Apex Specialist Superbadge

Use integration and business logic to push your Apex coding skills to the limit.

Challenge-1

Automate Record Creation

MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
        if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
```

```
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECTId,
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()
                 );
                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;
      }
   }
}
```

Challenge-2

Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc:

```
public with sharing class WarehouseCalloutService implements Queueable {

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

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        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> warehouseEq = 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 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){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
```

```
    }
}
```

Challenge-3

Schedule Synchronization

WarehouseSyncSchedule.apxc:

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

        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

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 ');



    }
}
```

Challenge-4

TestAutomation Logic

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;
      }
   }
}
```

MaintenanceRequestHelperTest.apxc:

```
@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 cs = new case(Type='Repair',
                          Status='New',
                          Origin='Web',
                          Subject='Testing subject',
                          Equipment__c=equipmentId,
                          Vehicle__c=vehicleId);
       return cs;
    }

    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(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 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 workP =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert workP;

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

        Case newReq = [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 =:newReq.Id];

    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, 'Routine Maintenance');
    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 = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

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

```
        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(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(createEquipment());
}
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(createEquipmentMaintenanceItem(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 =:'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);
    }
}
```

Challenge-5

Test callout logic

WarehouseCalloutServiceMock:

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

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]);
    }
}
```

Challenge-6

Test Scheduling Logic

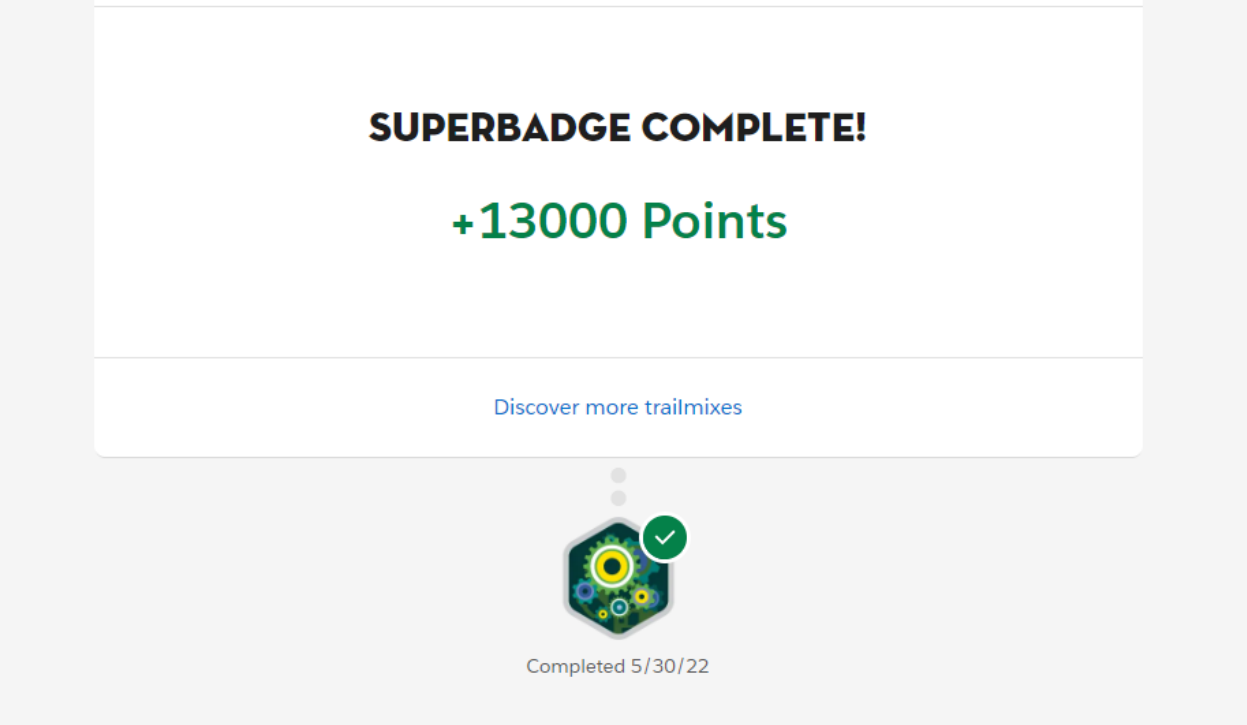
WarehouseSyncSchedule.apxc:

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

        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

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 ');
    }
}
```



Trailhead URL: https://trailblazer.me/id/dkosanam

ApexURL:

https://trailhead.salesforce.com/en/content/learn/superbadges/superbadge_apex

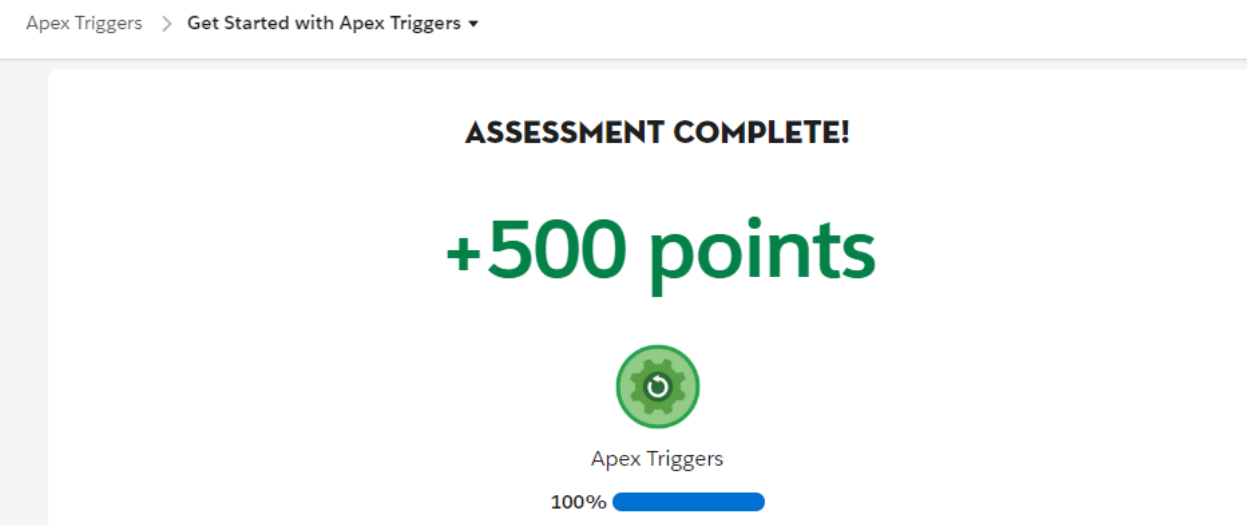
Apex Triggers

Challenge-1

Get Started with Apex Triggers

AccountAddressTrigger.apxc:

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

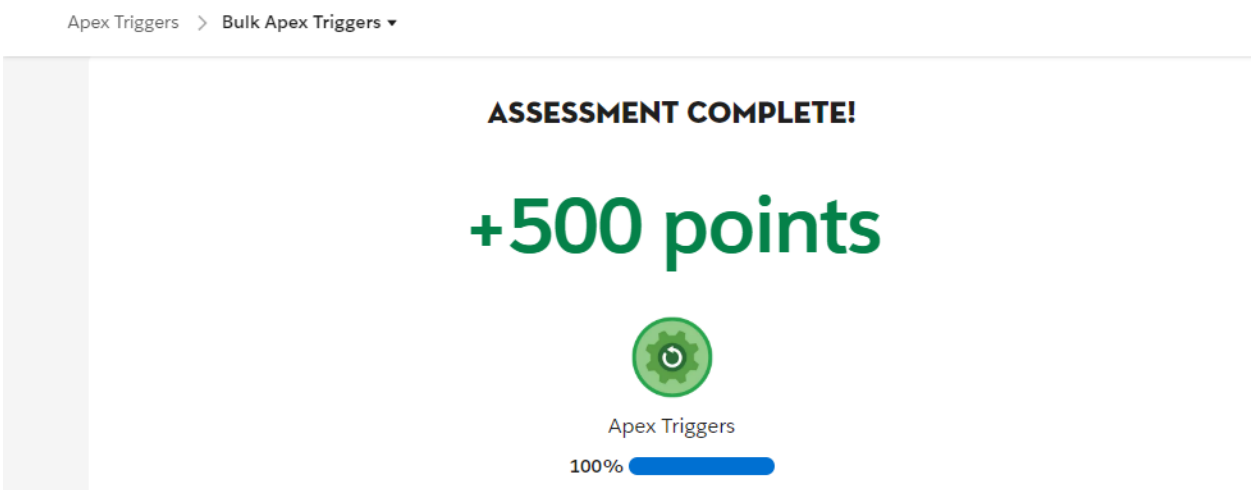


Challenge-2

Bulk Apex Triggers

ClosedOpportunityTrigger.apxc:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
   List<Task> taskList =new List<Task>();
    for(Opportunity opp : Trigger.New){
        if(opp.StageName == 'Closed Won'){
            taskList.add(new Task(Subject='Follow Up Test Task',WhatId =opp.Id));
        }
    }
    if(taskList.size()>0){
        insert taskList;
    }
}
```



Apex Testing

Challenge-1

Get Started with Apex Unit Tests

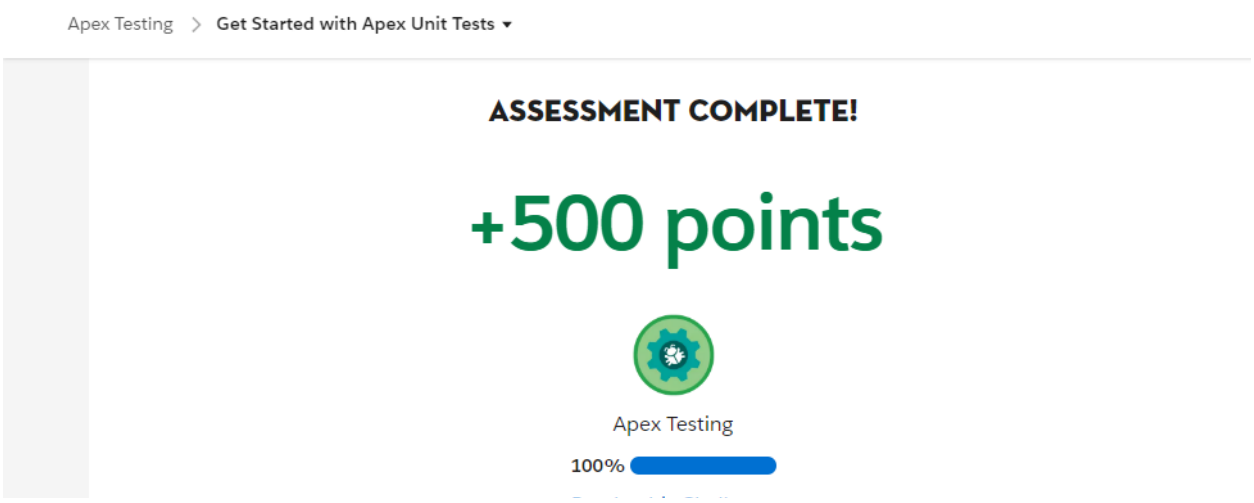
VerifyDate.apxc:

```
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('01/01/2021'),Date.parse('01/03/2021'));
       System.assertEquals(Date.parse('01/03/2021'),d);
    }
     @isTest static void test2(){
       Date d=verifyDate.CheckDates(Date.parse('01/01/2021'),Date.parse('03/03/2021'));
       System.assertEquals(Date.parse('01/31/2021'),d);
    }
}
```



Challenge-2

Test Apex Triggers

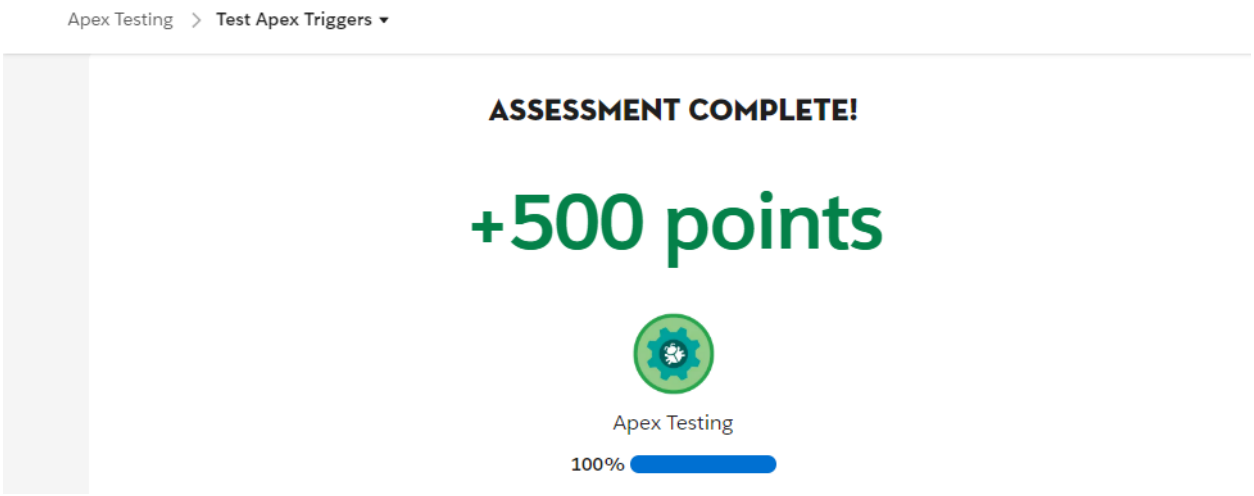
RestrictContactByName.apxc:

```
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 static void createBadContact()
   {
      Contact c=new Contact(FirstName='John',LastName='INVALIDNAME');
      Test.startTest();
      Database.SaveResult result = Database.insert(c, false);
      Test.stopTest();
```

```
        System.assert(!result.isSuccess());
    }
}
```



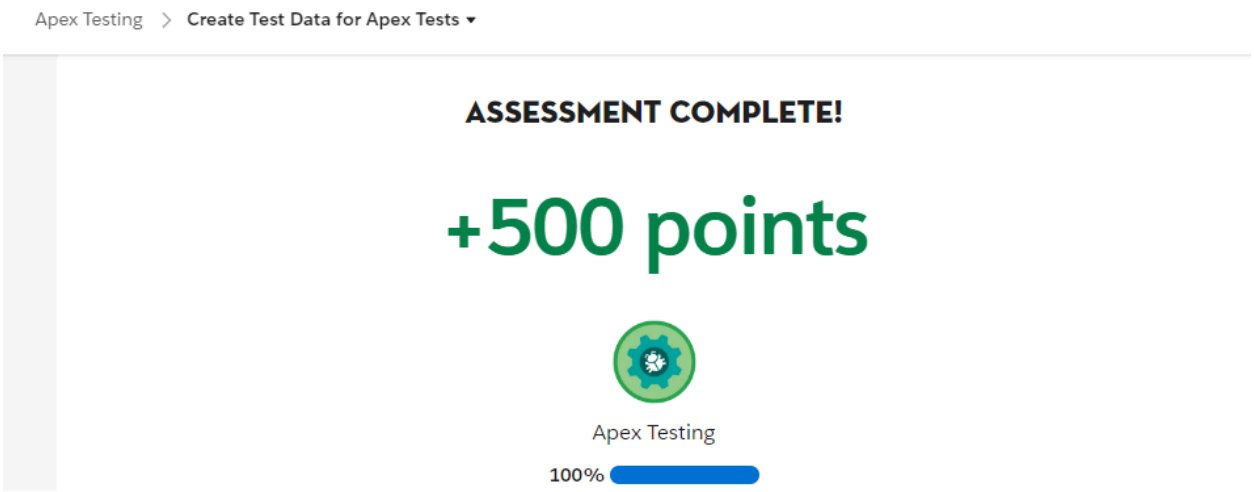
Challenge-3

Create Test Data for Apex Tests

RandomContactFactory.apxc:

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num,String lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i=1;i<=num;i++){
            Contact ct=new Contact(FirstName ='Test'+i,LastName =lastname);
```

```
            contactList.add(ct);
        }
        return contactList;
    }

}
```



Asynchronous Apex

Challenge-1

Use Future Methods

AccountProcessor.apxc:

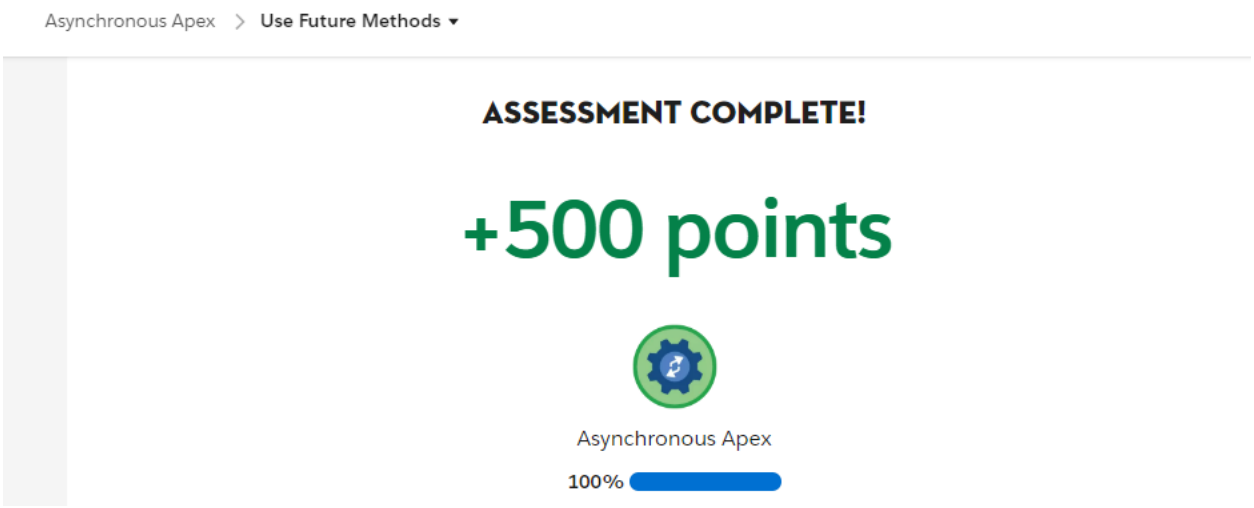
```
public class AccountProcessor {

  @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);

          Test.startTest();



     AccountProcessor.countContacts(accountIds);

    Test.stopTest();

  }

}
```



Challenge-2

Use Batch Apex

LeadProcessor.apxc:

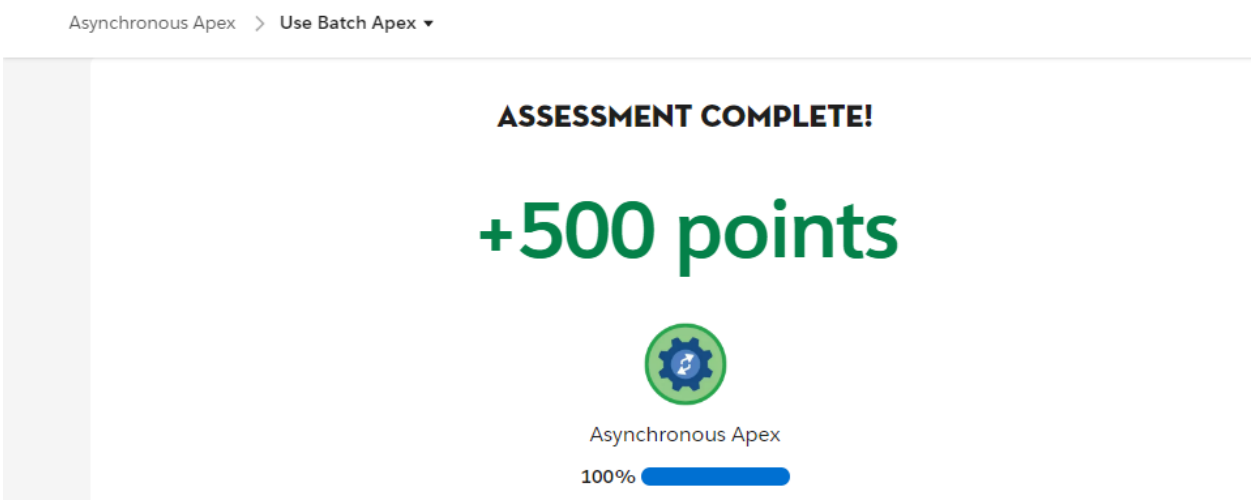
```
global class LeadProcessor implements Database.Batchable<sObject> {

    global Integer count = 0;

   global Database.QueryLocator start(Database.BatchableContext bc) {
```

```
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

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

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

LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();
```

```
    for(Integer i=0; i<200; i++){
        Lead L = new lead();
       L.LastName = 'name' + i;
        L.company = 'Company';
        L.Status = 'Random Status';
        L_list.add(L);
    }
    insert L_list;

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



Challenge-3

Control Processes with Queueable Apex

AddPrimaryContact.apxc:

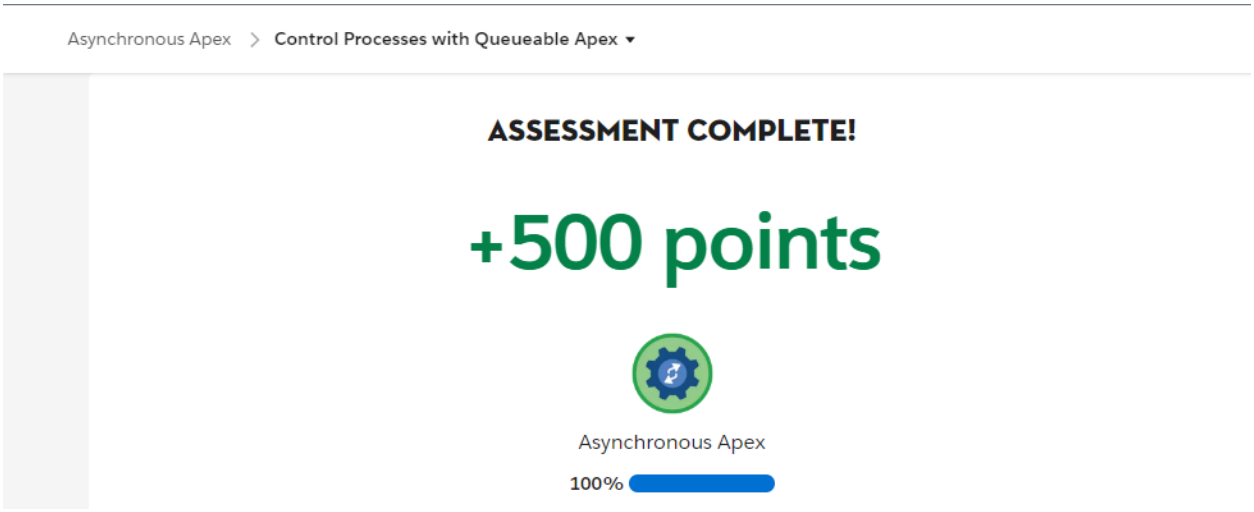
```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact(Contact con, String state){
      this.con = con;
      this.state = state;
  }
  public void execute(QueueableContext context){
     List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts) from Account where BillingState = :state Limit 200];
     List<Contact> primaryContacts = new List<Contact>();

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

```
}
```

AddPrimaryContactTest.apxc:

```
@isTest
public class AddPrimaryContactTest {
    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0; i<50;i++){
            testAccounts.add(new Account(Name='Account' +i,BillingState='CA'));


        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account' +j,BillingState='NY'));
         }
        insert testAccounts;
        Contact testContact = new Contact(FirstName = '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')]);
```

```
}
}
```



Challenge-4

Schedule Jobs Using the Apex Scheduler

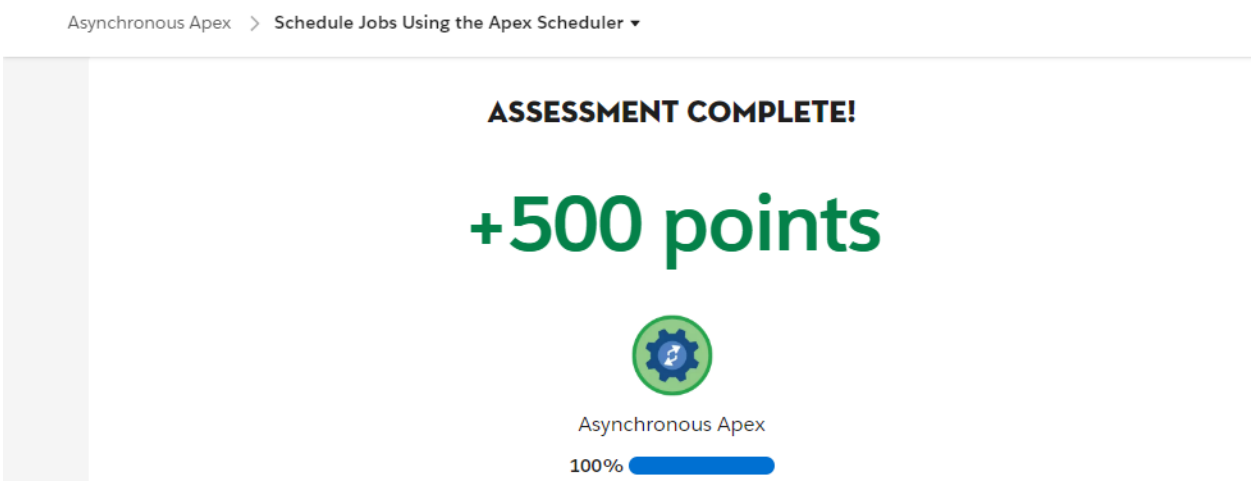
DailyLeadProcessor.apxc:

```
public class DailyLeadProcessor implements Schedulable
{
   public void execute(SchedulableContext SC)
   {
      List<Lead> LeadObj =[SELECT Id From Lead Where LeadSource =null limit 200];
```

```
        for(Lead l : LeadObj){
          l.LeadSource ='Dreamforce';
            update l;
      }
   }
}
```

DailyLeadProcessorTest.apxc:

```
@isTest
private class DailyLeadProcessorTest {
   static testmethod void testDailyLeadProcessor()  {
      String CRON_EXP = '0 0 1 * * ?';
      List<Lead> lead = new List<Lead>();
      for(Integer i=0; i<200; i++){
         lead.add(new Lead(LastName ='Dreamforce'+i, Company ='Test Inc.', Status = 'Open. Not Contacted'));
      }
      insert lead;
 Test.startTest();
      String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new DailyLeadProcessor());
   }
}
```



APEX INTEGRATION SERVICES

Challenge-1

Apex REST Callouts

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;
    }
}
```

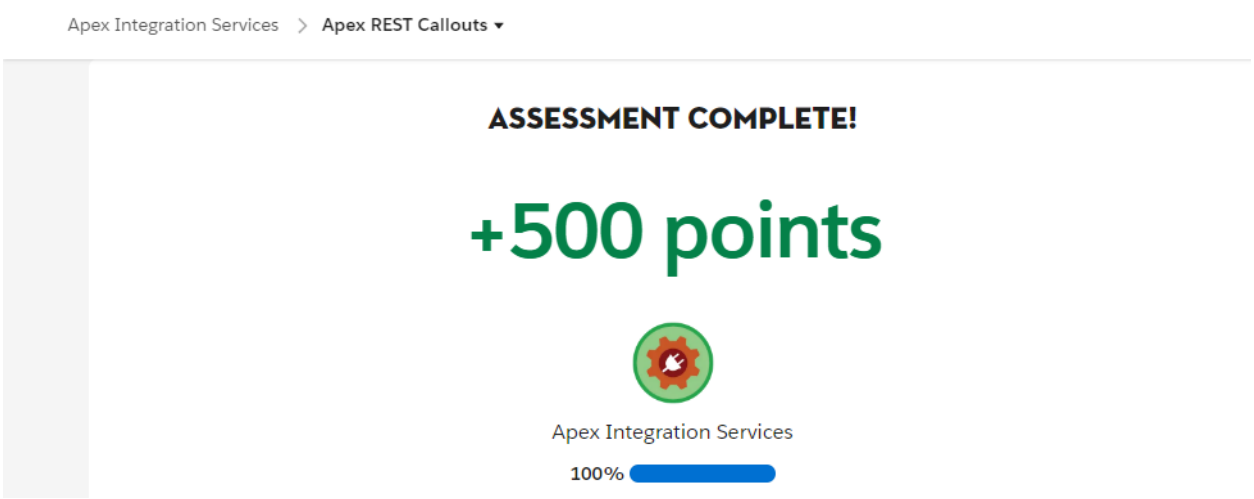
AnimalLocatorMock.apxc:

```
@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.setStatusCode(200);
        return response;
    }
}
```

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);
}
}
```



Challenge-2

Apex SOAP Callouts

ParkLocator.apxc:

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

```
}
```

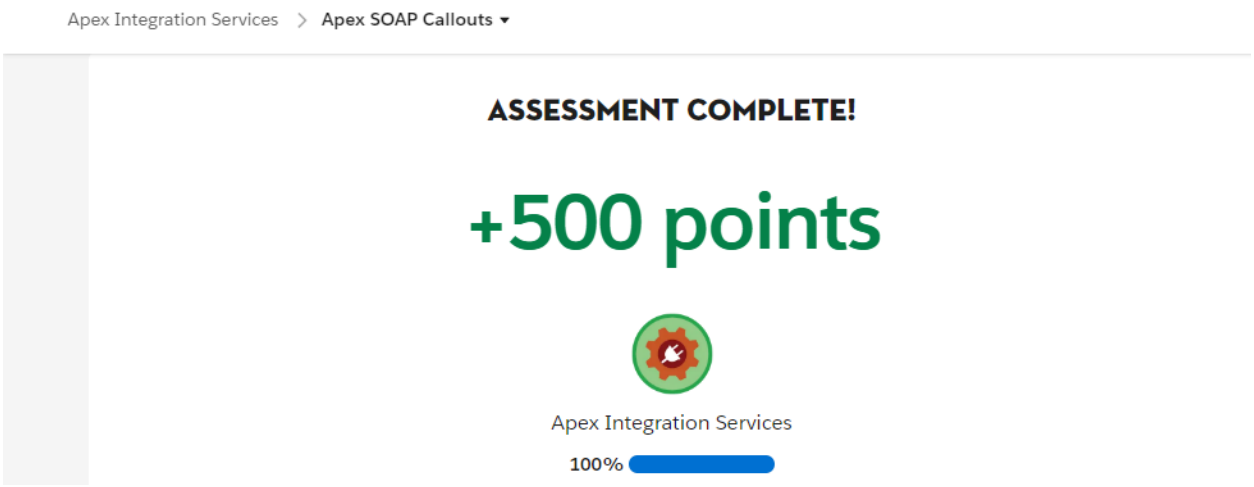
ParkLocatorTest.apxc:

```
@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);
    }
}
```

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);
  }
}
```



Challenge-3

Apex Web Services

AccountManager.apxc:

```
@RestResource(urlMapping='/Accounts/*')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
        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 testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        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;
    }
}
```

ASSESSMENT COMPLETE!

+500 points

Apex Integration Services

100%