SPSGP-23176-Salesforce Developer Catalyst Self-Learning & Super Badges

Apex Triggers

https://trailhead.salesforce.com/en/content/learn/modules/apex_triggers/apex_triggers_intro

AccountAddressTrigger:

```
trigger AccountAddressTrigger on Account (before insert, before
update) {
    for(Account a : Trigger.new){
        If (a.Match_Billing_Address__c == true) {
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_bulk

ClosedOpportunityTrigger:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after
update) {
 List<Task> taskList = new List<Task>();
    for(Opportunity opp : Trigger.new) {
                          if(Trigger.isInsert) {
                                       if(Opp.StageName == 'Closed Won') {
```

```
                                                taskList.add(new Task(Subject =
'Follow Up Test Task', WhatId = opp.Id));
                                    }
                        }
if(Trigger.isUpdate) {
                                    if(Opp.StageName == 'Closed Won'
                                    && 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 Units Tests

https://trailhead.salesforce.com/content/learn/modules/apex_testing

VerifyDate:

```
public class VerifyDate {
	 public static Date CheckDates(Date date1, Date date2) {
		 if(DateWithin30Days(date1,date2)) {
			 return date2;
		} else {
			 return SetEndOfMonthDate(date1);
		}
	}

	private static Boolean DateWithin30Days(Date date1, Date
date2) {
	if( date2 < date1) { return false; }

	Date date30Days = date1.addDays(30);
		if( date2 >= date30Days ) { return false; }
		else { return true; }
	}
```

```
            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:

```
@isTest
private class TestVerifyDate {

   @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);
   }

   @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);
    }

    @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);
    }
}
```

Test Apex Triggers

RestrictContactByName:

```
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');
                        }


            }
}
```

TestRestrictContactByName:

```
@isTest
private class TestRestrictContactByName {

   static testMethod void  metodoTest()
   {

       List<Contact> listContact= new List<Contact>();
       Contact c1 = new Contact(FirstName='Francesco',
LastName='Riggio' , email='Test@test.com');
       Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
       listContact.add(c1);
       listContact.add(c2);

       Test.startTest();
           try
           {
               insert listContact;
           }
           catch(Exception ee)
           {
           }

       Test.stopTest();

   }

}
```

Create Test Data for Apex Tests

RandomContactFactory:

```
public with sharing class RandomContactFactory
{
        public static List<Contact> generateRandomContacts( Integer
noOfContacts, String lastName )
        {
                List<Contact> contacts = new List<Contact>();

                for( Integer i = 0; i < noOfContacts; i++ )
                {
                        Contact con = new Contact( FirstName =
'Test '+i, LastName = lastName );
                        contacts.add( con );
                }

                return contacts;
        }
}
```

Apex Integration Services

Integrate with external apps using Apex REST and SOAP services.

https://trailhead.salesforce.com/content/learn/modules/apex_integration_services

Apex REST Callouts

```
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 = '';
        if (response.getStatusCode() == 200) {
           Map < String, Object > results = (Map < String, Object >)
JSON.deserializeUntyped(response.getBody());
           Map < string, Object > animals = (Map < String, Object >)
results.get('animal');
           strResp = string.valueof(animals.get('name'));
        }
        return strResp ;
    }
}



@isTest
private class AnimalLocatorTest {
    static testMethod void testPostCallout() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String strResp = AnimalLocator.getAnimalNameById(2);
    }
}



@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":2, "name":"Test"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

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



@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'India';
   System.assertEquals(new List<String>{'Lal Bhag', 'Cubbon Park',
'Pazhassi Dam'},     ParkLocator.country(country));
    }
}



@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();
      response_x.return_x = new List<String>{'Lal Bhag', 'Cubbon
Park', 'Pazhassi Dam'};
      response.put('response_x', response_x);
  }
}
```

Apex Web Services

```
@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');

        system.debug(accountId);

        Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM
Contacts) FROM Account WHERE Id = :accountId LIMIT 1];

        return objAccount;

    }

}



@isTest

private class AccountManagerTest{

    static testMethod void testMethod1(){

        Account objAccount = new Account(Name = 'test Account');

        insert objAccount;

        Contact objContact = new Contact(LastName = 'test Contact',

                                         AccountId = objAccount.Id);

        insert objContact;

        Id recordId = objAccount.Id;

        RestRequest request = new RestRequest();

        request.requestUri =

            'https://sandeepidentity-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'

            + recordId +'/contacts';

        request.httpMethod = 'GET';
        RestContext.request = request;

        Account thisAccount = AccountManager.getAccount();

        System.assert(thisAccount!= null);

        System.assertEquals('test Account', thisAccount.Name);

    }

}
```

Asynchronous Apex

https://trailhead.salesforce.com/en/content/learn/modules/asynchronous_apex

Use Future Methods

```
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;
  }
}



@IsTest
public class AccountProcessorTest {
    public static testmethod void TestAccountProcessorTest()
    {
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact cont = New Contact();
        cont.FirstName ='Bob';
        cont.LastName ='Masters';
        cont.AccountId = a.Id;
        Insert cont;

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

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


        Account ACC = [select Number_of_Contacts__c from Account
where id = :a.id LIMIT 1];

        System.assertEquals (
Integer.valueOf(ACC.Number_of_Contacts__c) ,1);

  }


}
```

Use Batch Apex

```
global class LeadProcessor implements
Database.Batchable<Sobject>

{

    global Database.QueryLocator start(Database.BatchableContext bc)

    {

        return Database.getQueryLocator([Select LeadSource From Lead
]);

    }


    global void execute(Database.BatchableContext bc, List<Lead>
scope)

    {

            for (Lead Leads : scope)

            {
                Leads.LeadSource = 'Dreamforce';
            }
        update scope;
    }

    global void finish(Database.BatchableContext bc){   }
}



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

        Test.startTest();

            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);

        Test.stopTest();
    }
}
```

Control Processes with Queueable Apex

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

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

    }

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

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

          AddPrimaryContact apc = new AddPrimaryContact(co, state);
          Test.startTest();
            System.enqueueJob(apc);
          Test.stopTest();
     }
 }
```

Schedule Jobs Using the Apex Scheduler

```
global class DailyLeadProcessor implements Schedulable {


    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where
LeadSource = null];
        if(!lList.isEmpty()) {
                                            for(Lead l: lList) {
                                                           l.LeadSource = 'Dreamforce';
                                                                         }
                                            update lList;
                                       }
 }
}
 @isTest
private class DailyLeadProcessorTest {
               static testMethod void testDailyLeadProcessor() {
                              String CRON_EXP = '0 0 1 * * ?';
                              List<Lead> lList = new List<Lead>();
                   for (Integer i = 0; i < 200; i++) {
                                            lList.add(new
Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open -
Not Contacted'));
                              }
```

```
                insert lList;

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

Apex Specialist Superbadge

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));
             } 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;
       }
   }
}
```

MaitenanceRequest.apxt

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

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


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

   }
}
```

Challenge 2

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements
Queueable {
   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 = (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

WarehouseSyncShedule.apxc :-

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

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).i
d, 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

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {

   private static final String WAREHOUSE_URL = 'https://th-
superbadge-apex.herokuapp.com/equipment';
   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();
      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 {
   global static HttpResponse respond(HttpRequest request){
     System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
      System.assertEquals('GET', request.getMethod());
```

```
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replac
ement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":
"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Challenge 6

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();
        CronTrigger a=[SELECT Id FROM CronTrigger where
NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
    }
}
```