SPSGP - 11410 - Salesforce Developer Catalyst

Self - Learning & Super Badges

APEX TRIGGERS

AccountAddressTrigger.axpt:

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

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

}
```

ClosedOpportunityTrigger.axpt:

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

```
        insert tasklist;
    }

}
```

APEX TESTING

VerifyDate.apxc:

```
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
        private static 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
private class TestVerifyDate{
   @isTest static void CheckDatesTesttrue(){
      Date date1=date.today();
      Date date2=date1.addDays(29);
      Date t = VerifyDate.CheckDates(date1,date2);
      System.assertEquals(t, date2);
   }
   @isTest static void dateOver(){
      Date date1=date.today();
      Date date2=date1.addDays(31);
      date t=VerifyDate.CheckDates(date1,date2);
      System.assertNotEquals(t,date1);
   }
}
```

RestrictContactByName.apxt:

```
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.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());
```

```
    }
}
```

RandomContactFactory.apxt:

```
public class RandomContactFactory {

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

        }
        return contacts;
    }
}
```

ASYNCHRONOUS APEX

AccountProcessor.apxc:

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

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

```
        List<Account> accounts = [Select Id,Name,(Select Id from Contacts)
from Account Where Id in :accountIds];
        for (Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c=contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }

}
```

AccountProcessorTest.apxc:

```
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;

         Contact newContact1=new
Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
         insert newContact1;


         Contact newContact2 =new
```

```
Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);



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

}
```

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 {
    @isTest
    public static void testit(){
        List<lead> l_lst = 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_lst.add(l);
        }
        insert l_lst;

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

}
```

AddPrimaryContact.apxc:

```
public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    Public AddPrimaryContact(Contact con,String state){
        this.con = con;
        this.state=state;

    }

    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id,Name,(Select
```

```
FirstName,LastName, Id from contacts)
                        from Account where BillingState = :state Limit
200];
      List<Contact> primaryContacts = new List<Contact>();

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

      }
      }
   }
```

AddPrimaryContactTest.apxc:

```
@isTest
public class AddPrimaryContactTest {

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

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

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

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

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

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

}
```

DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable{
```

```
  global void execute(SchedulableContext sc){
    List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE
LeadSource = null LIMIT 200];
    List<Lead> lstOfUpdatedLead = new List<Lead>();
    if(!lstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
        ld.LeadSource = 'Dreamforce';
        lstOfUpdatedLead.add(ld);
      }
      UPDATE lstOfUpdatedLead;
    }
  }
}
```

DailyLeadProcessorTest.apxc:

```
@isTest
 private class DailyLeadProcessorTest{
  @testSetup
  static void setup(){
    List<Lead> listOfLead = new List<Lead>();
    for(Integer i = 1; i <= 200; i++){
      Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i,
Status = 'Working - Contacted');
      listOfLead.add(ld);
    }
    Insert listOfLead;
```

```
  }
  static testmethod void testDailyLeadProcessorScheduledJob(){
    String sch = '0 5 12 * * ?';
    Test.startTest();
    String jobId = System.schedule('ScheduledApexTest', sch, new
DailyLeadProcessor());
    List<Lead> listOfLead = [SELECT Id FROM Lead WHERE
LeadSource = null LIMIT 200];
    System.assertEquals(200, listOfLead.size());
    Test.stopTest();
  }
 }
```

APEX INTEGRATION SERVICES

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
private class AnimalLocatorTest{
   @isTest static  void AnimalLocatorMock1() {
      Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());
      string result=AnimalLocator.getAnimalNameById(3);
      string expectedResult='cow';
      System.assertEquals(result, expectedResult);
   }
```

```
}
```

AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

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);
    }
}
```

AccountManager.apxc:

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

AccountManagerTest.apxc:

```
@isTest
```

```
private class AccountManagerTest {
   @isTest static void testAccountManager() {
      Id recordId = createTestRecord();

      Contact Cont1 = new Contact();
      Cont1.LastName = 'ABC';
      Cont1.AccountId = recordId ;
      insert Cont1;

      Contact Cont2 = new Contact();
      Cont2.LastName = 'XYZ';
      Cont2.AccountId = recordId ;
      insert Cont2 ;

      RestRequest request = new RestRequest();
      request.requestUri =
         'https://playful-badger-8yy3gf-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
     Account lst = AccountManager.getAccount();
     /* Tried to return Contact List but failed Trailhead Verification
     List<Contact> lst = new List<Contact>();
     lst = AccountManager.getAccount();
     */
   }
static Id createTestRecord() {
      Account accTest = new Account(Name='Test Record');
      insert accTest ;
      return accTest.Id;
   }
  }
```

APEX SPECIALIST SUPER BADGE

Challenge-1

MaintenanceRequestHelper.apxc:

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


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



            }
         }
      }

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

```
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN
:ValidIds GROUP BY Maintenance_Request__c];

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

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

            );

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

            newCases.add(nc);
         }

        insert newCases;
```

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

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaintenanceRequest.apxt:

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

MaintenanceRequestHelperTest.apxc:

```
@istest
public with sharing class MaintenanceRequestHelperTest {

  private static final string STATUS_NEW = 'New';
  private static final string WORKING = 'Working';
```

```
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

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

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

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

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
```

```
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                    Maintenance_Request__c =
requestId);
        return wp;
    }


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

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

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

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

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

        Case newReq = [Select id, subject, type, Equipment__c,
```

```
Date_Reported__c, Vehicle__c, Date_Due__c
                from case
                where status =:STATUS_NEW];

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

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

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

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

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

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

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

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

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

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

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

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

```
        insert equipmentList;

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

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

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

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

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

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

```
    }
}
```

Challenge - 2

WarehouseCalloutService.apxc:

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

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

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

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

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

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

```
        System.debug(response.getBody());

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

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

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

```
    }

}
```

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":fals
e,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id
":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"C
ooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d
66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

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);
    }
}
```

Challenge - 3

WarehouseSyncSchedule.apxc:

```
global with sharing class WarehouseSyncSchedule implements
Schedulable{
    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();
    }
}
```

Challenge-4

MaintenanceRequestHelperTest.apxc:

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
```

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

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

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

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

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
```

```
equipmentId,id requestId){
      Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                    Maintenance_Request__c =
requestId);
      return wp;
   }


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

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

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

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

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

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

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

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

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

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

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

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

```
        insert workP;

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

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

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

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

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

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

```
        insert vehicleList;
        insert equipmentList;

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

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

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

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

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

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

MaintenanceRequestHelper.apxc:

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


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



                }
            }
        }

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

```
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN
:ValidIds GROUP BY Maintenance_Request__c];

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

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

            );

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

            newCases.add(nc);
         }

         insert newCases;

         List<Equipment_Maintenance_Item__c> clonedWPs = new
```

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

            }
        }
        insert ClonedWPs;
    }
  }
}
```

Challenge-5

WarehouseCalloutService.apxc:

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

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

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

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

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

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