APEX TRIGGERS

Get Started With Apex

AccountAddressTrigger.apxt

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

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

Bulk Apex Triggers

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (afterinsert, after update){ List <Task>
todoList = new List <Task>();

for (Opportunity opp :Trigger.new){ if(Trigger.isInsert || Trigger.isUpdate) {
if(opp.StageName == 'Closed Won') {
todoList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
}
}
}
if(todoList.size()>0) { insert todoList;
    }
 }
```

APEX TESTING

Get Started With Apex Unittest

VerifyDate.apxc

```
public class VerifyDate {
/ method to handle potentialchecks 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 privatestatic Boolean
DateWithin30Days(Date date1,Date date2) {
/ check for date2 being in the past 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 if( date2
>= date30Days) { return false; }
else { return true; }
}

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


 }
```

TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate { @isTest static void Test1(){
Date d = VerifyDate.CheckDates(Date.parse('05/17/2022'),Date.parse('05/21/2022'));
System.assertEquals(Date.parse('05/21/2022'),d);
    }
@isTest static void Test2() {
Date d = VerifyDate.CheckDates(Date.parse('05/17/2022'),Date.parse('06/21/2022'));
System.assertEquals(Date.parse('06/21/2022'),d);
    }
 }
```

Test Apex Triggers

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (beforeinsert, before update){ For (Contact c :
Trigger.New)
{
if(c.LastName == 'INVALIDNAME')
{
c.AddError('The Last Name "'+c.LastName+'" is not allowedfor DML');
                        }


             }
  }
```

TestRestrictContactByName.apxc

```
    @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());
     }
```

Create Test Data for Apex Tests RandomContactFactory.apxc

```
  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;
        }
        }
```

ASYNCHRONUS APEX

Use Future Methods

AccountProcessor.apxc

```
public classAccountProcessor
{
@future
public staticvoid countContacts(Set<id> setId)
{
List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from contacts)
from account where id in :setId ];
for( Account acc : lstAccount)


{
List<Contact> lstCont= acc.contacts ;

acc.Number_of_Contacts__c = lstCont.size();
}
update lstAccount;
   }
  }
```

AccountProcessorTest.apxc

```
@IsTest
public class AccountProcessorTest {
public static testmethod void TestAccountProcessorTest(){ Accounta = new Account();
a.Name = 'Test Account'; Inserta;
Contact cont = New Contact(); cont.FirstName ='Bob'; cont.LastName ='Masters';
cont.AccountId = a.Id;
Insert cont;

set<Id> setAccId = new Set<ID>(); setAccId.add(a.id);

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

Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT 1];
System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
  }


 }
```

Uses Batch Apex

LeadProcessor.apxc

```
global class LeadProcessor implements Database.Batchable<sObject> {
  global Integer count = 0;


  global Database.QueryLocator start (Database.BatchableContext bc) {
    return Database.getQueryLocator('Select Id, LeadSource from lead');
  }


  global void execute (Database.BatchableContext bc,List<Lead> l_lst) {
    List<lead> l_lst_new = new List<lead>();
    for(lead l : l_lst) {
      l.leadsource = 'Dreamforce';
      l_lst_new.add(l);
      count+=1;
    }
    update l_lst_new;
  }

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

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest
 {
```

```
static testMethod void testMethod1()
    {
List<Lead> lstLead = new List<Lead>(); for(Integer i=0 ;i <200;i++)
      {
Lead led = new Lead(); led.FirstName ='FirstName'; led.LastName ='LastName'+i;
led.Company ='demo'+i; lstLead.add(led);
      }

insert lstLead; Test.startTest();
LeadProcessor obj = new LeadProcessor(); DataBase.executeBatch(obj) ;

Test.stopTest();
    }
  }
```

Control Processeswith Queueable Apex

AddPrimaryContact.apcx

```
public class AddPrimaryContact implements Queueable
{
private Contact c; private String state;
public AddPrimaryContact(Contact c, String state)
{
this.c = c;

this.state = state;
}
public void execute(QueueableContext context)
{
List<Account> ListAccount = [SELECT ID, Name ,(Selectid,FirstName,LastName from
contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
List<Contact> lstContact = new List<Contact>(); for (Account acc:ListAccount)
       {
Contact cont = c.clone(false,false,false,false); cont.AccountId = acc.id;
lstContact.add( cont );
       }

if(lstContact.size() >0 )
       {
insert lstContact;
       }


    }
```

```
 }
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
@isTest static void TestList()
{
List<Account> Teste = new List <Account>(); for(Integer i=0;i<50;i++)
{
Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
}
for(Integer j=0;j<50;j++)
{
Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
}
insert Teste;

Contact co = new Contact(); co.FirstName='demo'; co.LastName ='demo';insert co;
String state = 'CA';

AddPrimaryContact apc = new AddPrimaryContact(co, state);


Test.startTest(); System.enqueueJob(apc); Test.stopTest();
      }
  }
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable{ globalvoid
execute(SchedulableContext ctx){
List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){
List<Lead> newLeads= new List<Lead>();


for(Lead lead : leads){ lead.LeadSource = 'DreamForce'; newLeads.add(lead);
          }

update newLeads;
       }
```

```
    }
  }
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest{
/ Seconds MinutesHours Day_of_month MonthDay_of_week optional_year publicstatic
String CRON_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){ List<Lead> leads = new List<Lead>();

for(Integer i = 0; i < 200; i++){
Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' +
i, Status = 'Open - Not Contacted');
leads.add(lead);
}
insert leads; Test.startTest();
/ Schedulethe test job
String jobId = System.schedule('Update LeadSourceto DreamForce', CRON_EXP,new
DailyLeadProcessor());

/ Stopping the test will run the job synchronously Test.stopTest();
    }
  }
```

APEX INTEGRATION SERVICES

Apex REST Callouts

AnimalLocator.apxc

```
public class AnimalLocator
{
public static String getAnimalNameById(Integer id)
{
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
request.setMethod('GET');
HttpResponse response = http.send(request);
String strResp = '';
system.debug('******response '+response.getStatusCode());
system.debug('******response '+response.getBody());
// If the request is successful, parse the JSON response.
if (response.getStatusCode() == 200)
{
// Deserializes the JSON string into collections of primitive data types.
Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
// Cast the values in the 'animals' key as a list
Map<string,object> animals = (map<string,object>) results.get('animal');
System.debug('Received the following animals:' + animals );
strResp = string.valueof(animals.get('name'));
System.debug('strResp >>>>>>' + strResp );
}
return strResp ;
}
}
```

AnimalLocatorTest.apxc

```
@isTest
public class AnimalLocatorTest {
   @isTest
   public static void testGetAnimalNameById() {
      AnimalLocatorMock mock = new AnimalLocatorMock();
      // Associate the callout with a mock response
      Test.setMock(HttpCalloutMock.class, mock);
      // Call method to test
      String animalName = AnimalLocator.getAnimalNameById(1);
      // Verify we got a chicken
      System.assertEquals('chicken', animalName,
      'The name of the test animal should be a chicken.');
   }
}
```

AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   // Implement this interface method
   global HTTPResponse respond(HTTPRequest request) {
      // Create a fake response
      HttpResponse response = new HttpResponse();
      response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
      response.setStatusCode(200);
      return response;
   }
}
```

Apex SOAP Callouts

ParkService.apxc

```
//Generated by wsdl2apex

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

ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

ParkServiceMock.apxc

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
          Object stub,
          Object request,
          Map<String, Object> response,
          String endpoint,
          String soapAction,
          String requestName,
          String responseNS,
          String responseName,
          String responseType) {
       ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
       List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
       response_x.return_x = lstOfDummyParks;

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

AsyncParksServices.apxc

```
//Generated by wsdl2apex

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'}
          );
        }
    }
}
```

Apex Web Services

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts') Global with sharing class
AccountManager {
@HttpGet
global static Account getAccount(){ RestRequest request = RestContext.request;
/ Grab the accountId from end of URL
String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
Accountacc = [select Id,Name,(select Id,Namefrom Contacts) from Account where Id =
:accountId];
system.debug('Account and RelatedContacts->>>>'+acc); return acc;
    }
  }
AccountManagerTest.apxc @isTest
private class AccountManagerTest {
/ Helper method to create dummy record static Id createTestRecord(){
/ Create test record
Account TestAcc = new Account(Name='Test Account', Phone='8786757657');
insertTestAcc;
List<Contact> conList = new List<Contact>(); ContactTestCon = new Contact(); for(Integer
i=1;i<=3;i++){
TestCon.LastName = 'Test Contact'+i; TestCon.AccountId = TestAcc.Id;
/ conList.add(TestCon);
insert conList;/Its not best practice but I have use it for testingpurposes
}
/ insert conList;
/ insert TestAcc; returnTestAcc.Id;
}
/ Method to test getAccount() @isTest static void getAccountTest(){
Id recordId = createTestRecord();
/ setup a test request

RestRequest request= new RestRequest();
/ set requestproperties request.requestURI =
'https:/ yourInstance.salesforce.com/services/apexrest/Accounts/' + recordId +'/contacts';
request.httpMethod = 'GET';
/ Finally, assign the request to RestContext if used RestContext.request = request;
/ End test setup

/ Call the method
Account thisAcc= AccountManager.getAccount();
/ Verify the result system.assert(thisAcc != null);
system.assertEquals('Test Account', thisAcc.Name);
/ system.assertEquals(3, thisAcc.Contact__c.size()); how to get this
    }
  }
```

APEX SPECIALIST SUPERBADGE

APEX SPECIALIST SUPERBADGE

Challenge 2 AutomatedRecord 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);
            }
        }
    }
    //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, Equipmentc,
Equipmentr.Maintenance_Cycle_c,
                                                (SELECT Id,Equipment_c,Quantityc 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.Equipmentr.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
      }
   }
}
```

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';
   //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 using Apex code

WarehouseSyncShedule.apxc :-

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

Challenge 5:   Test automationlogic

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);
            }
         }
      }
      //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, Equipmentc,
Equipmentr.Maintenance_Cycle_c,
                                        (SELECT Id,Equipment_c,Quantityc 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.Equipmentr.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;
    }
  }
}
```

MaintenanceRequest.apxc :-

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

MaintenanceRequestHelperTest.apxc :-

```
@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.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);
    }
}
```

Challenge 6

Test callout logic

WarehouseCalloutService.apxc :-

```
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');
    }
}
```

WarehouseCalloutServiceTest.apxc :-

```
@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);
   }
}
```

WarehouseCalloutServiceMock.apxc :-

```
@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 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;
   }
}
```

Challenge 7
Test schedulinglogic

WarehouseSyncSchedule.apxc :-

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

WarehouseSyncScheduleTest.apxc :-

```
@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();
   }
}
```