SALESFORCE DEVELOPER CATALYST

- **CREATE NEW PLAYGROUND FOR STARTINNG THE MODULES**
- **PRIMARILY DO THE PREREQUISITES AND UNDERSTAND THE USE CASE GIVEN FOR US TO SOLVE**

APEX TRIGGERS

`AccountAddressTrigger:-`

1.open Developer console click
file-new-apex class-name-AccountAddressTrigger

Code:-

```
trigger AccountAddressTrigger on Account (before insert, before update) {
//For this challenge, you need to create a trigger that, before insert or update, checks for a checkbox, and if the checkbox field is true, sets the Shipping Postal Code (whose API name is ShippingPostalCode) to be the same as the Billing Postal Code (BillingPostalCode).

//The Apex trigger must be called 'AccountAddressTrigger'.
//The Account object will need a new custom checkbox that should have the Field Label 'Match Billing Address' and Field Name of 'Match_Billing_Address'. The resulting API Name should be 'Match_Billing_Address__c'.
//With 'AccountAddressTrigger' active, if an Account has a Billing Postal
```

Code and 'Match_Billing_Address__c' is true, the record should have the Shipping Postal Code set to match on insert or update.

```
    for (account acct:trigger.new)
    {if(acct.Match_Billing_Address__c == true)
      {acct.shippingPostalCode = acct.billingPostalCode;}
    }
//my code states that if the Match Billing Checkbox is checked then before an account is inserted
/// or updated then the billing and shipping codes will be matched and added as new to the database.
/// my question is why is line 9 and action.  When I first tried to write it I wanted to write "then" as in if/then
}
```

ClosedOpportunityTrigger:-

```
trigger ClosedOpportunityTrigger on Opportunity(after insert, after update) {
    List<Task> oppList = new List<Task>();
    List<Opportunity> oppListQuery = [SELECT Id,StageName,(SELECT WhatId,Subject FROM Tasks) FROM Opportunity
                    WHERE Id IN :Trigger.New AND StageName LIKE '%Closed Won%'];

    for (Opportunity a : oppListQuery) {
        oppList.add(new Task( WhatId=a.Id, Subject='Follow Up Test Task'));

    }

    if (oppList.size() > 0) {
        insert oppList;
    }
}
```

APEX TESTING :-

Apex Class:- VerifyDate.apxc

```
public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2.
Otherwise use the end of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
    private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
    if( date2 < date1) { return false; }

    //check that date2 is within (>=) 30 days of date1
    Date date30Days = date1.addDays(30); //create a date 30 days away
from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
```

```
        }

        //method to return the end of the month of a given date
        private static Date SetEndOfMonthDate(Date date1) {
            Integer totalDays = Date.daysInMonth(date1.year(),
date1.month());
            Date lastDay = Date.newInstance(date1.year(), date1.month(),
totalDays);
            return lastDay;
        }

}
```

TestVerifyDate.apxc(Apex Class):-

```
@isTest
private class TestVerifyDate {

   //testing that if date2 is within 30 days of date1, should return date 2
   @isTest static void testDate2within30daysofDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 04, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 04, 11);
      System.assertEquals(testDate,resultDate);
   }

   //testing that date2 is before date1. Should return "false"
   @isTest static void testDate2beforeDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 02, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 02, 11);
```

```
        System.assertNotEquals(testDate, resultDate);
    }

    //Test date2 is outside 30 days of date1. Should return end of month.
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

RestrictContactByName.apxt

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

     //check contacts prior to insert or update for invalid data
     For (Contact c : Trigger.New) {
          if(c.LastName == 'INVALIDNAME') { //invalidname is
invalid
               c.AddError('The Last Name "'+c.LastName+'" is not
allowed for DML');
          }

     }



}
```

TestRestrictContactByName

```
        @isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        //try inserting a Contact with INVALIDNAME
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;

        // Perform test
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        // Verify
        // In this case the creation should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                             result.getErrors()[0].getMessage());

    }
}
```

RandomContactFatory.apxc:-

```
//@isTest
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
```

```
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }

}
```

TestDataFacatory.apxc

```
@isTest
public class TestDataFactory {
    public static List<Account> createAccountsWithOpps(Integer numAccts, Integer
numOppsPerAcct) {
        List<Account> accts = new List<Account>();
        for(Integer i=0;i<numAccts;i++) {
            Account a = new Account(Name='TestAccount' + i);
            accts.add(a);
        }
        insert accts;
        List<Opportunity> opps = new List<Opportunity>();
        for (Integer j=0;j<numAccts;j++) {
            Account acct = accts[j];
            // For each account just inserted, add opportunities
            for (Integer k=0;k<numOppsPerAcct;k++) {
                opps.add(new Opportunity(Name=acct.Name + ' Opportunity ' + k,
                                         StageName='Prospecting',
                                         CloseDate=System.today().addMonths(1),
                                         AccountId=acct.Id));
            }
        }
        // Insert all opportunities for all accounts.
        insert opps;
        return accts;
    }
}
```

ASYNCHRONOUS APEX(MODULE)

Asynchronous Processing Basics

COMPLETE THE QUIZ BY READING THE GIVEN DATA IN MODULE AND TRY UNDERSTAND IT WHICH CAN BE HELPFUL FOR NEXT MODULES

Use Future Method

AccountProcessor.apxc

```
public class AccountProcessor
{
  @future
  public static void countContacts(Set<id> setId)
  {
     List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from contacts ) from
account where id in :setId ];
     for( Account acc : lstAccount )
     {
        List<Contact> lstCont = acc.contacts ;

        acc.Number_of_Contacts__c = lstCont.size();
     }
     update lstAccount;
  }
}
```

AccountProcessorTest.apxc

```
@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

LeadProcessor.apxc

```
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){    }
}
```

LeadProcessorTest.apxc

```
@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

AddPrimaryContact.apxc

```
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.apxc

```
@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

DailyLeadProcessor.apxc

```
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;
                }
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {

    @isTest
    public static void testDailyLeadProcessor(){

        //Creating new 200 Leads and inserting them.
        List<Lead> leads = new List<Lead>();
        for (Integer x = 0; x < 200; x++) {
            leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
        }
        insert leads;

        //Starting test. Putting in the schedule and running the DailyLeadProcessor execute method.
        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());
        Test.stopTest();

        //Once the job has finished, retrieve all modified leads.
        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT
200];

        //Checking if the modified leads are the same size number that we created in the start of this
method.
        System.assertEquals(200, listResult.size());

    }
}
```

Monitor Asynchronous Apex

READ AND UNDERSTAND THE GIVEN DATA AND COMPLETE THE QUIZ

Apex Integration Services

AnimalLocator.apxc

```
public class AnimalLocator
{

  public static String getAnimalNameById(Integer id)
   {
       Http http = new Http();
       HttpRequest request = new HttpRequest();
       request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
       request.setMethod('GET');
       HttpResponse response = http.send(request);
         String strResp = '';
          system.debug('*******response '+response.getStatusCode());
          system.debug('*******response '+response.getBody());
       // If the request is successful, parse the JSON response.
       if (response.getStatusCode() == 200)
       {
           // Deserializes the JSON string into collections of primitive data types.
          Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
           // Cast the values in the 'animals' key as a list
          Map<string,object> animals = (map<string,object>) results.get('animal');
           System.debug('Received the following animals:' + animals );
           strResp = string.valueof(animals.get('name'));
           System.debug('strResp >>>>>>' + strResp );
       }
```

```
        return strResp ;
    }

}
```

AnimalLocatorTest.apxc

```
@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);
    }
}
```

AnimalLockMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
         HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

`ParkLocator`.apxc

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

`ParkLocatorTest`.apxc

```
@isTest
```

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

      System.assertEquals('Park1', arrayOfParks[0]);
   }
}
```

Apex Web Services

AccountManager.apxc

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

        return acc;
    }
}
```

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
```

```
        // Call the method to test
        Account  acc = AccountManager.getAccount();

        // Verify results
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId =
acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

Challenge 1

Automated Record Creation

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
  public static void updateWorkOrders() {
    List<case> newCaseList = new List<case>();
    Integer avgAmount=10000;

    List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
    List<case> caseList = [SELECT id,Vehicle__c,Subject,ProductID,Product__c,
(SELECT id from Equipment_Maintenance_Items__r) from case where
status='closed' and Type IN ('Repair', 'Routine Maintenance') and ID IN :Trigger.new
LIMIT 200];
    Map<id,Equipment_Maintenance_Item__c> equip = new
map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c,
Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle__c from
Equipment_Maintenance_Item__c ]);
    for(case c: caseList){
      case newCase = new Case();
      newCase.Type = 'Routine Maintenance';
      newCase.Status = 'New';
      newCase.Vehicle__c = c.Vehicle__c;
```

```
        newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request'
: c.Subject;
        newCase.Date_Reported__c = Date.today();
        newCase.ProductId = c.ProductId;
        newCase.Product__c = c.Product__c;
        newCase.parentID = c.Id;


        for(Equipment_Maintenance_Item__c emi :
c.Equipment_Maintenance_Items__r ){
          avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment__r.Maintenan
ce_Cycle__c));
          newEMI.add(new Equipment_Maintenance_Item__c(
            Equipment__c = equip.get(emi.id).Equipment__c,
            Maintenance_Request__c = c.id,
            Quantity__c = equip.get(emi.id).Quantity__c));
        }
        Date dueDate = date.TODAY().adddays(avgAmount);
        newCase.Date_Due__c =dueDate;
        newCaseList.add(newCase);

      }
      if(newCaseList.size()>0){
        Database.insert(newCaseList);
      }

      for(Case c2: newCaseList){
        for(Equipment_Maintenance_Item__c emi2 : newEmi){
          if(c2.parentID == emi2.Maintenance_Request__c){
            emi2.Maintenance_Request__c = c2.id;
          }
        }
      }

      if(newEmi.size()>0){
        Database.insert(newEmi);
```

```
    }
  }
}
```

MaitenanceRequest.apxt (CLICK NEW APPEX TRIGGER)

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

  //ToDo: Call MaintenanceRequestHelper.updateWorkOrders

  if(trigger.isAfter){

    MaintenanceRequestHelper.updateWorkOrders();

  }

}
```

Challenge 2

Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

  //class that makes a REST callout to an external warehouse system to get a list of equipment that
needs to be updated.
  //The callout's JSON response returns the equipment records that you upsert in Salesforce.

  @future(callout=true)
  public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);

    List<Product2> warehouseEq = new List<Product2>();
```

```
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan,
maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within
Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

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

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

After saving the code open execute anonymous window (CTRl+E) and run this method ,

```
System.enqueueJob(new WarehouseCalloutService());
```

Now check Challenge.

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 {
    @istest
    public static void BulkTesting(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10, Replacement_Part__c =
true);

        Database.insert(pt2);


        List<case> caseList = new List<case>();
        for(Integer i=0;i<300;i++){
            caseList.add(new case(
                Type = 'Routine Maintenance',
```

```
            Status = 'Closed',
            Subject = 'testing',
            Date_Reported__c = Date.today(),
            ProductId = pt2.id
        ));
    }
    if(caseList.size()>0){
        Database.insert(caseList);
        System.debug(pt2.id);
        System.debug(caseList.size());
    }


    List<Equipment_Maintenance_Item__c> newEMI = new List<Equipment_Maintenance_Item__c>();
    for(Integer i=0;i<5;i++){
        newEMI.add(new Equipment_Maintenance_Item__c(
            Equipment__c = pt2.id,
            Maintenance_Request__c = caseList[1].id,
            Quantity__c = 10));
    }
    if(newEmi.size()>0){
        Database.insert(newEmi);
    }

    for(case c :caseList){
        c.Subject = 'For Testing';
    }
    Database.update(caseList);
    Integer newcase = [Select count() from case where ParentId = :caseList[0].id];
    System.assertEquals(1, newcase);

}

@istest
public static void positive(){
    product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
    insert pt2;

    Case cParent = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c = Date.today(),
                            ProductId = pt2.id);
    insert cParent;
```

```
        Case cChild = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c = Date.today(),
                               ProductId = pt2.id,parentID = cParent.ParentId);
        insert cChild;

        cParent.subject = 'child refrecer record';
        update cParent;

        Integer newcase = [Select count() from case where ParentId = :cParent.id];
        System.assertEquals(1, newcase);

    }
    @istest public static void negetive(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
        insert pt2;

        Case c = new Case(Type = 'Repair',status = 'New',Date_Reported__c = Date.today(),
                          ProductId = pt2.id);
        insert c;

        c.Status = 'Working';
        update c;


        Integer newcase = [Select count() from case where ParentId = :c.id];
        System.assertEquals(0, newcase);
    }



}
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
```

```
    public static void updateWorkOrders() {
        List<case> newCaseList = new List<case>();
        Integer avgAmount=10000;

        List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
        List<case> caseList = [SELECT id,Vehicle__c,Subject,ProductID,Product__c,
(SELECT id from Equipment_Maintenance_Items__r) from case where status='closed'
and Type IN ('Repair', 'Routine Maintenance') and ID IN :Trigger.new LIMIT 200];
        Map<id,Equipment_Maintenance_Item__c> equip = new
map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c,
Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle__c from
Equipment_Maintenance_Item__c ]);
        for(case c: caseList){
            case newCase = new Case();
            newCase.Type = 'Routine Maintenance';
            newCase.Status = 'New';
            newCase.Vehicle__c = c.Vehicle__c;
            newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request' :
c.Subject;
            newCase.Date_Reported__c = Date.today();
            newCase.ProductId = c.ProductId;
            newCase.Product__c = c.Product__c;
            newCase.parentID = c.Id;



            for(Equipment_Maintenance_Item__c emi :
c.Equipment_Maintenance_Items__r ){
                avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment__r.Maintenance_Cy
cle__c));
                newEMI.add(new Equipment_Maintenance_Item__c(
                    Equipment__c = equip.get(emi.id).Equipment__c,
                    Maintenance_Request__c = c.id,
                    Quantity__c = equip.get(emi.id).Quantity__c));
            }
            Date dueDate = date.TODAY().adddays(avgAmount);
```

```
            newCase.Date_Due__c =dueDate;
            newCaseList.add(newCase);

        }
        if(newCaseList.size()>0){
            Database.insert(newCaseList);
        }

        for(Case c2: newCaseList){
            for(Equipment_Maintenance_Item__c emi2 : newEmi){
                if(c2.parentID == emi2.Maintenance_Request__c){
                    emi2.Maintenance_Request__c = c2.id;
                }
            }
        }

        if(newEmi.size()>0){
            Database.insert(newEmi);
        }
    }
}
```

MaintenanceRequest.apxt :-

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

    //ToDo: Call MaintenanceRequestHelper.updateWorkOrders

    if(trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders();

    }

}
```

Challenge 5

Test callout logic

- Go to the developer console use below code ,

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();
  }

}
```

WarehouseCalloutServiceTest.apxc :-

```
@IsTest
private class WarehouseCalloutServiceTest {
  // implement your mock callout test here
  @isTest static void mainTest(){
    Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
    Test.startTest();
    Id jobID = System.enqueueJob(new WarehouseCalloutService());
    //System.assertEquals('Queued',aaj.status);
    Test.stopTest();
```

```
        AsyncApexJob aaj = [SELECT Id, Status, NumberOfErrors FROM AsyncApexJob WHERE
Id = :jobID];
        System.assertEquals('Completed',aaj.status);
        System.assertEquals(0, aaj.NumberOfErrors);
    }
}
```

WarehouseCalloutServiceMock.apxc :-

```
@istest

global class WarehouseCalloutServiceMock implements HttpCalloutMock{

    // implement http mock callout

    global HttpResponse respond(HttpRequest request){

        HttpResponse response = new HttpResponse();

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


response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":true,"quantity":5,"name":"
Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"220000"}]');

        response.setStatusCode(200);

        return response;

    }


}
```

Challenge 6
Test scheduling logic

- Go to the developer console use below code ,

WarehouseSyncSchedule.apxc :-

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

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 ');


   }
}
```