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

1) Get Started with Apex Trigger

AccountAddressTriggerCode :

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

   for(Account account:Trigger.New){
      if(account.Match_Billing_Address__c == true)
         account.ShippingPostalCode = account.BillingPostalCode;
      }
}
```

2) Bulk Apex Triggers Unit

ClosedOpportunityTriggerCode :

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

1)Get Started with Apex Unit Testing

VerifyDateCode :

```
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
	@TestVisible 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
	@TestVisible 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;
	}

}
```

TestVerifyDateCode :

```
@isTest
private class TestVerifyDate {

    @isTest static void Test_CheckDate_Case1(){
        Date D = verifyDate.CheckDates(date.parse('01/01/2022'), date.parse('01/05/2022'));
        System.assertEquals(date.parse('01/05/2022'),D);
    }

    @isTest static void Test_CheckDate_Case2(){
        Date D = verifyDate.CheckDates(date.parse('01/01/2022'), date.parse('05/05/2022'));
        System.assertEquals(date.parse('01/31/2022'), D);
    }

    @isTest static Void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('12/30/2021'));
        System.assertEquals(false, flag);
    }

    @isTest static Void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('02/02/2022'));
        System.assertEquals(false, flag);
    }

    @isTest static Void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('01/15/2022'));
        System.assertEquals(true, flag);
    }

    @isTest static Void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
    }
}
```

2) Test Apex Triggers Unit

RestrictContactByNameCode :

```
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');
                }

        }
}
```

TestRestrictContactByNameCode :

```
@isTest
public class TestRestrictContactByName {

   @isTest static void Test_insertupdateContact(){
      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 LastName "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
   }
}
```

3) Create Test Data for Apex Tests :

RandomContactFactoryCode :

```
public class RandomContactFactory {

   public static List<Contact> generateRandomContacts(Integer num,string lastname){
      List<Contact> contactList = new List<Contact>();
      for(Integer i=0;i<num;i++){
         Contact cnt = new Contact(FirstName = 'Test' +i, LastName = lastname);
         contactList.add(cnt);
      }

      return contactList;
   }

}
```

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

1)Quiz
2)Use Future Methods

AccountProcessorCode :

```
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;

    }
}
```

AccountProcessorTestCode :

```
@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='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

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

    }
```

```
}
```

3)Use Batch Apex

LeadProcessorCode :

```
global class LeadProcessor implements Database.Batchable<sObject> {
   global Integer count=0;

   global Database.QueryLocator start(Database.BatchableContext bc){
      return Database.getQueryLocator('SELECT ID,LeadSource FROM Lead');
  }
   global void execute(Database.BatchableContext bc,List<Lead>L_list){
      List<lead>L_list_new=new List<lead>();

      for(lead L:L_list){
         L.leadsource='Dreamforce';
         L_list_new.add(L);
         count+=1;
      }
      update l_list_new;
  }
   global void finish(Database.BatchableContext bc){
      system.debug('count =' + count);
   }
}
```

LeadProcessorTestCode :

```
@isTest
public class LeadProcessorTest {

   @isTest
   public static void testit(){
```

```
        List<lead> L_list = new List<lead>();

        for(Integer i=0 ;i<200; i++){
            Lead L = new lead();
            L.LastName ='name' + i;
            L.Company = 'Company';
            L.Status ='Random Status';
            L_list.add(L);

        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

4)Controp Processes with Queueable Apex

AddPrimaryContactCode :

```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact(Contact con,String state){
        this.con=con;
        this.state=state;
   }

    public void execute (QueueableContext context ){
        List<Account> accounts = [Select Id,Name,(Select FirstName,LastName,Id from
contacts)
                                  from Account where Billingstate= :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();
```

```
        for(Account acc:accounts) {
    Contact c = con.clone();
    c.AccountId = acc.Id;
    primaryContacts.add(c);
}

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

AddPrimaryContactTestCode :

```
@isTest
public class AddPrimaryContactTest{

    static testmethod void testQueueable(){

        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account '+i,Billingstate='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));
        }
        insert testAccounts;

        Contact testContact = new Contact(FirstName='John',LastName='Doe');
        insert testContact;
        AddPrimaryContact addit= new addPrimaryContact(testContact,'CA');
        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();
        System.assertEquals(50,[Select count() from Contact where accountId in(Select Id
from Account where Billingstate='CA')]);
```

```
    }
}
```

5)Schedule Jobs Using the Apex Scheduler

DailyLeadProcessorCode :

```
global class DailyLeadProcessor implements Schedulable {

    global void execute(SchedulableContext ctx) {


        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = ''
LIMIT 200];


        for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }

        //Updating all elements in the list.
        update leads;
    }

}
```

DailyLeadProcessorTestCode :

```
@isTest
private class DailyLeadProcessorTest {

    @isTest
    public static void testDailyLeadProcessor(){
```

```
        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;


        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new
DailyLeadProcessor());
        Test.stopTest();


        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource =
'Dreamforce' LIMIT 200];


        System.assertEquals(200, listResult.size());

    }
}
```

Apex Integration Services
:https://trailhead.salesforce.com/content/learn/modules/apex_integration_services?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst

1)Quiz
2)Apex REST Callouts

AnimalLocatorCode :

```
public class AnimalLocator {


    public class Animal {
```

```
            public Integer id;
            public String name;
            public String eats;
            public String says;
        }
    public Animal animal;

    public static string getAnimalNameById(integer id){
        string str;
        string URL='https://th-apex-http-callout.herokuapp.com/animals/'+id;

        http http=new http();
        httprequest Req=new httprequest();
        req.setEndpoint(URL);
        req.setMethod('GET');
        httpResponse Response=http.send(req);

        system.debug('Response Code: '+response.getStatusCode());
        system.debug('Response Body: '+response.getBody());
        //type ResultType= type.forName('Animals');
        //system.debug('Type: '+ ResultType);
        AnimalLocator obj= new AnimalLocator();
        obj=(AnimalLocator) System.JSON.deserialize(response.getBody(),
AnimalLocator.class);
        System.debug('Obj: '+obj.animal.name );
        str=obj.animal.name;
        System.debug('Name: '+str );
        return str;
    }
}
```

AnimalLocatorTestCode :

```
@istest

public class AnimalLocatorTest {
```

```
   testmethod static void  Restcallout(){


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

       string s=AnimalLocator.getAnimalNameById(1);


   }


}
```

AnimalLocatorMock Code :

```
@istest

public class AnimalLocatorMock implements HttpCalloutMock {


   public httpresponse respond(httprequest req){


       httpresponse Response=new httpresponse();

       response.setStatusCode(200);

       response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');


       return response;

   }

}
```

2)Apex SOAP Callouts

ParkServiceCode :

```
//Generated by wsdl2apex


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-service.herokuapp.com/service/parks';
        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;
    }
  }
}
```

ParkLocatorCode :

```
public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new
parkService.parksImplPort();
        return park.byCountry(country);
    }
}
```

ParkLocatorTestCode :

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new
```

```
ParkServiceMock());
        // Call the method that invokes a callout
        //Double x = 1.0;
        //Double result = AwesomeCalculator.add(x, y);

        String country = 'Germany';
        String[] result = ParkLocator.Country(country);


        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea
National Park', 'Hainich National Park', 'Bavarian Forest
National Park'}, result);
    }
}
```

ParkServiceMock Code :

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
```

```
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
        // start - specify the response you want to send
        parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
        response_x.return_x = new List<String>{'Hamburg Wadden
Sea National Park', 'Hainich National Park', 'Bavarian Forest
National Park'};


        //calculatorServices.doAddResponse response_x = new
calculatorServices.doAddResponse();
        //response_x.return_x = 3.0;
        // end
        response.put('response_x', response_x);
    }
}
```

4) Apex Web Services

AccountManagerCode :

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

```
    @HttpGet
    global static account getAccount() {

        RestRequest request = RestContext.request;

        String accountId =
request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
          request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from
contacts) from account where id = :accountId];
        List<contact> co = [select id, name from contact where
account.id = :accountId];
        system.debug('** a[0]= '+ a[0]);
        return a[0];

    }

}
```

AccountManagerTestCode :

```
@istest
public class AccountManagerTest {
@istest static void testGetContactsByAccountId() {
```

```
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+ recordId+'/Contacts';
request.httpMethod = 'GET';
RestContext.request = request;

Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount!= null);
System.assertEquals('Test record', thisAccount.Name);
}

// Helper method
static Id createTestRecord() {

// Create test record
Account accountTest = new Account(
Name='Test record');
insert accountTest;
Contact contactTest = new Contact(
FirstName='John',
LastName='Doe',
```

```
AccountId=accountTest.Id

);

return accountTest.Id;

}

}
```

APEX SPECIALIST SUPERBADGE :

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

1)Quiz

2)Automate Record Creation

MaintenanceRequestHelperCode :

```
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 (AggregateResultar : 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__cwp :closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__cwpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaitenanceRequestCode :

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

3)Synchronize Salesforce Data

WarehouseCalloutServiceCode :

```
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){
upsertwarehouseEq;
System.debug('Your equipment was synced with the warehouse
one');
System.debug(warehouseEq);
            }

        }
    }
}
```

4)Schedule Synchronization

WarehouseSyncScheduleCode :

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContextctx) {

WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

5)Test Automatic Logic

MaintenanceRequestHelperTestCode :

```
@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__ccreateVehicle(){
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__ccreateWorkPart(id
equipmentId,idrequestId){
Equipment_Maintenance_Item__cwp = 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__cworkP =
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__cworkPart = [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__cworkP = 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__cworkPart = [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);
    }
}
```

MaintenanceRequestCode :

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

6) Test Callout Logic

WarehouseCalloutServiceCode :

```
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){
upsertwarehouseEq;
System.debug('Your equipment was synced with the warehouse one');
System.debug(warehouseEq);
            }

        }
    }
}
```

WarehouseCalloutServiceTestCode :

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

`WarehouseCalloutServiceMockCode :`

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   // implement http mock callout
   global static HttpResponserespond(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,"quantity":
5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
response.setStatusCode(200);
      return response;
   }
}
```

7) Test Scheduling Logic

WarehouseSyncScheduleCode :

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContextctx) {

WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

`WarehouseSyncScheduleTestCode` :

```
@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 ');


   }
}
```