**Apex Specialist Superbadge :**

**APEX SPECIALIST :**

## Challenge 2 : Automate Record Creation

### MaintenanceRequest.cls

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

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

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

}

}

### MaintenanceRequestHelper.cls

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

Set<Id> validIds = new Set<Id>();

For (Case c : updWorkOrders){

if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

validIds.add(c.Id);

}

}

}

//When an existing maintenance request of type Repair or Routine Maintenance is closed,

//create a new maintenance request for 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 c FROM Equipment\_Maintenance\_Items r)

FROM Case WHERE Id IN :validIds]);

Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

AggregateResult[] results = [SELECT Maintenance\_Request c,

MIN(Equipment r.Maintenance\_Cycle c)cycle

FROM Equipment\_Maintenance\_Item c

WHERE Maintenance\_Request c IN :ValidIds GROUP BY Maintenance\_Request c];

for (AggregateResult ar : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request c'), (Decimal) ar.get('cycle'));

}

List<Case> newCases = new List<Case>();

for(Case cc : closedCases.values()){

Case nc = new Case (

ParentId = cc.Id,

Status = 'New',

Subject = 'Routine Maintenance',

Type = 'Routine Maintenance',

Vehicle c = cc.Vehicle c,

Equipment c =cc.Equipment c,

Origin = 'Web',

Date\_Reported c = Date.Today()

);

//If multiple pieces of equipment are used in the maintenance request,

//define the due date by applying the shortest maintenance cycle to today’s date.

If (maintenanceCycles.containskey(cc.Id)){

nc.Date\_Due c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

} else {

nc.Date\_Due c = Date.today().addDays((Integer) cc.Equipment r.maintenance\_Cycle c);

}

newCases.add(nc);

}

insert newCases;

List<Equipment\_Maintenance\_Item c> clonedList = new List<Equipment\_Maintenance\_Item c>();

for (Case nc : newCases){

for (Equipment\_Maintenance\_Item c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items r){

Equipment\_Maintenance\_Item c item = clonedListItem.clone();

item.Maintenance\_Request c = nc.Id;

clonedList.add(item);

}

}

insert clonedList;

}

}

}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);

}

}

}

//When an existing maintenance request of type Repair or Routine Maintenance is closed,

//create a new maintenance request for 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 c FROM Equipment\_Maintenance\_Items r)

FROM Case WHERE Id IN :validIds]);

Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

AggregateResult[] results = [SELECT Maintenance\_Request c,

MIN(Equipment r.Maintenance\_Cycle c)cycle

FROM Equipment\_Maintenance\_Item c

WHERE Maintenance\_Request c IN :ValidIds GROUP BY Maintenance\_Request c];

for (AggregateResult ar : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request c'), (Decimal) ar.get('cycle'));

}

List<Case> newCases = new List<Case>();

for(Case cc : closedCases.values()){

Case nc = new Case (

ParentId = cc.Id,

Status = 'New',

Subject = 'Routine Maintenance',

Type = 'Routine Maintenance',

Vehicle c = cc.Vehicle c,

Equipment c =cc.Equipment c,

Origin = 'Web',

Date\_Reported c = Date.Today()

);

//If multiple pieces of equipment are used in the maintenance request,

//define the due date by applying the shortest maintenance cycle to today’s date.

If (maintenanceCycles.containskey(cc.Id)){

nc.Date\_Due c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

} else {

nc.Date\_Due c = Date.today().addDays((Integer) cc.Equipment r.maintenance\_Cycle c);

}

newCases.add(nc);

}

insert newCases;

List<Equipment\_Maintenance\_Item c> clonedList = new List<Equipment\_Maintenance\_Item c>();

for (Case nc : newCases){

for (Equipment\_Maintenance\_Item c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items r){

Equipment\_Maintenance\_Item c item = clonedListItem.clone();

item.Maintenance\_Request c = nc.Id;

clonedList.add(item);

}

}

insert clonedList;

}

}

}

## Challenge-3 : Synchronize Salesforce data with an external system

### WarehouseCalloutService.cls

public with sharing class WarehouseCalloutService implements Queueable {

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

//Write a 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(){

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());

//class maps the following fields:

//warehouse SKU will be external ID for identifying which equipment records to update within Salesforce

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');

}

}

## Challenge 4 : Schedule synchronization

### WarehouseSyncShedule.cls

global with sharing class WarehouseSyncSchedule implements Schedulable{

global void execute(SchedulableContext ctx){

System.enqueueJob(new WarehouseCalloutService());

}

}

## Challenge 5 : Test automation logic

### MaintenanceRequest.cls

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

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

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

}

}

### MaintenanceRequestHelper.cls

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

Set<Id> validIds = new Set<Id>();

For (Case c : updWorkOrders){

if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

validIds.add(c.Id);

}

}

}

//When an existing maintenance request of type Repair or Routine Maintenance is closed,

//create a new maintenance request for 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 c FROM Equipment\_Maintenance\_Items r)

FROM Case WHERE Id IN :validIds]);

Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

AggregateResult[] results = [SELECT Maintenance\_Request c,

MIN(Equipment r.Maintenance\_Cycle c)cycle

FROM Equipment\_Maintenance\_Item c

WHERE Maintenance\_Request c IN :ValidIds GROUP BY Maintenance\_Request c];

for (AggregateResult ar : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request c'), (Decimal) ar.get('cycle'));

}

List<Case> newCases = new List<Case>();

for(Case cc : closedCases.values()){

Case nc = new Case (

ParentId = cc.Id,

Status = 'New',

Subject = 'Routine Maintenance',

Type = 'Routine Maintenance',

Vehicle c = cc.Vehicle c,

Equipment c =cc.Equipment c,

Origin = 'Web',

Date\_Reported c = Date.Today()

);

//If multiple pieces of equipment are used in the maintenance request,

//define the due date by applying the shortest maintenance cycle to today’s date.

//If (maintenanceCycles.containskey(cc.Id)){

nc.Date\_Due c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

//} else {

// nc.Date\_Due c = Date.today().addDays((Integer) cc.Equipment r.maintenance\_Cycle c);

//}

newCases.add(nc);

}

insert newCases;

List<Equipment\_Maintenance\_Item c> clonedList = new List<Equipment\_Maintenance\_Item c>();

for (Case nc : newCases){

for (Equipment\_Maintenance\_Item c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items r){

Equipment\_Maintenance\_Item c item = clonedListItem.clone();

item.Maintenance\_Request c = nc.Id;

clonedList.add(item);

}

}

insert clonedList;

}

}

}

### MaintenanceRequestHelperTest.cls

@isTest

public with sharing class MaintenanceRequestHelperTest {

// createVehicle

private static Vehicle c createVehicle(){

Vehicle c vehicle = new Vehicle C(name = 'Testing Vehicle');

return vehicle;

}

// createEquipment

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;

}

// createMaintenanceRequest

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;

}

// createEquipmentMaintenanceItem

private static Equipment\_Maintenance\_Item c createEquipmentMaintenanceItem(id equipmentId,id requestId){

Equipment\_Maintenance\_Item c equipmentMaintenanceItem = new Equipment\_Maintenance\_Item c(

Equipment c = equipmentId,

Maintenance\_Request c = requestId);

return equipmentMaintenanceItem;

}

@isTest

private static void testPositive(){

Vehicle c vehicle = createVehicle();

insert vehicle;

id vehicleId = vehicle.Id;

Product2 equipment = createEquipment();

insert equipment;

id equipmentId = equipment.Id;

case createdCase = createMaintenanceRequest(vehicleId,equipmentId);

insert createdCase;

Equipment\_Maintenance\_Item c equipmentMaintenanceItem = createEquipmentMaintenanceItem(equipmentId,createdCase.id);

insert equipmentMaintenanceItem;

test.startTest();

createdCase.status = 'Closed';

update createdCase;

test.stopTest();

Case newCase = [Select id, subject,

type, Equipment c,

Date\_Reported c,

Vehicle c, Date\_Due c from case

where status ='New'];

Equipment\_Maintenance\_Item c workPart = [select id

from Equipment\_Maintenance\_Item c

where Maintenance\_Request c =:newCase.Id];

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

system.assert(allCase.size() == 2);

system.assert(newCase != null);

system.assert(newCase.Subject != null);

system.assertEquals(newCase.Type, 'Routine Maintenance');

SYSTEM.assertEquals(newCase.Equipment c, equipmentId);

SYSTEM.assertEquals(newCase.Vehicle c, vehicleId);

SYSTEM.assertEquals(newCase.Date\_Reported c, system.today());

}

@isTest

private static void testNegative(){

Vehicle C vehicle = createVehicle();

insert vehicle;

id vehicleId = vehicle.Id;

product2 equipment = createEquipment();

insert equipment;

id equipmentId = equipment.Id;

case createdCase = createMaintenanceRequest(vehicleId,equipmentId);

insert createdCase;

Equipment\_Maintenance\_Item c workP = createEquipmentMaintenanceItem(equipmentId, createdCase.Id);

insert workP;

test.startTest();

createdCase.Status = 'Working';

update createdCase;

test.stopTest();

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

Equipment\_Maintenance\_Item c equipmentMaintenanceItem = [select id

from Equipment\_Maintenance\_Item c

where Maintenance\_Request c = :createdCase.Id];

system.assert(equipmentMaintenanceItem != null);

system.assert(allCase.size() == 1);

}

@isTest

private static void testBulk(){

list<Vehicle C> vehicleList = new list<Vehicle C>();

list<Product2> equipmentList = new list<Product2>();

list<Equipment\_Maintenance\_Item c> equipmentMaintenanceItemList = new list<Equipment\_Maintenance\_Item c>();

list<case> caseList = new list<case>();

list<id> oldCaseIds = new list<id>();

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

vehicleList.add(createVehicle());

equipmentList.add(createEquipment());

}

insert vehicleList;

insert equipmentList;

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

caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));

}

insert caseList;

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

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.ge t(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);

}

}

## Challenge-6 : Test callout logic

### WarehouseCalloutService.cls

public with sharing class WarehouseCalloutService implements Queueable {

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

//Write a 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(){

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());

//class maps the following fields:

//warehouse SKU will be external ID for identifying which equipment records to update within Salesforce

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.cls

@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, "name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"55d6622672 6b611100aaf742","replacement":true,"quantity":183,"name":"Cooling

Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611 100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

response.setStatusCode(200);

return response;

}

}

### WarehouseCalloutServiceTest.cls

@IsTest

private class WarehouseCalloutServiceTest {

// implement your mock callout test here

@isTest

static void testWarehouseCallout() {

test.startTest();

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

WarehouseCalloutService.execute(null);

test.stopTest();

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

System.assertEquals(3, product2List.size());

}

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

}

## Challenge 7 : Test scheduling logic

### WarehouseCalloutServiceMock.cls

@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, "name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"55d6622672

6b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611 100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

response.setStatusCode(200);

return response;

}

}

### WarehouseSyncShedule.cls

global with sharing class WarehouseSyncSchedule implements Schedulable {

// implement scheduled code here

global void execute (SchedulableContext ctx){

System.enqueueJob(new WarehouseCalloutService());

}

}

### WarehouseSyncSheduleTest.cls

@isTest

public with sharing class WarehouseSyncScheduleTest {

// implement scheduled code here

//

@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 Testing

### Challenge : Get Started with Apex Unit Tests VerifyDate.cls

public class VerifyDate {

//method to handle potential checks against two dates public static Date CheckDates(Date date1, Date date2) {

//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of

the month

if(DateWithin30Days(date1,date2)) { return date2;

} else {

}

}

return SetEndOfMonthDate(date1);

//method to check if date2 is within the next 30 days of date1

@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {

//check for date2 being in the past if( date2 < date1) { return false; }

if( date2 < date1) { return false; }

//check that date2 is within (>=) 30 days of date1

Date date30Days = date1.addDays(30); //create a date 30 days away from date1

//check that date2 is within (>=) 30 days of date1

// Date date30Days = date1.addDays(30); //create a date 30 days away from date1 if( date2 >= date30Days ) { return false; }

else { return true; }

}

//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); return lastDay;

}

}

### TestVerifyDate .cls

@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('02/02/2019'));

System.assertEquals(false, flag);

}

@isTest static void Test\_DateWithin30Days\_case3(){

Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('01/15/2019'));

System.assertEquals(false, flag);

}

@isTest static void Test\_SetEndOfMonthDate(){

Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

}

}

**Challenge : Test Apex Triggers**

**RestrictContactByName.cls**

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');

}

}

}

**TestRestrictContactByName .cls**

@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 allowed for DML', result.getErrors()[0].getMessage());

}

}

## Challenge : Create Test Data for Apex Tests

### RandomContactFactory .cls

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;

}

}

# Apex Triggers

## Challenge : Get Started with Apex Triggers

### AccountAddressTrigger.cls

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

for(Account account:Trigger.New){

if(account.Match\_Billing\_Address c == True){

account.ShippingPostalCode = account.BillingPostalCode;

}

}

}

## Challenge : Bulk Apex Triggers

### ClosedOpportunityTrigger.cls

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) { List<Task> taskList = new List<Task>();

for(Opportunity opp: Trigger.New){ if(opp.StageName == 'Closed Won'){

taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));

}

}

if(taskList.size()>0){ insert taskList;

}

}

# Asynchronous Apex :

## Challenge : Use Future Methods

### AccountProcessor .cls

public class AccountProcessor { @future

public static void countContacts(List<Id> accountIds){ List<Account> accountsToUpdate = new List<Account>();

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

:accountIds];

for(Account acc:accounts){

List<Contact> contactList = acc.Contacts; acc.Number\_Of\_Contacts c = contactList.size(); accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

### AccountProcessorTest .cls

@IsTest

private class AccountProcessorTest {

@IsTest

private static void testCountContacts(){

Account newAccount = new Account(Name='Test Account');

insert newAccount;

Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =

newAccount.Id);

insert newContact1;

Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =

newAccount.Id);

insert newContact2;

List<Id> accountIds = new List<Id>();

accountIds.add(newAccount.Id);

Test.startTest();

AccountProcessor.countContacts(accountIds);

Test.stopTest();

}

}

## Challenge : Use Batch Apex

### LeadProcessor .cls

global class LeadProcessor implements Database.Batchable<sObject>{

global Integer count = 0;

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

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

}

global void execute (Database.BatchableContext bc, List<Lead> L\_list){

List<Lead> L\_List\_new = new List<Lead>();

for(lead L:L\_list){

L.leadsource = 'Dreamforce';

L\_list\_new.add(L);

count += 1;

}

update L\_list\_new;

}

global void finish(Database.BatchableContext bc){

system.debug('count = ' +count);

}

}

### LeadProcessorTest.cls

@isTest

public class LeadProcessorTest {

@isTest

public static void testify(){

List<lead> L\_list = new List<lead>();

for(Integer i=0; i<200; i++){

Lead L = new lead();

L.LastName = 'name' +i;

L.Company = 'Company';

L.Status = 'Random Status';

L\_list.add(L);

}

insert L\_list;

Test.startTest();

LeadProcessor lp = new LeadProcessor();

Id batchId = Database.executeBatch(lp);

Test.stopTest();

}

}

## Challenge : Control Processes with Queueable Apex

### AddPrimaryContact.cls

public class AddPrimaryContact implements Queueable {

private Contact con;

private String state;

public AddPrimaryContact(Contact con, String state){

this.con = con;

this.state = state;

}

public void execute(QueueableContext context){

List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)

from Account where BillingState = :state Limit 200];

List<Contact> primaryContacts = new List<Contact>();

for(Account acc:accounts){

Contact c = con.clone();

c.AccountId = acc.Id;

primaryContacts.add(c);

}

if(primaryContacts.size() > 0){

insert primaryContacts;

}

}

}

### AddPrimaryContactTest.cls

@isTest

public class AddPrimaryContactTest {

static testmethod void testQueueable(){

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

for(Integer i=0;i<50;i++){

testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));

}

for(Integer j=0;j<50;j++){

testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));

}

insert testAccounts;

Contact testContact = new Contact(FirstName = 'John', LastName = 'Doe');

insert testContact;

AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

Test.startTest();

system.enqueueJob(addit);

Test.stopTest();

System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from

Account Where BillingState='CA')]);

}

}

## Challenge : Schedule Jobs Using the Apex Scheduler

### DailyLeadProcessor .cls

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';

leadstoupdate.add(l);

}

update leadstoupdate;

}

}

### DailyLeadProcessorTest .cls

@isTest

private class DailyLeadProcessorTest {

public static String CRON\_EXP = '0 0 0 15 3 ? 2022';

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();

DailyLeadProcessor ab = new DailyLeadProcessor();

String jobId = System.schedule('jobName', '0 5 \* \* \* ?', ab);

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 Integration Services

## Challenge : Apex REST Callouts

### AnimalLocator .cls

public class AnimalLocator {

public static string getAnimalNameById(Integer id) {

Http http = new Http();

HttpRequest req = new HttpRequest();

req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);

req.setMethod('GET');

Map<String,Object> animals = new Map<String,Object>();

HttpResponse res = http.send(req);

// If the request is successful, parse the JSON response.

if(res.getStatusCode() == 200) {

Map<String,Object> results = (Map<String,Object>)

JSON.deserializeUntyped(res.getBody());

animals = (Map<String,Object>)results.get('animal');

}

else{

System.debug('The status code returned was not expected:'+res.getStatusCode()+''+res.getStatus());

}

return (string)animals.get('name');

}

}

### AnimalLocatorTest .cls

@isTest

private class AnimalLocatorTest{

@isTest static void AnimalsCalloutsTest(){

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

String result = AnimalLocator.getAnimalNameById(1);

String expectedValue = 'chicken';

System.assertEquals(result,expectedValue);

}

}

### AnimalLocatorMock .cls

@isTest

global class AnimalLocatorMock implements HttpCalloutMock{

// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

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

response.setBody('{"animals":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');

//response.setBody('{"animals":{"id":1,"name":"chicken"}}');

response.setStatusCode(200);

return response;

}

}

### AnimalsHttpCalloutMock .cls

@isTest

global class AnimalsHttpCalloutMock implements HttpCalloutMock {

// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

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

response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');

response.setStatusCode(200);

return response;

}

}

## Challenge : Apex SOAP Callouts

### ParkService.cls

//Generated by wsdl2apex

public class ParkService {

public class byCountryResponse { public String[] return\_x;

private String[] return\_x\_type\_info = new String[]{'return',['h](http://parks.services/%27%2Cnull%2C%270%27%2C%27-)t[tp://parks.services/',null,'0','-](http://parks.services/%27%2Cnull%2C%270%27%2C%27-) 1','false'};

private String[] apex\_schema\_type\_info = new String[]{['http://parks](http://parks.services/%27%2C%27false%27%2C%27false%27).[services/','false','false'](http://parks.services/%27%2C%27false%27%2C%27false%27)};

private String[] field\_order\_type\_info = new String[]{'return\_x'};

}

public class byCountry { public String arg0;

private String[] arg0\_type\_info = new String[]{'arg0',['h](http://parks.services/%27%2Cnull%2C%270%27%2C%271%27%2C%27false%27)t[tp://parks.services/',null,'0','1','false'](http://parks.services/%27%2Cnull%2C%270%27%2C%271%27%2C%27false%27)};

private String[] apex\_schema\_type\_info = new String[]{['http://parks](http://parks.services/%27%2C%27false%27%2C%27false%27).[services/','false','false'](http://parks.services/%27%2C%27false%27%2C%27false%27)};

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[][{'h](http://parks.services/%27)tt[p://parks.services/',](http://parks.services/%27) '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/',](http://parks.services/%27) 'byCountry', '[http://parks.services/',](http://parks.services/%27) 'byCountryResponse',

'ParkService.byCountryResponse'}

);

response\_x = response\_map\_x.get('response\_x'); return response\_x.return\_x;

}

}

}

**ParkLocator.cls**

public class ParkLocator {

public static List<String> country(String country){ ParkService.ParksImplPort parkservice =

new parkService.ParksImplPort(); return parkservice.byCountry(country);

}

}

### ParkLocatorTest.cls

@isTest

private class ParkLocatorTest { @isTest static void testCallout() {

// This causes a fake response to be generated Test.setMock(WebServiceMock.class, new ParkServiceMock());

// Call the method that invokes a callout String country = 'United States';

List<String> result = ParkLocator.country(country); List<String> parks = new List<String>();

parks.add('yosemite'); parks.add('Yellowstone'); parks.add('Another Park');

// Verify that a fake result is returned System.assertEquals(parks, result);

}

}

### ParkServiceMock.cls

@isTest

global class ParkServiceMock implements WebServiceMock { global void doInvoke(

Object stub, Object request,

Map<String, Object> response, String endpoint,

String soapAction, String requestName, String responseNS, String responseName, String responseType) {

// start - specify the response you want to send List<String> parks = new List<String>();

parks.add('yosemite'); parks.add('Yellowstone'); parks.add('Another Park');

ParkService.byCountryResponse response\_x = new ParkService.byCountryResponse();

response\_x.return\_x = parks;

// end

response.put('response\_x', response\_x);

}

}

## Challenge : Apex Web Services

### AccountManager .cls

@RestResource(urlMapping = '/Accounts/\*/contacts') global with sharing class AccountManager {

@HttpGet

global static Account getAccount(){

RestRequest request = RestContext.request;

string accountId = request.requestURI.substringBetween('Accounts/','/contacts');

Account result = [SELECT Id,Name, (Select Id, Name from Contacts) from Account where Id=:accountId Limit 1];

return result;

}

}

### AccountManagerTest .cls

@isTest

public class AccountManagerTest {

@isTest static void testGetContactByAccountId(){

Id recordId = createTestRecord();

RestRequest request = new RestRequest();

request.requestUri = 'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'

+ recordId+'/contacts';

request.httpMethod = 'GET';

RestContext.request = request;

Account thisAccount = AccountManager.getAccount();

System.assert(thisAccount != null);

System.assertEquals('Test record', thisAccount.Name);

}

static Id createTestRecord(){

Account accountTest = new Account(

Name = 'Test record');

insert accountTest;

Contact contactTest = new Contact(

FirstName = 'John',

LastName = 'Doe',

AccountId = accountTest.Id

);

insert contactTest;

return accountTest.Id;

}

}