SAURAV KUMAR GUPTA

SPSG-89531

vtu11817@veltech.edu.in

Vel Tech Rangarajan Dr Sagunthala R&D Institute of Science and Technology

https://trailblazer.me/id/sveltechchennai38

INTERN : Salesforce Developer Catalyst Self-Learning & Super Badges

Module: Get started with APEX triggers

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


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

Module: APEX Testing - Bulk APEX Triggers

```
trigger ClosedOpportunityTrigger on Opportunity (afterinsert, 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;
}
```

Module: Get Started with APEX Unit Tests

```
public class VerifyDate {


        //method to handle potential checks against two
        datespublic static Date CheckDates(Date date1,
        Date date2) {
                //if date2iswithin the next 30 days of date1,use date2.  Otherwiseuse the end of
                the
month
```

```
        if(DateWithin30Days(date1,dat
                e2)) {return date2;
        } else {
                return SetEndOfMonthDate(date1);
        }
}

 //method to checkif date2 is within the next 30 days of date1
 @TestVisible private static Boolean DateWithin30Days(Date date1,Date date2) {
        //check for date2 being in the
pastif( date2 < date1){ return false; }

//check that date2 is within(>=)30 days of date1
Date date30Days = date1.addDays(30); //createadate 30 days away from
        date1if( date2 >= date30Days ) { return false; }
        else { return true; }
 }
 //method to return the end of the month of a given
 date@TestVisible private static Date
 SetEndOfMonthDate(Date date1){
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(),
        totalDays);returnlastDay;
```

```
        }

}

@isTest
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);

    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
}
    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
        date.parse('12/30/2019'));System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
```

```
        date.parse('02/02/2020'));System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
        date.parse('01/15/2020'));System.assertEquals(false, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}
```

Model: Test Apex Triggers

```
trigger RestrictContactByName on Contact (beforeinsert, before update){

        //check contactsprior to insert or update for
        invaliddataFor (Contactc : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {    //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not
                        allowedfor DML');
                }
        }

}

@isTest
public class TestRestrictContactByName {
```

```
  @isTest static void
    Test_insertupdateContact(){Contact cnt =
    new Contact();
    cnt.LastName = 'INVALIDNAME';

    Test.startTest();
    Database.SaveResult result = Database.insert(cnt,
    false);Test.stopTest();

    System.assert(!result.isSuccess());
    System.assert(result.getErrors().size() >
    0);
    System.assertEquals('The Last Name "INVALIDNAME" is not allowdedfor
DML',result.getErrors()[0].getmessage());

  }
}
```

Module: Create Test

Data for Apex Tests

```
public class RandomContactFactory
{
```

```
    public static List<Contact> generateRandomContacts(Integer numcnt, string
      lastname){List<Contact> contacts=new List<Contact> ();
      for(Integer i=0;i<numcnt;i++){
        Contact cnt = new Contact(FirstName = 'Test '+i, LastName =
        lastname);contacts.add(cnt);

      }
      return contacts;
    }
}
```

Asynchronous Apex >Use Future Methods

```
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;



    }

}
```

```
@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();
    }

}
```

Module:Use Batch Apex

```
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);
    }
}

@isTest
public classLeadProcessorTest {

    @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();
    }
}
```

Module: Control Processes with Queueable Apex

```
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, 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;
        }
    }
}


@isTest
public classAddPrimaryContactTest {


    statictestmethod 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(addi

    t);Test.stopTest();



    System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from Account
  whereBillingState='CA')]);

    }
}
```

Module: Apex Integration Services

```
 global class DailyLeadProcessor implements

    Schedulable{global 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';leadstoupdat
            e.add(l);
        }
        update leadstoupdate;
    }
}

@isTest
private classDailyLeadProcessorTest {

    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');
    }
}
```

APEX CALL OUT SERVICES

```
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');
   }
}
```

```
  @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
        );
     }
  }
```

```
  @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 Integration Services

Apex SOAP Callouts

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space
        return parkSvc.byCountry(theCountry);
    }
}
```

```
@isTest
private class ParkLocatorTest
    {@isTeststatic 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', 'MackinacNational Park',
         'Yosemite'};System.assertEquals(parks, result);
    }
}

@isTest
globalclass ParkServiceMock implements
   WebServiceMock {global void doInvoke(
           Object stub,
           Objectreque
           st,

           Map<String, Object>
           response,String endpoint,
           String soapAction,
           String
           requestName,
           String responseNS,
           String
           responseName,
```

```
        StringresponseTyp

        e) {

      // start - specify the response you want to send

      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();

      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',

      'Yosemite'};

      // end

      response.put('response_x', response_x);

   }

}
```

```
//Generated by wsdl2apex


public class AsyncParkService {

   publicclass byCountryResponseFuture extendsSystem.WebServiceCalloutFuture

      {publicString[] getValue() {

         ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);

         return response.return_x;

      }

   }

   public class AsyncParksImplPort {
```

```
public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';

public Map<String,String>

inputHttpHeaders_x;public String

clientCertName_x;

public Integer timeout_x;

private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};

public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
continuation,Stringarg0) {

    ParkService.byCountry request_x = new

    ParkService.byCountry();request_x.arg0 = arg0;

    return(AsyncParkService.byCountryResponseFuture)

     System.WebServiceCallout.beginInvoke(this,

     request_x,

     AsyncParkService.byCountryResponseFuture.cla

     ss,continuation,

     new String[]{endpoint_x,

     '',

     'http://parks.service

     s/','byCountry',

     'http://parks.service

     s/',

     'byCountryRespons
```

```
            e',
            'ParkService.byCountryResponse'}
          );
        }
    }
}
```

Apex Web Services

```
@RestResource(urlMapping='/Accounts/*/conta
cts')global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req =
        RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/',
        '/contacts');Account acc = [SELECT Id, Name, (SELECT Id, Name
        FROM Contacts)
                FROM Account WHERE Id =
        :accId];return acc;
    }
}
```

```
@isTest
private classAccountManagerTest {
```

```
privatestatic testMethod void
    getAccountTest1() {Id recordId=
    createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
    +'/contacts' ;request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount !=
    null);
    System.assertEquals('Test record', thisAccount.Name);

}

// Helper method
    static Id createTestRecord() {
    // Create test record
    AccountTestAcc = new
```

```
        Account(Name='Test
        record');
      insert TestAcc;
      Contact TestCon= new
      Contact(LastName='Test',
      AccountId =
      TestAcc.id);return
      TestAcc.Id;
    }
}
```

SUPERBADGE:APEX SPECIALIST

Step2: Automate recordcreation

```
public with sharing class MaintenanceRequestHelper {
    publicstatic void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
      nonUpdCaseMap) {Set<Id> validIds = new Set<Id>();



      For (Casec : 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_Itemsr)
                                    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
WHEREMaintenance_Request_c IN :ValidIds GROUP BY Maintenance_Request_c];

for (AggregateResult ar : results){
    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'),(Decimal)ar.get('cycle'));
}

    for(Case cc :
        closedCasesM.values()){Case nc
        = new Case (
            ParentId =
        cc.Id,Status =
```

```
    'New',
        Subject = 'Routine Maintenance',
        Type = 'Routine
        Maintenance',Vehicle_c =
        cc.Vehicle_c, Equipment_c
        =cc.Equipment_c,Origin=
        'Web',
        Date_Reported__c = Date.Today()

    );

    If (maintenanceCycles.containskey(cc.Id)){
        nc.Date_Due_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
    } else {
        nc.Date_Due__c = Date.today().addDays((Integer) cc.Equipment__r.maintenance_Cycle__c);
    }

    newCases.add(nc);
 }

insert newCases;
```

```
        List<Equipment_Maintenance_Item_c> clonedWPs = new List<Equipment_Maintenance_Item_
        c>();for (Case nc : newCases){
            for (Equipment_Maintenance_Item_c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items_r){
                Equipment_Maintenance_Itemc wpClone = wp.clone();
                wpClone.Maintenance_Request_c = nc.Id;
                ClonedWPs.add(wpClone);


            }
        }
        insert ClonedWPs;
    }
  }
}


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


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


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


  }
```

}

Synchronize Salesforce data with an external system

```
public with sharingclass WarehouseCalloutService implements Queueable {

    privatestatic final StringWAREHOUSE_URL = 'https://th-superbadge-

apex.herokuapp.com/equipment';



    //classthat makesa REST calloutto an externalwarehouse systemto get a list ofequipment that needs to be updated.

    //Thecallout’s JSON responsereturns the equipmentrecords that you upsertin Salesforce.



    @future(callout=true)

    publicstatic 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());


    //classmaps the following fields:replacement part (alwaystrue), cost, currentinventory,
lifespan, maintenance cycle, and warehouseSKU
    //warehouse SKU will be external ID for identifying which equipment
recordstoupdate withinSalesforce
    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){upsertwarehouseEq;
              System.debug('Your equipment was synced with the warehouseone');
        }
    }
}


  publicstatic void execute(QueueableContext
    context){runWarehouseEquipmentSync();
  }


}
```

Schedule synchronization

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

}

Test automation logic

```
public with sharingclass MaintenanceRequestHelper {

    publicstatic 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);

                }

            }

        }



        //Whenan existingmaintenance request of type Repairor Routine Maintenance isclosed,

        //create a new maintenance requestfor a future routine checkup.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__cFROM
```

```
Equipment_Maintenance_Itemsr)

                                        FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();


    //calculate the maintenance requestdue dates by using the maintenance cycledefined
on the related equipmentrecords.
    AggregateResult[] results = [SELECT Maintenance_Request_c,
                                 MIN(Equipment_r.Maintenance_Cycle_c)cycle
                                 FROMEquipment_Maintenance_Item_c
                                 WHEREMaintenance_Request_c IN :ValidIdsGROUP 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 equipmentare used in the maintenance request,
        //define the due date by applying the shortest maintenance cycle to today’sdate.
        //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;
    }
  }
}


@istest
public with sharingclass MaintenanceRequestHelperTest{
```

```
private static final stringSTATUS_NEW =
'New';private static final string WORKING =
'Working';private static final string CLOSED =
'Closed'; privatestaticfinal string
REPAIR='Repair';
private staticfinalstring REQUEST_ORIGIN = 'Web';
private staticfinalstring 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');
    returnVehicle;
}
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;
    }


    PRIVATESTATIC Equipment_Maintenance_Item_c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item_c wp = new
Equipment_Maintenance_Itemc(Equipment_c = equipmentId,
                                                    Maintenance_Request__c = requestId);
        return wp;
    }



    @istest
```

```
    privatestatic void testMaintenanceRequestPositive(){
        Vehicle_cvehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();
        insertequipment;
        id equipmentId = equipment.Id;

        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insertsomethingToUpdate;

        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
                    wherestatus =:STATUS_NEW];

        Equipment_Maintenance_Itemc 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
    privatestatic void testMaintenanceRequestNegative(){
```

```
Vehicle_Cvehicle = createVehicle();
insert vehicle;
id vehicleId= vehicle.Id;

product2 equipment = createEq();
insertequipment;

id equipmentId = equipment.Id;

case emptyReq= createMaintenanceRequest(vehicleId,equipmentId);
insertemptyReq;

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_Itemc 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 = [selectid
                                                    from Equipment_Maintenance_Item__c
                                                    where Maintenance_Request__cin: oldRequestIds];

     system.assert(allRequests.size() == 300);
  }
}

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

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

Test calloutlogic

```
 @isTest
 global class WarehouseCalloutServiceMock implements HttpCalloutMock {
     // implementhttp mock callout
     global staticHttpResponse respond(HttpRequest request){

         HttpResponse response = new HttpResponse();
         response.setHeader('Content-Type', 'application/json');

 response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"n
 ame":"Generator
 1000kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d662
 267 26b611100aaf742","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;
    }
}
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test
            here@isTest
    staticvoid 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);
    }
}
```

Test schedulinglogic

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

 @isTest
 global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
```

```
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"n
ame":"Generator
1000kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d662
267 26b611100aaf742","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;

    }

}



@isTest

public with sharingclass WarehouseSyncScheduleTest {

    // implement scheduledcode here

    //

    @isTest static void test() {

        String scheduleTime = '00 00 00 * * ? *';

        Test.startTest();

        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        StringjobId = System.schedule('Warehouse Time to Scheduleto 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();
    }
}
```

Module: Get Started With Apex triggers

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

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

Module:Apex Testing: Bulk Apex Triggers

```
trigger ClosedOpportunityTrigger on Opportunity (afterinsert, 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;
    }
}
```

Module:Get Started With Apex Unit Tests

```
public class VerifyDate {

        //method to handle potential checks against two
        datespublic static Date CheckDates(Date date1,
        Date date2) {
                //if date2iswithin the next 30 days of date1,use date2.  Otherwiseuse the end of
                the
month
                if(DateWithin30Days(date1,dat
                        e2)) {return date2;
```

```
        } else {
                return SetEndOfMonthDate(date1);
        }
}

 //method to checkif date2 is within the next 30 days of date1
 @TestVisible private static Boolean DateWithin30Days(Date date1,Date date2) {
        //check for date2 being in the
pastif( date2 < date1){ return false; }

//check that date2 is within(>=)30 days of date1
Date date30Days = date1.addDays(30); //createadate 30 days away from
        date1if( date2 >= date30Days ) { return false; }
        else { return true; }
 }

 //method to return the end of the month of a given
 date@TestVisible private static Date
 SetEndOfMonthDate(Date date1){
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(),
        totalDays);returnlastDay;
```

```
        }

}

@isTest
private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'), D);

   }

   @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'), D);
}
   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
      date.parse('12/30/2019'));System.assertEquals(false, flag);
   }
   @isTest static void Test_DateWithin30Days_case2(){
```

```
    Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
    date.parse('02/02/2020'));System.assertEquals(false, flag);
  }
  @isTest static void Test_DateWithin30Days_case3(){
    Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
    date.parse('01/15/2020'));System.assertEquals(false, flag);
  }
  @isTest static void Test_SetEndOfMonthDate(){
    Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
  }
}
```

Module: Test Apex Triggers

```
trigger RestrictContactByName on Contact (beforeinsert, before update){

        //check contactsprior to insert or update for
        invaliddataFor (Contactc : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {    //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not
                        allowedfor DML');
                }

        }
```

```
}
@isTest
public class TestRestrictContactByName {


   @isTest static void

      Test_insertupdateContact(){Contact cnt =

      new Contact();

      cnt.LastName = 'INVALIDNAME';



      Test.startTest();

      Database.SaveResult result = Database.insert(cnt,

      false);Test.stopTest();



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

      System.assert(result.getErrors().size() >

      0);

      System.assertEquals('The Last Name "INVALIDNAME" is not allowdedfor
DML',result.getErrors()[0].getmessage());



   }

}
```

Module: Create Test

Data for Apex Tests

```
public class RandomContactFactory
{

   public static List<Contact> generateRandomContacts(Integer numcnt, string
      lastname){List<Contact> contacts=new List<Contact> ();
      for(Integer i=0;i<numcnt;i++){
         Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
         contacts.add(cnt);
      }
      return contacts;
   }
}
```

Asynchronous Apex >Use Future Methods

```
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;


    }

}

@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();
    }

}
```

Module:Use Batch Apex

```
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);
    }
}


@isTest
public classLeadProcessorTest {


    @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();
  }
}
```

Module: Control Processes with Queueable Apex

```
        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, 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;
        }
    }


}


@isTest
public classAddPrimaryContactTest {


    statictestmethod 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(addi

    t);Test.stopTest();



    System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from Account
  whereBillingState='CA')]);

    }

}
```

Module: Apex Integration Services

```
  global class DailyLeadProcessor implements

    Schedulable{global 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';leadstoupdat
            e.add(l);
        }
        update leadstoupdate;
    }
}


@isTest
private classDailyLeadProcessorTest {


    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');
  }
}
```

Apex REST Callouts

```
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');
```

```
}
}

  @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

        );

     }

  }



  @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

```
public class ParkLocator {

    public static string[] country(string theCountry) {

        ParkService.ParksImplPort parkSvc = new  ParkService.ParksImplPort(); // remove space
        return parkSvc.byCountry(theCountry);

    }

}
```

```
@isTest

private class ParkLocatorTest

    {@isTeststatic 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', 'MackinacNational Park',

         'Yosemite'};System.assertEquals(parks, result);

    }

}
```

```
@isTest

globalclass ParkServiceMock implements

   WebServiceMock {global void doInvoke(

         Object stub,

         Objectreque

         st,

         Map<String, Object> response,

         String endpoint,

         String

         soapAction,

         String

         requestName,

         String

         responseNS,

         String

         responseName,

         String

         responseType) {

      // start - specify the response you want to send

      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();

      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',

      'Yosemite'};

      // end
```

```
        response.put('response_x', response_x);
    }
}
```

```
//Generated by wsdl2apex

public class AsyncParkService {
    publicclass byCountryResponseFuture extendsSystem.WebServiceCalloutFuture
        {publicString[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
        service.herokuapp.com/service/parks';public Map<String,String>
        inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
```

```
        public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
  continuation,Stringarg0) {
            ParkService.byCountry request_x = new
            ParkService.byCountry();request_x.arg0 = arg0;
            return(AsyncParkService.byCountryResponseFuture)
              System.WebServiceCallout.beginInvoke(this,
              request_x,
              AsyncParkService.byCountryResponseFuture.cla
              ss,continuation,
              new
              String[]{endpoint_x,
              '',
              'http://parks.service
              s/','byCountry',
              'http://parks.service
              s/',
              'byCountryRespons
              e',
              'ParkService.byCountryResponse'}
            );
        }
    }
}
```

Apex Integration Services:

Apex Web Services

```
@RestResource(urlMapping='/Accounts/*/conta

cts')global class AccountManager {

    @HttpGet

    global static Account getAccount() {

        RestRequest req =

        RestContext.request;

        String accId = req.requestURI.substringBetween('Accounts/',

        '/contacts');Account acc = [SELECT Id, Name, (SELECT Id, Name

        FROM Contacts)

                FROM Account WHERE Id = :accId];
        return acc;

    }

}
```

```
@isTest

private classAccountManagerTest {



    privatestatic testMethod void

        getAccountTest1() {Id recordId=

        createTestRecord();
```

```
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
    +'/contacts' ;request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount !=
    null);
    System.assertEquals('Test record',thisAccount.Name);

}

// Helper method
    static Id createTestRecord() {
    // Create test record
    AccountTestAcc = new
      Account(Name='Test
      record');

    insert TestAcc;
```

```
        Contact TestCon= new

        Contact(LastName='Test',

        AccountId = TestAcc.id);

        return TestAcc.Id;

    }

}
```

APEX SUPERBADGE CHALLENGE 2: Automated Record Creation

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();


      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);


            }
         }
      }

      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
```

```
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
        AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

      for (AggregateResult ar : results){
         maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
      }

         for(Case cc : closedCasesM.values()){
            Case nc = new Case (
               ParentId = cc.Id,
            Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
               nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            } else {
               nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            }

            newCases.add(nc);
         }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
```

```
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaitenanceRequest.apxt :-

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

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

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

  }

}
```

CHALLENGE 3: 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';
```

```
    //class that makes a REST callout to an external warehouse system to get a list of equipment
that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

        List<Product2> warehouseEq = new List<Product2>();

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

            //class maps the following fields: replacement part (always true), cost, current inventory,
lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
```

```
            System.debug('Your equipment was synced with the warehouse one');
        }
      }
    }

    public static void execute (QueueableContext context){
      runWarehouseEquipmentSync();
    }

}
```

CHECK

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

CHALLENGE 4: Schedule synchronization using Apex code

WarehouseSyncShedule.apxc :-

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

CHALLENGE 5: Test automation logic

MaintenanceRequestHelperTest.apxc :-

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                      Maintenance_Request__c = requestId);
```

```
        return wp;
    }


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

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

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

        Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                from case
                where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
```

```
    }

    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

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

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

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

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

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
```

```
        list<id> oldRequestIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){
           workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
           req.Status = CLOSED;
           oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();


      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);



            }
         }
      }

      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
         AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

      for (AggregateResult ar : results){
         maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
      }

         for(Case cc : closedCasesM.values()){
            Case nc = new Case (
               ParentId = cc.Id,
            Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
```

```
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaintenanceRequest.apxt :-

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

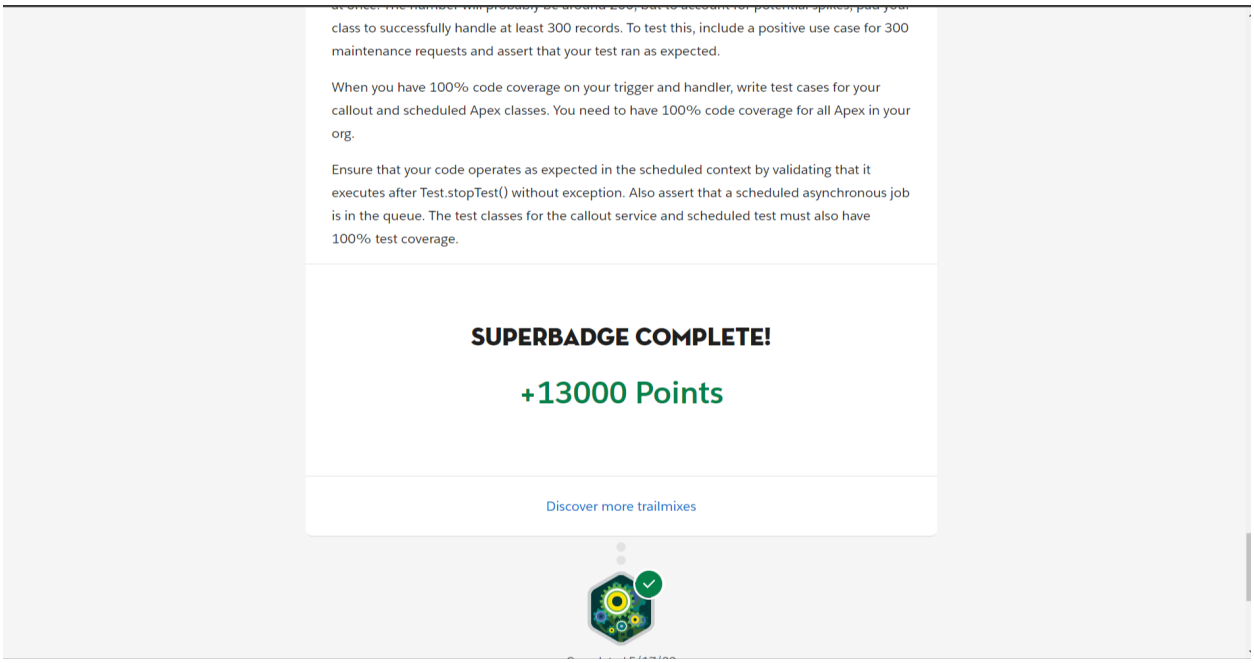
CHALLENGE 6: Test callout logic

WarehouseSyncSchedule.apxc :-

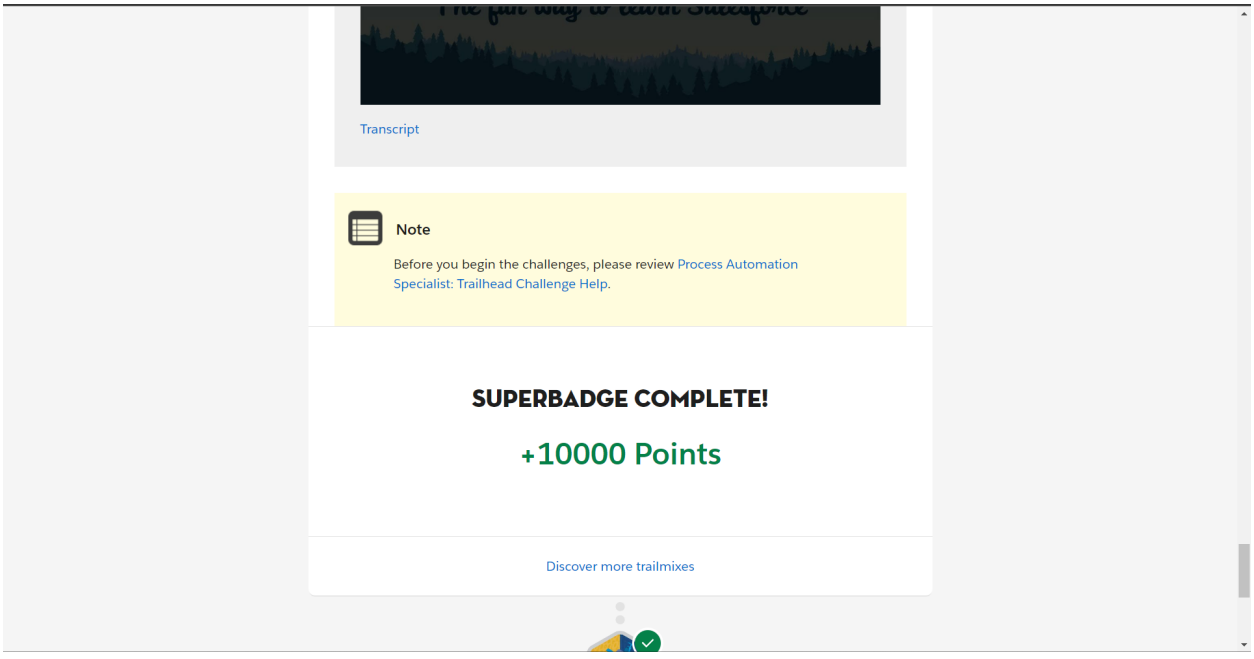
```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public class WarehouseSyncScheduleTest {

   @isTest static void WarehousescheduleTest(){
      String scheduleTime = '00 00 01 * * ?';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
      Test.stopTest();
      //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 ');


   }}
```



SCREENSHOT



SCREENSHOT