BAKI LEELA KOWSHIK REDDY
https://trailblazer.me/id/kreddy485

SALESFORCE DEVELOPER CATALYST

Apex Triggers:

AccountAddressTrigger.apxt :

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

ClosedOpportunityTrigger.apxt:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
List<Task> taskList = new List<Task>();
//first way
for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE
StageName='Closed
Won' AND Id IN : Trigger.New]){
taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
}
//second way and we should use this
/*
for(opportunity opp: Trigger.New){
if(opp.StageName!=trigger.oldMap.get(opp.id).stageName)
{
taskList.add(new Task(Subject = 'Follow Up Test Task',
WhatId = opp.Id));
}
}
*/
if(taskList.size()>0){
insert tasklist;
}
}
```

Apex Testing:

Get started with apex unit tests:

1.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;
}
}
```

2.TestVerifyDate.apxc:

```
@isTest
private class TestVerifyDate {
static testMethod void TestVerifyDate() {
VerifyDate.CheckDates(System.today(),System.today().addDays(10));
VerifyDate.CheckDates(System.today(),System.today().addDays(78));
}
}
```

Test Apex Triggers:

1.ResttrictContactByName.apxc:

```
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');
}
}
}
```

2.TestRestrictContactByName.apxc:

```
@isTest
public class TestRestrictContactByName {
static testMethod void Test()
{
List<Contact> listContact= new List<Contact>();
Contact c1 = new Contact(FirstName='Raam', LastName='Leela' ,
email='ramleela@test.com');
Contact c2 = new Contact(FirstName='gatsby', LastName =
'INVALIDNAME',email='gatsby@test.com');
listContact.add(c1);
listContact.add(c2);
Test.startTest();
try
{
insert listContact;
}
catch(Exception ee)
{
}
Test.stopTest();
}
}
```

Create Test Data for Apex Tests:

1.RandomContactFactory.apxc

```
public class RandomContactFactory {
public static List<Contact> generateRandomContacts(Integer NumberofContacts, String
lName){
```

```
List<Contact> con = new List<Contact>();
for(Integer i=0; i<NumberofContacts; i++){
lName = 'Test'+i;
Contact c = new Contact(FirstName=lName, LastName=lName);
con.add(c);
}
return con;
}
}
```

Asynchronous Apex:

Use Future methods:

1.AccountProcessor.apxc

```
public class AccountProcessor {
@future
public static void 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;
}
}
```

2.AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
public static testMethod void testAccountProcessorTest(){
Test.startTest();
Account a = new Account();
a.Name = 'The Pirates';
insert a;
Contact cont = new Contact();
cont.FirstName ='jack';
cont.LastName ='Sparrow';
cont.AccountId = a.Id;
insert cont;
```

```
Set<Id> setAccId = new Set<ID>();
setAccId.add(a.Id);
AccountProcessor.countContact(setAccId);
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);
Test.stopTest();
}
}
```

Use Batch Apex:

1.LeadProcessor.apxc

```
global class LeadProcessor implements Database.Batchable<sObject>,
Database.Stateful {
global Integer leadsProcessed = 0;
global Database.QueryLocator start(Database.BatchableContext bc){
return Database.getQueryLocator('select id, lastname ,status, company from Lead');
}
global void execute(Database.BatchableContext bc, List<Lead> allLeads){
List<Lead> leads = new List<Lead>();
for(Lead l: allLeads){
l.LeadSource='Dreamforce';
}
update leads;
}
global void finish(Database.BatchableContext bc){
System.debug(leadsProcessed + ' leads processed. Nigga!');
AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
JobItemsProcessed,
TotalJobItems, CreatedBy.Email
FROM AsyncApexJob
WHERE Id = :bc.getJobId()];
EmailManager.sendMail('jgatsby1996@gmail.com','Total Leads Porcessed are ','
'+leadsProcessed);
}
}
```

2.LeadProcessorTest

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

```
@testSetup
static void setup(){
List<Lead> leads = new List<Lead>();
for (Integer i=0;i<200;i++) {
leads.add(new Lead(Lastname='Last '+i,
status='Open - Not Contacted'
, company='LeadCompany'+i));
}
insert leads;
}
static testmethod void test() {
Test.startTest();
LeadProcessor lp = new LeadProcessor();
Id batchId = Database.executeBatch(lp);
Test.stopTest();
// after the testing stops, assert records were updated properly
System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
}
}
```

Control Processes with Queuable Apex:

1.AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {
public contact c;
public String state;
public AddPrimaryContact(Contact c, String state) {
this.c = c;
this.state = state;
}
public void execute(QueueableContext qc) {
system.debug('this.c = '+this.c+' this.state = '+this.state);
List<Account> acc_lst = new List<account>([select id, name, BillingState from account
where account.BillingState = :this.state limit 200]);
List<contact> c_lst = new List<contact>();
for(account a: acc_lst) {
contact c = new contact();
c = this.c.clone(false, false, false, false);
c.AccountId = a.Id;
```

```
c_lst.add(c);
}
insert c_lst;
}
}
```
2.AddPrimaryContactTest.apxc
```
@isTest
public class AddPrimaryContactTest {
@testSetup
public static void setup(){
List<account> acc_lst = new List<account>();
for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(i),billingstate='NY');
system.debug('account a = '+a);
acc_lst.add(a);
}
for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(50+i),billingstate='CA');
system.debug('account a = '+a);
acc_lst.add(a);
}
insert acc_lst;
}
public static testMethod void TestQueueable(){
List<Account> ac_ca=[select id from Account where billingstate='CA'];
contact c = new contact(lastname='bhau');
AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
Test.startTest();
System.enqueueJob(apc);
Test.stopTest();
system.assertEquals(50, [select count() from contact where AccountId IN :ac_ca]);
}
}
```
Schedule jobs using the apex scheduler:

1.DailyLeadProcessor.apxc
```
public class DailyLeadProcessor implements schedulable{
public void execute(schedulableContext sc) {
```

```
List<lead> l_lst_new = new List<lead>();
List<lead> l_lst = new List<lead>([select id, leadsource from lead where leadsource =
null]);
for(lead l : l_lst) {
l.leadsource = 'Dreamforce';
l_lst_new.add(l);
}
update l_lst_new;
}
}
```

2.DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
@testSetup
static void setup(){
List<Lead> lstOfLead = new List<Lead>();
for(Integer i = 1; i <= 200; i++){
Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working -
Contacted');
lstOfLead.add(ld);
}
Insert lstOfLead;
}
static testmethod void testDailyLeadProcessorScheduledJob(){
String sch = '0 5 12 * * ?';
Test.startTest();
String jobId = System.schedule('ScheduledApexTest', sch, new DailyLeadProcessor());
List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
System.assertEquals(200, lstOfLead.size());
Test.stopTest();
}
}
```

Apex Integration Services

Apex Rest Callouts:

1.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 ;
}
}
```

2.AnimalLocatorTest

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

3.AnimalLocatorMock

```
@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;
}
}
```

Apex SOAP Callouts:

1.ParkLocator.apxc

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

2.ParkLocatorTest.apxc

```
@isTest
public class ParkLocatorTest {
@isTest static void testMock(){
test.setMock(WebserviceMock.class, new ParkServiceMock());
String[] parksName = ParkLocator.Country('India');
List<String> country = new List<String>();
country.add('Inamdar National Park');
country.add('Riza National Park');
country.add('Shilpa National Park');
System.assertEquals(country, parksName, 'park names are not as expected');
}
}
```

3.ParkServiceMock

```
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> country = new List<String>();
country.add('Inamdar Shola National Park');
country.add('Riza National Park');
country.add('Shilpa National Park');
response_x.return_x = country;
response.put('response_x', response_x);
}
}
```

Apex Web Services:

1.AccountManager.apxc

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

2.AccountMAnagerTest

```
@isTest
private class AccountManagerTest {
private static testMethod void getAccountTest1() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId
+'/contacts' ;
request.httpMethod = 'GET';
RestContext.request = request;
// Call the method to test
Account thisAccount = AccountManager.getAccount();
```

```
// Verify results
System.assert(thisAccount != null);
System.assertEquals('Test record', thisAccount.Name);
}
// Helper method
static Id createTestRecord() {
// Create test record
Account TestAcc = new Account(
Name='Test record');
insert TestAcc;
Contact TestCon= new Contact(
LastName='Test',
AccountId = TestAcc.id);
return TestAcc.Id;
}
}
```

Apex Specialist Badge

Challenge 1:

Automated Record Creation

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));
} else {
nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
}
newCases.add(nc);
}
insert newCases;
List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c wpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);
```

```
}
}
insert ClonedWPs;
}
}
}
```

2.MaitenanceRequest.apxt

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

Challenge 2

Synchronize Salesforce data with an external system:

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.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();
}
}
```

Challenge 3

Schedule synchronization using Apex code:

WarehouseSyncShedule.apxc :-

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

Challenge 4

Test automation logic:

MaintenanceRequestHelperTest.apxc :-

```
@istest
```

```
public with sharing class MaintenanceRequestHelperTest {
private static final string STATUS_NEW = 'New';
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';
private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';
PRIVATE STATIC Vehicle__c createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
return Vehicle;
}
PRIVATE STATIC Product2 createEq(){
product2 equipment = new product2(name = 'SuperEquipment',
lifespan_months__C = 10,
maintenance_cycle__C = 10,
replacement_part__c = true);
return equipment;
}
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
case cs = new case(Type=REPAIR,
Status=STATUS_NEW,
Origin=REQUEST_ORIGIN,
Subject=REQUEST_SUBJECT,
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cs;
}
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return wp;
}
@istest
private static void testMaintenanceRequestPositive(){
```

```
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
Product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;
case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
insert somethingToUpdate;
Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
insert workP;
test.startTest();
somethingToUpdate.status = CLOSED;
update somethingToUpdate;
test.stopTest();
Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
from case
where status =:STATUS_NEW];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newReq.Id];
system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}
@istest
private static void testMaintenanceRequestNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;
```

```
case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
insert emptyReq;
Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
insert workP;
test.startTest();
emptyReq.Status = WORKING;
update emptyReq;
test.stopTest();
list<case> allRequest = [select id
from case];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :emptyReq.Id];
system.assert(workPart != null);
system.assert(allRequest.size() == 1);
}
@istest
private static void testMaintenanceRequestBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>();
list<Product2> equipmentList = new list<Product2>();
list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
list<case> requestList = new list<case>();
list<id> oldRequestIds = new list<id>();
for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle());
equipmentList.add(createEq());
}
insert vehicleList;
insert equipmentList;
for(integer i = 0; i < 300; i++){
requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
}
insert requestList;
for(integer i = 0; i < 300; i++){
workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
```

```
}
insert workPartList;
test.startTest();
for(case req : requestList){
req.Status = CLOSED;
oldRequestIds.add(req.Id);
}
update requestList;
test.stopTest();
list<case> allRequests = [select id
from case
where status =: STATUS_NEW];
list<Equipment_Maintenance_Item__c> workParts = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in: oldRequestIds];
system.assert(allRequests.size() == 300);
}
}
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
Set<Id> validIds = new Set<Id>();
For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
validIds.add(c.Id);
}
}
}
if (!validIds.isEmpty()){
List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
```

```
Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];
for (AggregateResult ar : results){
maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
}
for(Case cc : closedCasesM.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = 'New',
Subject = 'Routine Maintenance',
Type = 'Routine Maintenance',
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = 'Web',
Date_Reported__c = Date.Today()
);
If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
}
newCases.add(nc);
}
insert newCases;
List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c wpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);
}
}
insert ClonedWPs;
}
```

```
}
}
```
MaintenanceRequest.apxt :-
```
trigger MaintenanceRequest on Case (before update, after update) {
if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
}
}
```
Challenge 5
Test callout logic:
WarehouseCalloutService.apxc :-
```
public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.herokuapp.com/equipment';
//@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());
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 = (Decimal) mapJson.get('lifespan');
myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
warehouseEq.add(myEq);
```

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

WarehouseCalloutServiceTest.apxc :-

```
@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
// implement mock callout test here
Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
System.assertEquals('GET', request.getMethod());
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
,"na
me":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
```

```
response.setStatusCode(200);
return response;
}
}
```

Challenge 6

Test scheduling logic:

WarehouseSyncSchedule.apxc :-

```
global class WarehouseSyncSchedule implements Schedulable {
global void execute(SchedulableContext ctx) {
WarehouseCalloutService.runWarehouseEquipmentSync();
}
}
```

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
String scheduleTime = '00 00 01 * * ?';
Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime,
new
WarehouseSyncSchedule());
Test.stopTest();
//Contains schedule information for a scheduled job. CronTrigger is similar to a cron job
on
UNIX systems.
// This object is available in API version 17.0 and later.
CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
System.assertEquals(jobID, a.Id,'Schedule ');
}
}
```