APEX SPECIALIST SUPERBADGE APEX CODES

Prerequisites that are done before doing tasks:

1. Installed this unlocked package (package ID: 04t6g000008av9iAAA).

2. Added picklist values Repair and Routine Maintenance to the Type field on the Case object.

3. Updated the Case page layout assignment to use the Case (HowWeRoll) Layout for my profile.

4. Renamed the tab/label for the Case tab to Maintenance Request.

5. Updated the Product page layout assignment to use the Product (HowWeRoll) Layout for my profile.

6. Renamed the tab/label for the Product object to Equipment.

7. Generated sample data through Create Default Data tab of the How We Roll Maintenance app.

Step - 2: Automate Recod Creation

/*MaintenanceRequest Trigger*/

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

Description: Created a trigger before or after an update request is taken place to the case which invokes updateWorkOrders method of MaintenanceRequestHelper class.

/*MaintenanceRequestHelper Class*/

```
public with sharing class MaintenanceRequestHelper {
```

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

      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

         AggregateResult[] results = [SELECT Maintenance_Request__c,
                           MIN(Equipment__r.Maintenance_Cycle__c)cycle
                           FROM Equipment_Maintenance_Item__c
                           WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

         List<Case> newCases = new List<Case>();
         for(Case cc : closedCases.values()){
            Case nc = new Case (
               ParentId = cc.Id,
               Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()
```

```
            );

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

Description: The parameters recieved from the MaintenanceRequest trigger are passed to updateworkOrders method wher it checks for new case status is closed or not. If the conditions are satisfied then it checks for the case type is 'Repair' or 'Routine Maintenance' and if it is within conditions it adds those case id's. Then after those id's were passed to check the due dates of the maintenance requests through maintenance cycle. If there are multiple equipment maintenance request define due date to shortest maintenance cycle and create those new case records.

Step - 3: Synchronize Salesforce data with an external system

/*WarehouseCalloutService Class*/

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';


    @future(callout=true)
```

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

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

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

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

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

```
        }
    }

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

}
```

Description: Implemented an Apex class WarehouseCalloutService that implements the queueable interface and makes a callout to the external service used for warehouse inventory management. This service receives updated values in the external system and updates the related records in Salesforce. Debugged the class by enqueueing the job to the method existed in WarehouseCalloutService class.

Step - 4: Schedule Synchronization

/*WarehouseSyncSchedule Class*/

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

Description: Created a scheduling logic that executes the callout WarehouseSyncSchedule and the scheduled job is named as WarehouseSyncScheduleJob.

Step - 6: Test automation logic

/*MaintenanceRequest Trigger*/

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

/*MaintenanceRequestHelper Class*/

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

      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

         AggregateResult[] results = [SELECT Maintenance_Request__c,
                           MIN(Equipment__r.Maintenance_Cycle__c)cycle
                           FROM Equipment_Maintenance_Item__c
                           WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

         List<Case> newCases = new List<Case>();
         for(Case cc : closedCases.values()){
            Case nc = new Case (
               ParentId = cc.Id,
               Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
```

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

            nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
          newCases.add(nc);
        }

        insert newCases;

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

/*MaintenanceRequestHelperTest Class*/

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
```

```
    }

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

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

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

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

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

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

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

```
        test.stopTest();

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

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

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assert(equals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }

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

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

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

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

```
        insert workP;

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

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

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

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

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

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

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

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

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,
```

```
caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

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

        list<case> newCase = [select id
                              from case
                              where status ='New'];



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

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

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

Description: Created the MaintenanceRequestHelperTest class to test the trigger actions for positive use case and negative use case. The trigger is fired on the positive use cases and on negative use cases the trigger is not fired. This MaintenanceRequestHelperTest class must test the trigger actions to handle upto 300 maintenance requests.

Step - 6: Test callout logic

/*WarehouseCalloutService Class*/

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
```

```
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

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

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

            if (product2List.size() > 0){
```

```
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
          }
        }
    }

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

}
```

/*WarehouseCalloutServiceMock Class*/

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request) {

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":
"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

Description: The WarehouseCalloutServiceMock implements the HttpCalloutMock and created a new HttpResponse object that stores the data of the callout request and header is set to JSON. The calllout is tested through the sample record of the WarehouseCalloutService through response body.

/*WarehouseCalloutServiceTest Class*/

```
@IsTest
private class WarehouseCalloutServiceTest {
        @isTest
   static void testWarehouseCallout() {
      test.startTest();
      test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      WarehouseCalloutService.runWarehouseEquipmentSync();
      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);
   }
}
```

Description: Test the WarehouseCalloutService by setting the mock to WarehouseCalloutServiceMock and run the runWarehouseEquipmentSync method. Stored the products into a list and checked the products from json data of the callout whether the two product code matches.

Step - 7: Test scheduling logic

/*WarehouseSyncScheduleTest Class*/

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
   @isTest static void test() {
      String scheduleTime = '00 00 00 * * ? *';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
      CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
```

```
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');
        Test.stopTest();
    }
}
```

APEX TRIGGERS AND CLASSES CODES

Get Started with Apex Triggers

```
trigger AccountAddressTrigger on Account (before insert,before update) {
          for(Account a : Trigger.New) {
        if (a.Match_Billing_Address__c==True) {
          a.BillingPostalCode = a.ShippingPostalCode;
        }
    }
}
```

Build Apex Triggers Unit

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

Get Started with Apex Unit Tests

```
@isTest
public class TestVerifyDate {
    @isTest static void testCheckDates(){
```

```
        Date dt = VerifyDate.CheckDates(date.parse('5/15/2022'), date.parse('5/20/2022'));
        System.assertEquals(date.parse('5/20/2022'), dt);
    }
    @isTest static void testCheckDatesNot(){
        Date dt = VerifyDate.CheckDates(date.parse('5/15/2022'), date.parse('8/20/2022'));
        System.assertEquals(date.parse('5/31/2022'), dt);
    }
    @isTest static void testDateWithin30Days()
    {
        Boolean dt = VerifyDate.DateWithin30Days(date.parse('5/15/2022'),
date.parse('5/19/2022'));
        System.assertEquals(True, dt);
    }
    @isTest static void testBackDate()
    {
        Boolean dt = VerifyDate.DateWithin30Days(date.parse('5/15/2022'),
date.parse('5/14/2022'));
        System.assertEquals(False, dt);
    }
    @isTest static void testGreater30Days()
    {
        Boolean dt = VerifyDate.DateWithin30Days(date.parse('5/15/2022'),
date.parse('9/19/2022'));
        System.assertEquals(False, dt);
    }
    @isTest static void testSetEndOfMonthDate(){
        Date dt = VerifyDate.SetEndOfMonthDate(date.parse('5/15/2022'));
        System.assertEquals(date.parse('5/31/2022'), dt);
    }
}
public class VerifyDate {

        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }

        public 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; }
        }

        public static Date SetEndOfMonthDate(Date date1) {
                Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
                Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
                return lastDay;
        }

}
```

Test Apex Triggers

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

        //check contacts prior to insert or update for invalid data
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }

        }



}

@isTest
public class TestRestrictContactByName {
   @isTest static void TestContactLastNameInsert(){
    Contact contact = new Contact();
    contact.LastName = 'INVALIDNAME';
    Test.startTest() ;
    Database.SaveResult result = Database.insert(contact, false);
    Test.stopTest();
    System.assert(!result.isSuccess()) ;
    System.assert(result.getErrors().size() > 0);
```

```
    System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

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;
      }
}
```

Use Future Methods

```
public class AccountProcessor {
        @future
   public static void countContacts(List<Id> accountIds) {

      List<Account> accounts = [SELECT Id, (SELECT Id FROM Contacts) FROM Account
                        WHERE Id IN :accountIds];

      for (Account acc : accounts) {
         acc.Number_of_Contacts__c = acc.Contacts.size();
      }

      update accounts;
}
}

@isTest
public class AccountProcessorTest {
         @isTest
   private static void countContactsTest() {

      List<Account> accounts = new List<Account>();
```

```
        for (Integer i=0; i<300; i++) {
            accounts.add(new Account(Name='Test Account' + i));
        }
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for (Account acc: accounts) {
            contacts.add(new Contact(FirstName=acc.Name, LastName='TestContact',
AccountId=acc.Id));
            accountIds.add(acc.Id);
        }
        insert contacts;

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

        List<Account> accs = [SELECT Id, Number_of_Contacts__c FROM Account];
        for (Account acc : accs) {
                        System.assertEquals(1, acc.Number_of_Contacts__c, 'ERROR: At least 1
Account record with incorrect Contact count');
        }
    }
}
```

Use Batch Apex

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

    public Integer recordCount = 0;

        public Database.QueryLocator start(Database.BatchableContext dbc){
                System.debug('Filling the bucket');
                return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
        }

        public void execute(Database.BatchableContext dbc, List<Lead> leads){
                for (Lead l : leads) {
                        l.LeadSource = 'Dreamforce';
                }
```

```
                update leads;
        recordCount = recordCount + leads.size();
                System.debug('Records processed so far ' + recordCount);
            }

            public void finish(Database.BatchableContext dbc){
        System.debug('Total records processed ' + recordCount);
            }
}

@isTest
private class LeadProcessorTest {

    @isTest
    private static void testBatchClass() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
            leads.add(new Lead(LastName='Connock', Company='Salesforce'));
        }
        insert leads;
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();
        List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 Lead record not updated
correctly');
    }
}
```

Control Processes with Queueble Apex

```
public class AddPrimaryContact implements Queueable {

    private Contact contact;
    private String state;
    public AddPrimaryContact(Contact inputContact, String inputState) {
        this.contact = inputContact;
        this.state = inputState;
    }
```

```
    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state LIMIT
200];
        List<Contact> contacts = new List<Contact>();
        for ( Account acc : accounts) {
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }
}

@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass() {
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<500; i++) {
            Account acc = new Account(Name='Test Account');
            if ( i<250 ) {
                acc.BillingState = 'NY';
            } else {
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact = new Contact(FirstName='Simon', LastName='Connock');
        insert contact;
        Test.startTest();
        Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));
        Test.stopTest();
        List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState =
'CA'];
        System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of Contact records
found');
    }
}
```

Schedule Jobs using the Apex Scheduler

```
public class DailyLeadProcessor implements Schedulable {

   public void execute(SchedulableContext ctx) {
      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT
200];
      for ( Lead l : leads) {
         l.LeadSource = 'Dreamforce';
      }
      update leads;
   }
}

@isTest
private class DailyLeadProcessorTest {

   private static String CRON_EXP = '0 0 0 ? * * *';

   @isTest
   private static void testSchedulableClass() {
      List<Lead> leads = new List<Lead>();
      for (Integer i=0; i<500; i++) {
         if ( i < 250 ) {
            leads.add(new Lead(LastName='Connock', Company='Salesforce'));
         } else {
            leads.add(new Lead(LastName='Connock', Company='Salesforce',
LeadSource='Other'));
         }
      }
      insert leads;
      Test.startTest();
      String jobId = System.schedule('Process Leads', CRON_EXP, new DailyLeadProcessor());
      Test.stopTest();
      List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =
'Dreamforce'];
      System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 record not updated
correctly');
      List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger
```

```
WHERE Id = :jobId];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);
    }
}
```

Apex REST Callouts

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

@isTest
private class AnimalLocatorTest {
    @isTest
    static void animallocatorTest1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = Animallocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
}
}

@istest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HttpResponse respond (HttpRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('contentType','application/json');
        response.setBody('{"animal":{"id":1,"name":"moose", "eats":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
```

```
}
}
```

Apex SOAP Callouts

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

```
            '',
            'http://parks.services/',
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
          response_x = response_map_x.get('response_x');
          return response_x.return_x;
      }
   }
}

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

Apex Web Services

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

```
}

@istest
private class AccountManagerTest {
   @isTest static void testGetAccount() {
      Account a = new Account (Name='TestAccount');
      insert a;
      Contact c = new Contact(AccountId=a.Id, FirstName='Test', LastName='Test');
      insert c;
      RestRequest request = new RestRequest();
      request.requestUri
='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.Id+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account myAcct = AccountManager.getAccount();
      System.assert (myAcct != null);
      System.assertEquals('TestAccount', myAcct.Name);
   }
}


trigger AccountDeletion on Account (before delete) {
        for (Account a : [SELECT Id FROM Account
              WHERE Id IN (SELECT AccountId FROM Opportunity) AND
              Id IN :Trigger.old]) {
      Trigger.oldMap.get(a.Id).addError(
        'Cannot delete account with related opportunities.');
   }
}

public class AsyncParkService {
   public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
      public String[] 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,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}

trigger HelloWorldTrigger on Account (before insert) {
            System.debug('Hello World!');
}

public class TaskUtil {
             public static String getTaskPriority(String leadState) {
        // Validate input
        if (String.isBlank(leadState) || leadState.length() > 2) {
            return null;
        }
        String taskPriority;
        if (leadState == 'CA') {
             taskPriority = 'High';
        } else {
             taskPriority = 'Normal';
        }
```

```
        return taskPriority;
    }
}
@isTest
public class TaskUtilTest {
            @isTest static void testTaskPriority() {
        String pri = TaskUtil.getTaskPriority('NY');
        System.assertEquals('Normal', pri);
    }
    @isTest static void testTaskHighPriority() {
        String pri = TaskUtil.getTaskPriority('CA');
        System.assertEquals('High', pri);
    }
    @isTest static void testTaskPriorityInvalid() {
        String pri = TaskUtil.getTaskPriority('Montana');
        System.assertEquals(null, pri);
    }
}
public class TemperatureConverter {
            public static Decimal FahrenheitToCelsius(Decimal fh) {
        Decimal cs = (fh - 32) * 5/9;
        return cs.setScale(2);
    }
}



@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create an account with an opportunity, and then try to delete it
        Account acct = new Account(Name='Test Account');
        insert acct;
        Opportunity opp = new Opportunity(Name=acct.Name + ' Opportunity',
                                          StageName='Prospecting',
                                          CloseDate=System.today().addMonths(1),
                                          AccountId=acct.Id);
        insert opp;
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(acct, false);
```

```
        Test.stopTest();
        // Verify
        // In this case the deletion should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related opportunities.',
                             result.getErrors()[0].getMessage());
    }
}
```