APEX TRIGGER:

Challenge 1:

Create an Apex trigger for Account that matches Shipping Address Postal Code with Billing Address Postal Code based on a custom field.

```
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;
}
}
}
```

Challenge 2:

```
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:

Challange1:

```
@isTest
private class TestVerifyDate {
```

```
//testing that if date2 is within 30 days of date1, should return date 2
@isTest static void testDate2within30daysofDate1() {
Date date1 = date.newInstance(2018, 03, 20);
Date date2 = date.newInstance(2018, 04, 11);
Date resultDate = VerifyDate.CheckDates(date1,date2);
Date testDate = Date.newInstance(2018, 04, 11);
System.assertEquals(testDate,resultDate);
}
//testing that date2 is before date1. Should return "false"
@isTest static void testDate2beforeDate1() {
Date date1 = date.newInstance(2018, 03, 20);
Date date2 = date.newInstance(2018, 02, 11);
Date resultDate = VerifyDate.CheckDates(date1,date2);
Date testDate = Date.newInstance(2018, 02, 11);
System.assertNotEquals(testDate, resultDate);
}
//Test date2 is outside 30 days of date1. Should return end of month.
@isTest static void testDate2outside30daysofDate1() {
Date date1 = date.newInstance(2018, 03, 20);
Date date2 = date.newInstance(2018, 04, 25);
Date resultDate = VerifyDate.CheckDates(date1,date2);
Date testDate = Date.newInstance(2018, 03, 31);
System.assertEquals(testDate,resultDate);
}
}
```

Challenge 2:

Create an Apex trigger for Opportunity that adds a task to any opportunity set to 'Closed Won'.
To complete this challenge, you need to add a trigger for Opportunity. The trigger will add a task to any opportunity inserted or updated with the stage of 'Closed Won'. The task's subject must be 'Follow Up Test Task'.
The Apex trigger must be called 'ClosedOpportunityTrigger'
With 'ClosedOpportunityTrigger' active, if an opportunity is inserted or updated with a stage of 'Closed Won', it will have a task created with the subject 'Follow Up Test Task'.
To associate the task with the opportunity, fill the 'WhatId' field with the opportunity ID.
This challenge specifically tests 200 records in one operation.
Solution:

```
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;
}
}
```

Challenge 3:

```
public class RandomContactFactory {
public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
List<Contact> contactList = new List<Contact>();
for(Integer i=0;i<numContactsToGenerate;i++) {
Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
contactList.add(c);
System.debug(c);
}
System.debug(contactList.size());
return contactList;
}
}
```

Asynchronous Apex:

AccountProcessor.cls:

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountId_lst) {
      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from
contacts) from account]);
      for(account a:account_lst_all) {
         account_cno.put(a.id,a.contacts.size()); //populate the map
      }
      List<account> account_lst = new List<account>(); // list of account that we will
upsert
      for(Id accountId : accountId_lst) {
         if(account_cno.containsKey(accountId)) {
```

```
            account acc = new account();
            acc.Id = accountId;
            acc.Number_of_Contacts__c = account_cno.get(accountId);
            account_lst.add(acc);
          }
        }
        upsert account_lst;
    }
}
```

AccountProcessorTest.cls:

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;
        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;
        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from
account where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }
}
```

USing Batch Apex:

Apex Class:

```
global class DailyLeadProcessor implements Schedulable{
    global void 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;
      }
    }
}
```

Apex Test Class:

```
@isTest
private class DailyLeadProcessorTest{
   public static 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();
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
      Test.stopTest();
   }
}
```

Control Processes With Queueable 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 Job Using the Apex Scheduler:

Apex Class:

```
global class DailyLeadProcessor implements Schedulable{
    global void 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;
    }
  }
}
```

Apex Test Class

```
@isTest
private class DailyLeadProcessorTest{
   //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
   public static 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();
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
      Test.stopTest();
   }
}
Apex Integration Service:
AnimalLocator.cls:
public class AnimalLocator {
public class cls_animal {
public Integer id;
public String name;
public String eats;
public String says;
}
public class JSONOutput{
public cls_animal animal;
//public JSONOutput parse(String json){
//return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
//}
}
   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.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }
}
```

AnimalLocatorTest.cls:

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}
```

AnimalLocatorMock:

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        return response;
```

```
    }
}
```

Apex Soap Callouts:

Apex Class

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

Apex Test Class

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');
        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

Apex Mock Test Class

```
@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);
    }
}
```

Apex Web Services:

Apex Class

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
    }
}
```

Apex Test Class

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account  acc = AccountManager.getAccount();
        // Verify results
        System.assert(acc != null);
    }
    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;
        return acc.Id;
    }
}
```

```Apex Specialist

1-Automate record creation

```
public with sharing class MaintenanceRequestHelper {

    public static void updateWorkOrders() {

        List<case> newCaseList = new List<case>();

        Integer avgAmount=10000;

        List<Equipment_Maintenance_Item_c> newEMI = new List<Equipment_Maintenance_Item_c>();

        List<case> caseList = [SELECT id,Vehicle_c,Subject,ProductID,Productc, (SELECT id from
Equipment_Maintenance_Items_r) from case where status='closed' and Type IN ('Repair', 'Routine
Maintenance') and ID IN :Trigger.new LIMIT 200];

        Map<id,Equipment_Maintenance_Item_c> equip = new
map<id,Equipment_Maintenance_Itemc>([Select ID, Equipmentc,
Quantity_c,Equipment__r.id,Equipment_r.Maintenance_Cyclec from Equipment_Maintenance_Item_c]);

        for(case c: caseList){

            case newCase = new Case();

            newCase.Type = 'Routine Maintenance';

            newCase.Status = 'New';

            newCase.Vehicle_c = c.Vehicle_c;

            newCase.Subject = String.isBlank(c.Subject) ? 'Routine Maintenance Request' : c.Subject;

            newCase.Date_Reported__c = Date.today();

            newCase.ProductId = c.ProductId;

            newCase.Product_c = c.Product_c;

            newCase.parentID = c.Id;

            for(Equipment_Maintenance_Item_c emi : c.Equipment_Maintenance_Items_r){

                avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment_r.Maintenance_Cycle_c));

                newEMI.add(new Equipment_Maintenance_Item__c(

                    Equipment_c = equip.get(emi.id).Equipment_c,

                    Maintenance_Request__c = c.id,
```
```

```
                Quantity_c = equip.get(emi.id).Quantity_c));
            }
            Date dueDate = date.TODAY().adddays(avgAmount);
            newCase.Date_Due__c =dueDate;
            newCaseList.add(newCase);
        }
        if(newCaseList.size()>0){
            Database.insert(newCaseList);
        }
        for(Case c2: newCaseList){
            for(Equipment_Maintenance_Item__c emi2 : newEmi){
                if(c2.parentID == emi2.Maintenance_Request__c){
                    emi2.Maintenance_Request__c = c2.id;
                }
            }
        }
        if(newEmi.size()>0){
            Database.insert(newEmi);
        }
    }
}
```

MaintenanceRequest.apxt

```
    trigger MaintenanceRequest on Case (before update, after update) {
    //ToDo: Call MaintenanceRequestHelper.updateWorkOrders
    if(trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders();
    }
}
```

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-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();
   }

}
```

System.enqueueJob(new WarehouseCalloutService())

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());
    }
}
```

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_Itemc(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_Reportedc, Vehiclec, 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, Equipmentc,
Equipmentr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc 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(Equipmentr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc WHERE
Maintenance_Requestc 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-superbadge-
apex.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,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

run all

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 ');



    }
}
```