CODES FOR HANDS-ON CHALLENGES IN SALESFORCE SELF LEARNING:

Apex triggers module:

https://trailhead.salesforce.com/content/learn/modules/apex_triggers?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

Get Started with Apex Triggers Challenge:

1)AccountAddressTrigger.apxt

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

   for(Account acct:Trigger.new)
   {
      if(acct.Match_Billing_Address__c == True)
         acct.ShippingPostalCode = Acct.BillingPostalCode;
   }
}
```

Bulk Apex Triggers Challenge:

1)ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();

   for(Opportunity opp : Trigger.New) {
      if(opp.StageName == 'Closed Won'){
         taskList.add(new Task(Subject = 'Follow Up Test Task',
                               WhatId=opp.Id));
      }

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

APEX TESTING MODULE
https://trailhead.salesforce.com/content/learn/modules/apex_testing?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

1)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;
  }

}
```

2)TestVerifyDate.apxc

```
@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);
   }

}
```

Test Apex Triggers:

1) 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');
          }
  }
}
```

2) TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {

   @isTest
   public static void testContact(){
      Contact ct=new Contact();
      ct.LastName = 'INVALIDNAME';
      Database.SaveResult res = Database.insert(ct,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for

   }

}

```

Create Test Data For Apex Tests:

1) RandomContactFactory.apxc

```
public class RandomContactFactory {

   public static List<Contact> generateRandomContacts(Integer num, String
lastName){
      List<Contact> contactList = new List<Contact>();
      for(Integer i=1;i<=num;i++){
         Contact ct = new Contact(FirstName = 'Test '+i, LastName =lastName);
         contactList.add(ct);
      }
      return contactList;
   }
}
```

2)TestDataFactory.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;
    }
}
```

<u>ASYNCHRONOUS APEX MODULE:</u>
https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

<u>Use Future Methods:</u>

1)AccountProcessor.apxc:

```
public without sharing class AccountProcessor {

   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accounts = [SELECT Id,(SELECT Id FROM Contacts) FROM Account WHERE Id IN :accountIds];

      for (Account acc: accounts){
         acc.Number_Of_Contacts__c = acc.Contacts.size();
      }

      update accounts;
   }

}
```

2)AccountProcessorTest.apxc

```
@isTest
private class AccountProcessorTest {

   @isTest
   private static void countContactsTest() {

      // Load Test Data
      List<Account> accounts = new List<Account>();
      for(Integer i=0; i<300; i++) {
            accounts.add(new Account(Name='Test Account' +i));
        }
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for(Account acc: accounts) {
            contacts.add(new Contact(FirstName=acc.Name,
LastName='TestContact', AccountId=acc.Id));
            accountIds.add(acc.Id);
        }
        insert contacts;



        // Do the test
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();

        //Check result
        List<Account> accs = [SELECT Id, Number_of_Contacts__c FROM Account];
        for(Account acc : accs) {
            System.assertEquals(1, acc.Number_of_Contacts__c, 'ERROR: At least 1

        }


    }

}
```

USE BATCH APEX:

1) LeadProcessor.apxc

```
public without sharing class LeadProcessor implements
Database.Batchable<sObject> {

  public Database.QueryLocator start(Database.BatchableContext dbc){
    return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
  }

  public void execute(Database.BatchableContext dbc, List<Lead> leads) {
    for (Lead l : leads){
      l.LeadSource = 'Dreamforce';
    }
    update leads;
  }

  public void finish (Database.BatchableContext dbc){
    System.debug('Done');
  }

}

```

2) LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

  @isTest
  private static void testBatchClass() {

    //Load test data
    List<Lead> leads = new List<Lead>();
    for (Integer i=0; i<200; i++) {
            leads.add(new Lead(LastName='Connock', Company='Salesforce'));
        }
        insert leads;

        //Perform the test
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        //Check the result
        List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE LeadSource =
'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR:At least 1 Lead

    }

}

```

Control Processes with Queueable Apex:

1) AddPrimaryContact.apxc

```
public without sharing class AddPrimaryContact implements Queueable {

    private Contact contact;
    private String state;

    public AddPrimaryContact (Contact inputContact, String inputState) {
        this.contact = inputContact;
        this.state = inputState;
    }

    public void execute(QueueableContext context){

        //Retrieve 200 Account records
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

        //Create empty list of Contact records
        List<Contact> contacts = new List<Contact>();

        //Iterate through the Account records
        for (Account acc : accounts) {

            //Clone (copy) the Contact record, make the clone a child of the specific Account record
            //and add to the list of Contacts
            Contact contactClone = contact.Clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }

        insert contacts;

    }

}

```

2) AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass() {

        //Load test data
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<500; i++) {
            Account acc = new Account(Name='Test Account');
            if ( i<250 ) {
                acc.BillingState = 'NY';
            } else {
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact = new Contact(FirstName='Simon' ,LastName='Connock');
        insert contact;

        //Perform the test
        Test.startTest();
        Id jobId = System.enqueueJob(new AddPrimaryContact(Contact, 'CA'));
        Test.stopTest();

        //Check the result
        List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState = 'CA'];
        System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of

    }

}
```

<u>Schedule Jobs Using the Apex Scheduler:</u>

1) DailyLeadProcessor.apxc

```
public without sharing class DailyLeadProcessor implements Schedulable {

   public void execute(SchedulableContext ctx){
       //System.debug('Context ' + ctx.getTriggerId()); //Returns the ID of the CronTrigger schedule

       //Get 200 Lead records and modify the LeadSource field
       List<Lead> leads =  [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT 200];
       for ( Lead l : leads) {
           l.LeadSource = 'Dreamforce';
       }

       //Update the modified records
       update leads;


   }

}
```

2)DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {

   private static String CRON_EXP = '0 0 0 ? * * *'; // Midnight every day

   @isTest
   private static void testScheduleClass(){

       //Load test data
       List<Lead> leads = new List<Lead>();
       for (Integer i=0; i<500; i++) {
           if( i<250 ) {
               leads.add(new Lead(LastName='Connock', Company='Salesforce'));
```

```
            } else {
                leads.add(new Lead(LastName='Connock', Company='Salesforce', Leadsource='Other'));
            }
        }
        insert leads;

        //Perform the test
        Test.startTest();
        String jobId = System.schedule('Process Leads', CRON_EXP, new DailyLeadProcessor());
        Test.stopTest();

        //Check the result
        List<Lead> updateLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updateLeads.size(), 'ERROR:At least 1 record not


        //Check the scheduled time
        List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger WHERE Id = :jobId];
        system.debug('Next Fire Time ' + cts[0].NextFireTime);
    }
}

```

<u>APEX INTEGRATION SERVICES MODULE:</u>

https://trailhead.salesforce.com/content/learn/modules/apex_integration_services?trail mix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

Apex REST Callouts :

1)AnimalLocator.apxc

```
public class AnimalLocator {

   public static String getAnimalNameById (Integer i) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-

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

      //If the request is successful, parse the JSON response.
         Map<String, Object> result = (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
         Map<String, Object> animal = (Map<String, Object>)result.get('animal');
         System.debug('name: '+string.valueOf(animal.get('name')));
  return string.valueOf(animal.get('name'));
   }
}
```

2)AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

   global HttpResponse respond(HttpRequest request) {
      HttpResponse response = new HttpResponse();
      response.setHeader('contentType', 'application/json');

response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"

      response.setStatusCode(200);
```

```
        return response;
    }
}

```

3) AnimalLocatorTest.apxc

```
@isTest
public class AnimalLocatorTest {

    @isTest
    static void animalLocatorTest1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }
}
```

Apex SOAP Callouts :

1)ParkService.apxc

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
```

```
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-

        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
                          response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}

```

2)ParkLocator.apxc

```
public class ParkLocator {

    public static List < String > country(String country) {
        ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
        return prkSvc.byCountry(country);
    }

}
```

3) ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {

    @isTest static void testCallout () {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        List<String> expectedParks = new List<String>{'Yosemite', 'Sequoia', 'Crater


        System.assertEquals(expectedParks,ParkLocator.country(country));
    }
}
```

4) ParkServiceMock.apxc

```
@isTest
```

```
global class ParkServiceMock implements WebServiceMock {

   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
            // start - specify the response you want to send
            parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
            response_x.return_x = new List<String>{'Yosemite', 'Sequoia', 'Crater Lake'};
            response.put('response_x', response_x);

   }
}

```

<u>Apex Web Services :</u>

1) AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {

   @HttpGet
   global static Account getAccount() {
       RestRequest request = RestContext.request;
       String accountId =
request.requestURI.substringBetween('Accounts/','/contacts');
       Account result = [SELECT ID,Name,(SELECT ID, FirstName, LastName FROM
Contacts)
```

```
                FROM Account
                WHERE Id = :accountId];
        return result;
    }
}

```

2) AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

    @isTest
    static void testGetAccount() {
        Account a = new Account(Name='TestAccount');
        insert a;
        Contact c = new Contact(AccountId=a.Id, FirstName='Test',
LastName='Test');
        insert c;

        RestRequest request = new RestRequest();
        request.requestUri =
'hhttps://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/conta

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

        Account myAcct = AccountManager.getAccount();
        //verify results
        System.assert(myAcct != null);
        System.assertEquals('TestAccount', myAcct.Name);
    }
}

```

CODES FOR APEX SUPERBADGE IN SALESFORCE DEVELOPER SPECIALIST CHALLENGE:

https://trailhead.salesforce.com/content/learn/superbadges/superbadge_apex?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

STEP 2:Automate record creation:

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;
    }
  }
}
```

2) MaitenanceRequest.apxt

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

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

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

  }
}
```

STEP 3: Synchronize Salesforce data with an external system:

1)WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-


  //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();
    }

}

```

In Anonymous window execute this method:

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

STEP 4: Schedule synchronization using Apex code:

1)WarehouseSyncShedule.apxc

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

STEP 5: TEST AUTOMATION LOGIC:

1)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);
    }
}

```

2)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;
      }
    }
}
```

3)MaitenanceRequest.apxt

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

STEP 6)Test callout logic:-

1)WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-


    //Write a 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(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

2)WarehouseCalloutServiceMock.apxc

```
@isTest
```

```
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"qua









                    response.setStatusCode(200);

        return response;
    }
}
```

3)WarehouseCalloutServiceTest.apxc

```
       @IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
   @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

                 List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];
        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
    product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
    product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
    product2List.get(2).ProductCode);
    }
}

```

STEP 7)Test scheduling logic:

1)WarehouseCalloutServiceMock.cls:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"qua









        response.setStatusCode(200);

        return response;
    }
}
```

2)WarehouseSyncSchedule.cls :

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

3) WarehouseSyncScheduleTest.cls:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
   // implement scheduled code here
   //
   @isTest static void test() {
      String scheduleTime = '00 00 00 * * ? *';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
      String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
      CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
      System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not


      Test.stopTest();
   }
}
```