MODULE: Apex Triggers

UNIT: Get Started with Apex Triggers

AccountAddressTrigger:

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


    for(Account account:Trigger.New){


        if((account.Match_Billing_Address__c == True) &&
(account.BillingPostalCode!=Null)){


            account.ShippingPostalCode = account.BillingPostalCode;


        }


    }

}
```

ClosedOpportunityTrigger:

```
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;

    }

}
```

MODULE: Apex Testing

UNIT: Get Started with Apex Unit Tests

VerifyDate class :

```
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
```

```
public class TestVerifyDate {

    @isTest static void test1(){

        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));

        System.assertEquals(Date.parse('01/03/2020'), d);

    }

    @isTest static void test2(){

        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));

        System.assertEquals(Date.parse('01/31/2020'), d);

    }

}
```

UNIT: Test Apex Triggers

RestrictContactByName :

```
trigger RestrictContactByName on Contact (before insert, before update) {

  For (Contact c : Trigger.New) {

    if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid

      c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

    }

}
```

TestRestrictContactByName :

```
@isTest

public class TestRestrictContactByName {

    @isTest public static void testContact(){

        Contact cnt = new Contact();
```

```
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();

        Database.SaveResult result = Database.insert(cnt,false);

        Test.stopTest();

        System.assert(!result.isSuccess());

        System.assert(result.getErrors().size()>0);









        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());

    }




}
```

UNIT:Create Test Data for Apex Tests

RandomContactFactory class :

```
public class RandomContactFactory {



    public static List<Contact> generateRandomContacts(Integer numct,string lastname){

        List<Contact> contacts = new List<Contact>();

        for(Integer i=0;i<numct;i++){

            Contact cnt = new Contact(FirstName = 'Test' +i,LastName = lastname);

            contacts.add(cnt);

        }

        return contacts;
```

```
    }

}
```

MODULE: Asynchronous Apex

UNIT: Use Future Methods

AccountProcessor:

```
public class AccountProcessor {

    @future

    public static void countContacts(List<Id> accountIds){

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

        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from
Account Where Id in :accountIds];
```

```
        For(Account acc:accounts){

            List<Contact> contactList = acc.Contacts;

            acc.Number_Of_Contacts__c = contactList.size();

            accountsToUpdate.add(acc);

        }

        update accountsToUpdate;

    }



}
```

AccountProcessorTest :

```
@IsTest

private class AccountProcessorTest {
```

```
    @IsTest

    private static void testCountContacts(){

        Account newAccount = new Account(Name='Test Account');

        insert newAccount;



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

        insert newContact1;



        Contact newContact2 = new Contact(FirstName='John',LastName='Doe',AccountId
= newAccount.Id);

        insert newContact2;
```

```
        List<Id> accountIds = new List<Id>();

        accountIds.add(newAccount.Id);



        Test.startTest();

        AccountProcessor.countContacts(accountIds);

        Test.stopTest();



    }

}
```

UNIT: Use Batch Apex

LeadProcessor:

```
public class LeadProcessor implements Database.Batchable<sObject> {
```

```
 public Database.QueryLocator start(Database.BatchableContext bc) {

    // collect the batches of records or objects to be passed to execute

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

}

public void execute(Database.BatchableContext bc, List<Lead> leads){

     // process each batch of records

        for (Lead Lead : leads) {

            lead.LeadSource = 'Dreamforce';

        }

    update leads;

}
```

```
    public void finish(Database.BatchableContext bc){

     }

}
```

LeadProcessorTest:

```
@isTest

public class LeadProcessorTest {



        @testSetup

    static void setup() {

        List<Lead> leads = new List<Lead>();

        for(Integer counter=0 ;counter <200;counter++){

            Lead lead = new Lead();
```

```
        lead.FirstName ='FirstName';

        lead.LastName ='LastName'+counter;

        lead.Company ='demo'+counter;

        leads.add(lead);

    }

    insert leads;

}



@isTest static void test() {

    Test.startTest();

    LeadProcessor leadProcessor = new LeadProcessor();

    Id batchId = Database.executeBatch(LeadProcessor);
```

```
        Test.stopTest();

    }

}
```

UNIT:Control Processes with Queueable Apex

AddPrimaryContact:

```
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;

        }



    }

}
```

AddPrimaryContactTest :

```
@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();

    }

}
```

UNIT: Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor:

```
public class DailyLeadProcessor implements Schedulable  {

   Public void execute(SchedulableContext SC){

      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];

       for(Lead l:LeadObj){

          l.LeadSource='Dreamforce';

          update l;
```

```
    }

  }

}
```

DailyLeadProcessorTest:

```
@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());

        }

}
```

MODULE: Apex Integration Services

UNIT:Apex REST Callouts

AnimalLocator:

```
public class AnimalLocator{

   public static String getAnimalNameById(Integer x){

      Http http = new Http();

      HttpRequest req = new HttpRequest();

      req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);

      req.setMethod('GET');

      Map<String, Object> animal= new Map<String, Object>();

      HttpResponse res = http.send(req);

         if (res.getStatusCode() == 200) {

      Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());

     animal = (Map<String, Object>) results.get('animal');

      }
```

```
return (String)animal.get('name');

    }

}
```

AnimalLocatorTest :

```
@isTest

private class AnimalLocatorTest{

    @isTest static void AnimalLocatorMock1() {

        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

        string result = AnimalLocator.getAnimalNameById(3);

        String expectedResult = 'chicken';

        System.assertEquals(result,expectedResult );
```

```
    }

}
```

AnimalLocatorMock :

```
@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

    // Implement this interface method

    global HTTPResponse respond(HTTPRequest request) {

        // Create a fake response

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type', 'application/json');

        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');

        response.setStatusCode(200);
```

```
        return response;

    }

}
```

UNIT: Apex SOAP Callouts

ParkLocator:

```
public class ParkLocator {

    public static string[] country(string theCountry) {

        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove space

        return parkSvc.byCountry(theCountry);

    }

}
```

ParkLocatorTest:

```
@isTest

private class ParkLocatorTest {

    @isTest

    static void testCallout() {

        Test.setMock(WebServiceMock.class, new ParkServiceMock ());

        String country = 'United States';

        List<String> result = ParkLocator.country(country);

        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};

         System.assertEquals(parks, result);

    }

}
```

ParkServiceMock:

```
@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>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};

      // end

      response.put('response_x', response_x);

  }

}
```

UNIT: Apex Web Services

AccountManager:

```
@RestResource(urlMapping='/Accounts/*/contacts')
```

```
global class AccountManager {

    @HttpGet

    global static Account getAccount() {

        RestRequest req = RestContext.request;

        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

                       FROM Account WHERE Id = :accId];

        return acc;

    }

}
```

AccountManagerTest :

```
@isTest

private class AccountManagerTest {



    private static testMethod void getAccountTest1() {

        Id recordId = createTestRecord();

        // Set up a test request

        RestRequest request = new RestRequest();

        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts' ;

        request.httpMethod = 'GET';

        RestContext.request = request;

        // Call the method to test

        Account thisAccount = AccountManager.getAccount();
```

```
    // Verify results

    System.assert(thisAccount != null);

    System.assertEquals('Test record', thisAccount.Name);



}



// Helper method

    static Id createTestRecord() {

    // Create test record

    Account TestAcc = new Account(

      Name='Test record');

    insert TestAcc;
```

```
        Contact TestCon= new Contact(

        LastName='Test',

        AccountId = TestAcc.id);

        return TestAcc.Id;

    }

}
```

-

SUPERBADGE: Apex Specialist

Challenge 1:Automated Record Creation

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: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();
    }

}
```

open execute anonymous window (CTRl+E) and run this method ,

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());
}
}
```

Challenge 4:Test automation logic

MaintenanceRequestHelperTest.apxc :

```
@istest
public with sharing class
MaintenanceRequestHelperTest {

    private static final string
STATUS_NEW = 'New';
    private static final string
WORKING = 'Working';
    private static final string CLOSED
= 'Closed';
    private static final string REPAIR
= 'Repair';
    private static final string
REQUEST_ORIGIN = 'Web';
    private static final string
REQUEST_TYPE = 'Routine Maintenance';
```

```
    private static final string
REQUEST_SUBJECT = 'Testing subject';


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


    PRIVATE STATIC Product2 createEq(){
        product2 equipment =
new product2(name = 'SuperEquipment',


     lifespan_months__C = 10,


     maintenance_cycle__C = 10,


     replacement_part__c = true);
        return equipment;
    }


    PRIVATE STATIC Case
createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new
case(Type=REPAIR,

            Status=STATUS_NEW,
```

```
        Origin=REQUEST_ORIGIN,

        Subject=REQUEST_SUBJECT,

        Equipment__c=equipmentId,

        Vehicle__c=vehicleId);
    return cs;
  }


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

Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,




Maintenance_Request__c = requestId);
    return wp;
  }



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

```
vehicle.Id;


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


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



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


        test.startTest();


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


        Case newReq =
[Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c


        from case
```

```
        where status =:STATUS_NEW];

Equipment_Maintenance_Item__c workPart = [select id

            from
Equipment_Maintenance_Item__c

            where Maintenance_Request__c
=:newReq.Id];

system.assert(workPart != null);

system.assert(newReq.Subject != null);

system.assertEquals(newReq.Type, REQUEST_TYPE);

SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);

SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);

SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }

    @istest
    private static void
testMaintenanceRequestNegative(){
        Vehicle__C vehicle =
createVehicle();
```

```
        insert vehicle;
        id vehicleId =
vehicle.Id;


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


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



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


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


        list<case>
allRequest = [select id

                from case];
```

```
Equipment_Maintenance_Item__c workPart = [select id

        from
Equipment_Maintenance_Item__c

        where Maintenance_Request__c =
:emptyReq.Id];



system.assert(workPart != null);

system.assert(allRequest.size() == 1);
    }

    @istest
    private static void
testMaintenanceRequestBulk(){

list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2>
equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c>
workPartList = new list<Equipment_Maintenance_Item__c>();
        list<case>
requestList = new list<case>();
        list<id>
oldRequestIds = new list<id>();

        for(integer i = 0; i
```

```
< 300; i++){
            vehicleList.add(createVehicle());

equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;


        for(integer i = 0; i
< 300; i++){

requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert requestList;


        for(integer i = 0; i
< 300; i++){

workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
        }
        insert workPartList;


        test.startTest();
        for(case req :
requestList){

req.Status = CLOSED;
```

```
oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

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

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

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

MaintenanceRequestHelper.apxc
:

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



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



                }
            }
        }


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

```
GROUP BY Maintenance_Request__c];


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


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


                );


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


                newCases.add(nc);
            }
```

```
        insert newCases;


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


            }
          }
          insert ClonedWPs;
      }
   }
}
```

MaintenanceRequest.apxt :

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

Challenge 5:Test callout logic

WarehouseCalloutService.apxc :

```
public with sharing class WarehouseCalloutService {


    private static final String WAREHOUSE_URL = 'https://th-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,"quantit
y":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
response.setStatusCode(200);
return response;
}
}
```

Challenge 6:Test scheduling logic

WarehouseSyncSchedule.apxc :

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

WarehouseSyncScheduleTest.apxc :

```
@isTest
public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
String scheduleTime = '00 00 01 * * ?';
Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
```

```
String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime,
new WarehouseSyncSchedule());
Test.stopTest();
//Contains schedule information for a scheduled job. CronTrigger is similar to a cron
job on UNIX systems.
// This object is available in API version 17.0 and later.
CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
System.assertEquals(jobID, a.Id,'Schedule ');
}
}
```