SELF LEARNING AND SUPER BADGE OF APEX

INDEX

SELF LEARNING : APEX

SUPER BADGE : APEX SPECIALIST

APEX TRIGGERS

GET STARTED WITH TRIGGERS

FILENAME : AccountAddressTrigger

CODE :

```
trigger AccountAddressTrigger on Account (before insert,before update) {
for(Account account:Trigger.New)
{
    if(account.Match_Billing_Address__c==True){
        account.ShippingPostalCode = account.BillingPostalCode;
    }
}

}
```

BULK APEX TRIGGERS

FILENAME : ClosedOpportunityTrigger

CODE :

```
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

GET STARTED WITH APEX UNIT TESTS

FILENAME : VerifyDate

CODE :

```
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;
        }

}
```

UNIT TEST OF VerifyDate

FILENAME : TestVerifyDate

CODE :

```
@isTest
private class TestVerifyDate{
    @isTest static void checkdates1(){
     Date d =
VerifyDate.CheckDates(date.parse('01/01/2021'),date.parse('01/05/2021'));
     System.assertEquals(date.parse('01/05/2021'),d);
    }
    @isTest static void checkdates2(){
        Date d = verifyDate.CheckDates(date.parse('01/01/2022'),
date.parse('05/05/2022'));
        System.assertEquals(date.parse('01/31/2022'),d);
    }
}
```

TEST APEX TRIGGERS

FILENAME : RestrictContactByName

CODE :

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

UNIT TEST OF RestrictContactByName

FILENAME : TestRestrictContactByName

CODE :

```
@isTest
public class TestRestrictContactByName {
   @isTest static void test1(){
      Contact ct = new Contact();
      ct.LastName = 'INVALIDNAME';
      Test.startTest();
      Database.SaveResult results = Database.insert(ct,false);
      Test.stopTest();
      System.assert(!results.isSuccess());
      System.assert(results.getErrors().size() > 0);
      System.assertEquals('The Last Name "INVALIDNAME" IS NOT ALLOWED',
results.getErrors()[0].getMessage());
   }  }
```

CREATE TEST DATA FOR APEX TESTS

FILENAME : RandomContactFactory

CODE :

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(integer
no_contact,string lastname){
        List<Contact>contacts = new List<Contact>();
        for(integer i=0;i<no_contact;i++){
 Contact ct = new Contact(FirstName = 'Test'+i,Lastname=lastname);
            contacts.add(ct);
        }
        return contacts;
    }
}
```

ASYNCHRONOUS APEX

USE FUTURE METHODS

FILENAME : AccountProcessor

CODE :

```
public class AccountProcessor {
  @future
  public static void countContacts(List<Id> accountIds) {
    List<Account> accounts = [Select Id,(SELECT ID FROM Contacts) from Account
Where Id IN :accountIds];
    // process account records to do awesome stuff
     for(Account acc : accounts)
     {
        acc.Number_Of_Contacts__c = acc.Contacts.size();
     }
    update accounts;
  }
  }
```

UNIT TEST OF AccountProcessor

FILENAME : AccountProcessorTest

CODE :

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void APTest(){
        List<Account> accounts = new List<Account>();
        for(Integer i=0; i<300; i++){
            accounts.add(new Account(Name ='Sami Test' +i));
        }
        insert accounts;
        List<Contact> contacts = new List<Contact>();
        List<Id>accountIds = new List<Id>();
        for(Account ac:accounts){
            contacts.add(new Contact(FirstName = ac.Name,LastName =
'SamiContact',AccountId=ac.Id));
            accountIds.add(ac.Id);
        }
        insert contacts;
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        List<Account>acnts = [SELECT Id,Number_of_contacts__c From Account];
        for(Account acc:acnts){
            System.assertEquals(1, acc.Number_of_contacts__c,'Error At least 1
Account record');
        }
    }}
```

USE BATCH APEX

FILENAME : LeadProcessor

CODE :

```
global class LeadProcessor implements Database.Batchable<SObject> {

   global Database.QueryLocator start(Database.BatchableContext bc) {
      return Database.getQueryLocator(
         'SELECT ID from Lead'
      );
   }
   global void execute(Database.BatchableContext bc, List<Lead> scope){
      // process each batch of records
      List<Lead> leads = new List<Lead>();
      for (Lead lead : scope) {
         lead.LeadSource = 'Dreamforce';
         leads.add(lead);
      }
      update leads;
   }

global void finish(Database.BatchableContext bc){

   }
}
```

UNIT TEST OF LeadProcessor

FILENAME : LeadProcessorTest

CODE :

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();

        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,Company='Test Co'));
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();

        System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce']);
    }
}
```

CONTROL PROCESSES WITH QUEUEABLE APEX

FILENAME : AddPrimaryContact

CODE :

```
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;
      }
   }
}
```

UNIT TEST OF AccountPrimaryContact

FILENAME : AddPrimaryContactTest

CODE :

```
@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 i=0;i<50;i++)
      {
         testAccounts.add(new Account(Name='Account '+i,
                                      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')]);
}
}
```

SCHEDULE JOBS USING APEX SCHEDULER

FILENAME : DailyLeadProcessor

CODE :

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext ctx){
      List<lead> leadstoupdate = new List<lead>();
      List <Lead> leads = [Select Id
                           From Lead
                           Where LeadSource = NULL Limit 200];
      for(Lead l:leads){
         l.LeadSource = 'Dreamforce';
         leadstoupdate.add(l);
      }
      update leadstoupdate;
   }
}
```

UNIT TEST OF DailyLeadProcessor

FILENAME : DailyLeadProcessorTest

CODE :

```
@isTest
private class DailyLeadProcessorTest {
   public static String CRON_EXP = '0 0 0 15 3 ? 2023';
   static testmethod void testScheduledJob() {
      List<Lead> leads = new List<Lead>();
      for (Integer i=0; i<200; i++) {
         Lead l = new Lead(
            FirstName = 'First ' + i,
            LastName = 'LastName',
            Company = 'The Inc' );
         leads.add(l);
      }

      insert leads;

      Test.startTest();
      String jobId = System.schedule('ScheduledApexTest',CRON_EXP,new
DailyLeadProcessor());

      Test.stopTest();

      List<Lead> checkleads = new List<Lead>();
      checkleads = [SELECT Id
                FROM Lead
                WHERE LeadSource='Dreamforce'and Company='The Inc'];
      System.assertEquals(200,checkleads.size(),'Lead were not created');
   }
}
```

APEX INTEGRATION SERVICES

APEX REST CALLOUTS

FILENAME : AnimalLocator

CODE :

```
public class AnimalLocator {
   public static string getAnimalNameById(Integer i){
        Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+i);
      request.setMethod('GET');
      HttpResponse response = http.send(request);

      Map<String, Object> result = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
      Map<String, Object> animal = (Map<String, Object>)result.get('animal');
      system.debug('name: '+string.valueOf(animal.get('name')));
        return string.valueOf(animal.get('name'));

        }
}
```

UNIT TEST OF AnimalLocator

FILENAME : AnimalLocatorTest

CODE :

```
@isTest
private class AnimalLocatorTest {
   @isTest static void animallocatortest1(){
      Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      String actual = AnimalLocator.getAnimalNameById(5);
      String expected = 'moose';
      System.assertEquals(actual,expected);
   } }
```

MOCK TEST OF AnimalLocator

FILENAME : AnimalLocatorMock

CODE :

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HttpResponse respond(HttpRequest request){
      HttpResponse response = new httpResponse();
      response.SetHeader('contentType','application/jason');

response.SetBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bel
lows"}}');
      response.SetStatusCode(200);
      return response;
   }
}
```

APEX SOAP CALLOUTS

FILENAME : ParkLocator

CODE :

```
public class ParkLocator {
    public static List<String> country(String country){
        ParkService.ParksImplPort parkServices = new ParkService.ParksImplPort();
        return parkServices.byCountry(country);
    }
}
```

UNIT TEST OF ParkLocator

FILENAME : ParkLocatorTest

CODE :

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class,new ParkServiceMock());
        String country = 'India';
        List<String> expectedParks = new List<String>{'Yosemite','Sequoia','Crater
Lake'};
        System.assertEquals(expectedParks,ParkLocator.country(country));
    }
}
```

MOCK TEST OF ParkLocator

FILENAME : ParkServiceMock

CODE :

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
       // start - specify the response you want to send
       ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();

               response_x.return_x = new List<String>{'Yosemite','Sequoia','Crater
Lake'};
       // end
       response.put('response_x', response_x);
   }
    }
```

APEX WEB SERVICES

FILENAME : AccountManager

CODE :

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

}
```

UNIT TEST OF AccountManager

FILENAME : AccountManagerTest

CODE :

```
@isTest
private class AccountManagerTest {

    @isTest static void testGetAccount(){
        Account acc = new Account(Name = 'TestAccount');
        insert acc;
        Contact cont = new Contact(AccountId=acc.Id,
FirstName='Sami',LastName='Pasha');
        insert cont;
        RestRequest rest_request = new RestRequest();
        rest_request.requestURI =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+acc.id+'/
contacts';
            rest_request.httpMethod = 'GET';
        RestContext.request = rest_request;
        Account myaccount = AccountManager.getAccount();
        System.assert(myaccount!=null);
        System.assertEquals('TestAccount',myaccount.Name);

    }
}
```

SUPER BADGE : APEX SPECIALIST

CHALLENGE 2 : Automate record creation

FILENAME : MaintenanceRequestHelper

CODE :

```
//APEX CLASS FILE
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;
    }
  }
}
```

Maintenance Request

FILENAME : MaintenanceRequest

CODE :

```
//Apex Trigger File
trigger MaintenanceRequest on Case (before update, after update) {

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

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

    }

}
```

CHALLENGE 3 : Synchronize Salesforce data with an external system

FILENAME : WarehouseCalloutService

CODE :

```
//Apex Class File
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);
        }
    }
  }
public static void execute(QueueableContext context){
runWarehouseEquipmentSync();
} }
```

CHALLENGE 4 : Schedule synchronization

FILENAME : WarehouseSyncSchedule

CODE :

```
//Apex class file

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

CHALLENGE 5 : Test automation logic

FILENAME : MaintenanceRequestHelperTest

CODE :

```
//Apex class file
@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);
  }
}
```

MAINTENANCE REQUEST HELPER

FILENAME : MaintenanceRequestHelper

CODE :

```
//Apex class file

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;
    }
  }
}
```

MAINTENANCE REQUEST

FILENAME : MaintenanceRequest

CODE :

```
//Apex trigger file
trigger MaintenanceRequest on Case (before update, after update) {

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

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

    }

}
```

CHALLENGE 6 : Test Callout Logic

FILENAME : WarehouseCalloutService

CODE :

```
//Apex class file
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);
        }

    }
  }
}
```

UNIT TEST OF WarehouseCalloutService

FILENAME : WarehouseCalloutServiceTest

CODE :

```
@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]);
    }
}
```

MOCK TEST OF WarehouseCalloutService

FILENAME : WarehouseCalloutServiceMock

CODE :

```
@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","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"1000
03"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

CHALLENGE 7 : Test Scheduling Logic

FILENAME : WarehouseSyncSchedule

CODE :

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

UNIT TEST OF WarehouseSyncSchedule

FILENAME : WarehouseSyncScheduleTest

CODE :

```
//Apex class file
@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 ');


    }
}
```