Apex Specialist Superbadge

Step 2 : Automate Record Creation

MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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()){
         Map<Id,Case> closedCases = 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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Step 3 : Synchronize Salesforce data with an external system

WarehouseCalloutService.cls

```
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(){
    System.debug('go into runWarehouseEquipmentSync');
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);

    List<Product2> product2List = new List<Product2>();
    System.debug(response.getStatusCode());
    if (response.getStatusCode() == 200){
      List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
      System.debug(response.getBody());

      for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            product2.Cost__c = (Integer) mapJson.get('cost');
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
         }

         if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
         }
      }
   }

   public static void execute (QueueableContext context){
      System.debug('start runWarehouseEquipmentSync');
      runWarehouseEquipmentSync();
      System.debug('end runWarehouseEquipmentSync');
   }

}
```

Step 4 : Schedule synchronization

WarehouseSyncSchedule.cls

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

Step 5 : Test automation logic

MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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()){
        Map<Id,Case> closedCases = 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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today’s
date.
            //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> clonedList = new
List<Equipment_Maintenance_Item__c>();
         for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
               Equipment_Maintenance_Item__c item = clonedListItem.clone();
               item.Maintenance_Request__c = nc.Id;
               clonedList.add(item);
            }
         }
         insert clonedList;
      }
   }
}
```

MaintenanceRequestHelperTest.cls

```
@isTest
public with sharing class MaintenanceRequestHelperTest {


   private static Vehicle__c createVehicle(){
      Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
      return vehicle;
   }
   private static Product2 createEquipment(){
      product2 equipment = new product2(name = 'Testing equipment',
                                 lifespan_months__c = 10,
                                 maintenance_cycle__c = 10,
                                 replacement_part__c = true);
      return equipment;
   }

   private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
      case cse = new case(Type='Repair',
                     Status='New',
                     Origin='Web',
                     Subject='Testing subject',
                     Equipment__c=equipmentId,
                     Vehicle__c=vehicleId);
      return cse;
   }

   private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
      Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
         Equipment__c = equipmentId,
         Maintenance_Request__c = requestId);
      return equipmentMaintenanceItem;
   }

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

      Product2 equipment = createEquipment();
      insert equipment;
      id equipmentId = equipment.Id;
```

```
        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;

        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();

        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
                        Vehicle__c,
                        Date_Due__c
                       from case
                       where status ='New'];

        Equipment_Maintenance_Item__c workPart = [select id
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }
    @isTest
    private static void testNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

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

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

        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;

        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                          from Equipment_Maintenance_Item__c
                                          where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
```

```
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.
get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                              from case
                              where status ='New'];
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                   from Equipment_Maintenance_Item__c
                                                   where Maintenance_Request__c in: oldCaseIds];

        system.assert(newCase.size() == 300);

        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

Step 6 : Test callout logic

WarehouseCalloutService.cls

```
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(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
      System.debug(response.getBody());

      for (Object jR : jsonResponse){
        Map<String,Object> mapJson = (Map<String,Object>)jR;
        Product2 product2 = new Product2();

        product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

        product2.Cost__c = (Integer) mapJson.get('cost');

        product2.Current_Inventory__c = (Double) mapJson.get('quantity');

        product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');

        product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');

        product2.Warehouse_SKU__c = (String) mapJson.get('sku');

        product2.Name = (String) mapJson.get('name');
        product2.ProductCode = (String) mapJson.get('_id');
        product2List.add(product2);
      }

      if (product2List.size() > 0){
        upsert product2List;
        System.debug('Your equipment was synced with the warehouse one');
      }
    }
  }

  public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
  }
```

```
}
```

WarehouseCalloutServiceMock.cls

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {

    global static HttpResponse respond(HttpRequest request) {

        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"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceTest.cls

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
            @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}
```

Step 7 : test scheduling logic

WarehouseCalloutServiceMock.cls

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {

    global static HttpResponse respond(HttpRequest request) {

        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"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseSyncSchedule.cls

```
global with sharing class WarehouseSyncSchedule implements Schedulable {

    global void execute (SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

WarehouseSyncScheduleTest.cls

```
@isTest
public with sharing class WarehouseSyncScheduleTest {

       @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```

Apex Modules

AccountAddressTrigger.apxt

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

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

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;
    }
}
```

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);
   }
}
```

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());
   }
}
```

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);
       }
       System.debug(contacts);
       return contacts;
    }
}
```

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;
  }
}
```

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);
        AccountProcessor.countContacts(accountIds);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);

        Test.stopTest();

    }
}
```

LeadProcessor.apxc

```
public class LeadProcessor implements
    Database.Batchable<sObject>, Database.Stateful {

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead '
        );
    }
    public void execute(Database.BatchableContext bc, List<Lead> scope){
        List<Lead> leads=new List<Lead>();
        for (Lead lead : scope) {
           lead.LeadSource='Dreamforce';
            leads.add(lead);
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
    }
}
```

LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,Company='Test Co'));
        }
        insert leads;

    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
                    Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce']);
    }
}
```

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='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')]);
    }
}
```

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx) {
        List<Lead> leadstoupdate=new List<Lead>();
        List<Lead> leads = [SELECT Id
            FROM Lead
            WHERE LeadSource = NULL Limit 200
            ];
        for(Lead l:leads){
            l.LeadSource='Dreamforce';
            leadstoupdate.add(l);
        }
        update leadstoupdate;
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 3 ? 2023';
    static testmethod void testScheduledJob() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
            Lead l = new Lead(
                FirstName = 'First ' + i,
                LastName = 'LastName',
                Company = 'The Inc'
            );
            leads.add(l);
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('ScheduledApexTest',
                                CRON_EXP,
                                new DailyLeadProcessor());

        Test.stopTest();
        List<Lead> checkleads=new List<Lead>();
        checkleads = [SELECT Id
                  FROM Lead
                  WHERE LeadSource='Dreamforce' and Company='The Inc'];
        System.assertEquals(200,
                    checkleads.size(),
                    'Leads were not created');
    }
}
```

AnimalLocator.apxc

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

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest
{
    @isTest static void getAnimalNameByIdTest()
    {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string response = AnimalLocator.getAnimalNameById(1);

        System.assertEquals('chicken', response);
    }
}
```

ParkService.apxc

```
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ParkServiceMock.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 responseName,
        String responseType) {

     List<String> parks=new List<string>();
          parks.add('Yosemite');
          parks.add('Yellowstone');
          parks.add('Another Park');
     ParkService.byCountryResponse response_x =
        new ParkService.byCountryResponse();
     response_x.return_x = parks;
     response.put('response_x', response_x);
  }
}
```

ParkLocator.apxc

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

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>();
         parks.add('Yosemite');
         parks.add('Yellowstone');
         parks.add('Another Park');
      System.assertEquals(parks, result);
   }
}
```

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];
      return result;
   }
}
```

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest {
   @isTest static void testGetContactsByAccountById() {
      Id recordId = createTestRecord();
      RestRequest request = new RestRequest();
      request.requestUri =
      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;
   }
}
```