test.startTest();
    }
}

```

```

        String jobId = System.schedule('Update LeadSource to Dreamforce',CRON_EXP, new
DailyLeadProcessor());
        Test.stopTest();

```

```

    }

```

```

}

```

APEX INTEGRATION SERVICES

\*APEX REST CALLOUTS:

### 1. AnimalLocator.apxc

```

public class AnimalLocator {
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal = new Map<String, Object>();
        HttpResponse res = http.send(req);
        if(res.getStatusCode() == 200){
            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) results.get('animal');
        }
        return (String)animal.get('name');
    }
}

```

```

}

```

### 2. AnimalLocatorMock.apxc

@isTest

global class AnimalLocatorMock implements HttpCalloutMock{

```

    global HTTPResponse respond(HTTPRequest request){

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
        response.getStatusCode(200);
        return response;
    }
}

```

```
}
```

### **3. AnimalLocatorTest.apxc**

```
@isTest
```

```
private class AnimalLocatorTest {
```

```
    @isTest static void AnimalLocatorMock1(){
```

```
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
```

```
        string result = AnimalLocator.getAnimalNameById(3);
```

```
        String expectedResult = 'chicken';
```

```
        System.assertEquals(result, expectedResult);
```

```
    }
```

```
}
```

\*APEX SOAP CALLOUTS:

### **1. 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' };
1
```

```
        private String[] apex_schema_type_info = new String[]{ 'http://parks.services/', 'false', 'false' };
1
```

```
        private String[] field_order_type_info = new String[]{ 'return_x' };
1
```

```
    }
```

```
    public class byCountry {
```

```
        public String arg0;
```

```
        private String[] arg0_type_info = new String[]{ 'arg0','http://parks.services/', null, '0', '1', 'false' };
1
```

```
        private String[] apex_schema_type_info = new String[]{ 'http://parks.services/', 'false', 'false' };
1
```

```
        private String[] field_order_type_info = new String[]{ 'arg0' };
1
```

```
    }
```

```
    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' };
1
```

```
        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;
    }
}
}

```

## 2.ParkServiceMocck.apxc

@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 responseNam,
        String responseType
    ){
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone','Mackinac National Park','Yosemite'};
        response.put('response_x',response_x);
    }
}

```

### 3.ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        List<String> result = ParkLocator.Country(country);
        List<String> parks = new List<String>{'Yellowstone','Mackinac National Park','Yosemite'};
        System.assertEquals(parks, result);
    }
}
```

}

\*APEX WEB SERVICES:

### 1.AccountManager.apxc

```
@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,Name from Contacts) from Account where
Id=:accountId Limit 1];
        return result;
    }
}
```

}

### 2.AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.my.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);
    }
}
```

```
}

static Id createTestRecord(){
    Account accountTest = new Account(Name = 'Test record');
    insert accountTest;

    Contact contactTest = new Contact(
        FirstName='John',
        LastName='Doe',
        AccountId=accountTest.Id
    );
    insert contactTest;
    return accountTest.Id;
}
}
```