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

1. Get Started with Apex Trigger

AccountAddressTrigger Code :

```
trigger AccountAddressTrigger on Account (before insert, before update) {
   for(Account a : Trigger.new){
      if( !String.isBlank(a.BillingPostalCode) && a.Match_Billing_Address__c ){
        a.ShippingPostalCode = a.BillingPostalCode;
      }
   }
}
```

2. Bulk Apex Triggers Unit

ClosedOpportunityTriggerCode :

```
trigger ClosedOpportunityTrigger on Opportunity (before insert, before update) {
List<Task> newTask = new List <Task>();
   //Grab the Opportunity Id's from Opps that are Closed Won from the Context Variable and store them in opp
   for(Opportunity opp : [SELECT Id FROM Opportunity
                    WHERE StageName = 'Closed Won' IN :Trigger.New]){
   //Create a Follow Up Task against Id's that are stored in the variable opp
   newTask.add(new Task(Subject = 'Follow Up Test Task',
                Priority = 'High',
                WhatId = opp.Id));
   //Insert new Tasks
                      {insert newTask;
     }
   }
}
```

Apex Testing : https://trailhead.salesforce.com/content/learn/modules/apex_te st ing?trailmix_creator_id=trailblazerconnect&trailmix_slug=sales fo rce-developer-catalyst

1. Get Started with Apex Unit

Testing VerifyDate Code :

```
@isTest
private class VerifyDateTest {
@isTest static void CheckDatesTesttrue() {
        Date date1=date.today();
        Date date2=date1.addDays(29);
        Date t = VerifyDate.CheckDates(date1, date2);
        System.assertEquals(t, date2);
    }
    @isTest static void DateOver() {
        Date date1=date.today();
        Date date2=date1.addDays(31);
        Date t = VerifyDate.CheckDates(date1, date2);
        System.assertNotEquals(t, date1);
    }
}
```

TestVerifyDate Code :

```
@isTest
public class TestVerifyDate {
    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }
    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }
```

```
@isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
      System.assertEquals(false, flag);
   }
   @isTest static void Test_DateWithin30Days_case2(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2019'));
      System.assertEquals(false, flag);
   }
   @isTest static void Test_DateWithin30Days_case3(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
      System.assertEquals(true, flag);
   }
   @isTest static void Test_SetEndOfMonthDate(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
   }
}
```

2. TestApex Triggers Unit

RestrictContactByName Code :

```
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 Code :

```
@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 Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

3. Create Test Data for Apex Tests:

RandomContactFactory Code :

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer numcnt, string
lastname){
      List<Contact> contacts = new List<Contact>();
      for(Integer i=0;i<numcnt;i++){
         Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
         contacts.add(cnt);
      }
      return contacts;
   }
}
```

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

AccountProcessor Code :

```
public class AccountProcessor {
  @future
  public static void countContacts(List<Id> accountIds){
     List<Account> accountsToUpdate= new List<Account>();
     List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where Id in :accountIds];
     For (Account acc:accounts) {
       List<Contact> contactList = acc.Contacts;
       acc.Number_Of_Contacts__c= contactList.size();
       accountsToUpdate.add(acc);
     }
     update accountsToUpdate;
  }
}
```

AccountProcessorTest Code :

```
@IsTest
public 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

LeadProcessor Code :

```
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 );
    }
```

```
}
```

LeadProcessorTest Code :

```
@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

AddPrimaryContact Code :

```
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;
      }
   }
}
```

AddPrimaryContactTest Code :

```
@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

DailyLeadProcessor Code :

```
public class DailyLeadProcessor implements Schedulable{
   public void execute(SchedulableContext ctx){
      List<lead> leads = [Select Id From Lead Where LeadSource = NULL LIMIT 200];
      for(Lead l:leads){
         l.LeadSource = 'Dreamforce';
         update l;
      }
   }
}
```

DailyLeadProcessorTestCode :

```
@isTest
private class DailyLeadProcessorTest {
   static testMethod void testDailyLeadProcessor(){
      String CRON_EXP = '0 0 1 * * ?';
      List<Lead> lList = new List<lead>();
      for(Integer i=0; i<200; i++){
         lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));
      }
      insert lList;
      Test.startTest();
      String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
   }
}
```

Apex Integration Services
:https://trailhead.salesforce.com/content/learn/modules/apex_i nt egration_services?trailmix_creator_id=trailblazerconnect&trail mi x_slug=salesforce-developer-catalyst

1. Quiz
2. Apex REST Callouts

AnimalLocatorCode :

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
    Http http = new Http();
    HttpRequest req = new HttpRequest();
    req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
    req.setMethod('GET');
    Map<String, Object> animal= new Map<String, Object>();
    HttpResponse res = http.send(req);
       if (res.getStatusCode() == 200) {
    Map<String, Object> results =(Map<String, Object>)JSON.deserializeUntyped
(res.getBody());
   animal = (Map<String, Object>) results.get('animal');
      }
return (String)animal.get('name');
   }
}
```

AnimalLocatorTest Code :

```
@isTest
private class AnimalLocatorTest{
   @isTest static void AnimalLocatorMock1() {
      Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      String result = AnimalLocator.getAnimalNameById(3);
      String expectedResult = 'chicken';
      System.assertEquals(result,expectedResult );
   }
```

```
}
```

AnimalLocatorMock Code :

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type','application/json');
       response.setBody('{"animals":["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
       response.setStatusCode(200);
       return response;
    }
}
```

2)Apex SOAP Callouts

ParkService Code :

```
//Generated by wsdl2apex
public class ParkService {
public class byCountryResponse
{publicString[] 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 =
```

```
newString[]{'return_x'};
}
public class byCountry
{public
Stringarg0;
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 =
newString[]{'arg0'};
}
public class ParksImplPort {
public String endpoint_x = 'https://th-apexsoap-service.herokuapp.com/service/parks';
public Map<String,String> inputHttpHeaders_x;
public Map<String,String>
outputHttpHeaders_x;publicString
clientCertName_x;
public String clientCert_x;
public String
clientCertPasswd_x;
publicInteger timeout_x;
private String[] ns_map_type_info = new
String[]{'http://parks.services/',
'ParkService'};
public String[] byCountry(String arg0) {
ParkService.byCountry request_x =
newParkService.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.invok
e(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');
returnresponse_x.return_x;
}
}
}
```

ParkLocator Code :

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

ParkLocatorTest Code :

```
@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

AccountManager Code :

```
@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, Name from Contacts) from Account
where Id=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest Code :

```
@IsTest
public class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yoursInstance.my,salesforce.com/services/apexrest/Accounts/'+
                    recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
```

```
System.assertEquals('Test record', thisAccount.Name);
    }
    static Id createTestRecord(){
        Account accountTest = new Account(
            Name = 'Test record');
        insert accountTest;
        Contact contactTest = new Contact(
            FirstName='John',
            LastName = 'Doe',
            AccountId = accountTest.Id
        );
        insert contactTest;
        return accountTest.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE :
https://trailhead.salesforce.com/content/learn/modules/apex_integrat io n_services?trailmix_creator_id=trailblazerconnect&trailmix_slug=sale sforce-developer-catalyst

1. Quiz
2. Automate RecordCreation

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, Equipmentr.Maintenance_Cycle_c,(SELECT
Id,Equipment_c,Quantityc FROM Equipment_Maintenance_Items_r)
                                            FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment_r.Maintenance_Cycle_c)cycle FROM
Equipment_Maintenance_Item_c WHERE Maintenance_Request_c IN :ValidIds GROUP
BY Maintenance_Request__c];

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

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

              );

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

```
maintenanceCycles.get(cc.Id));
            } else {
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment_r.maintenance_Cycle_c);
            }

            newCases.add(nc);
          }

          insert newCases;

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

            }
          }
          insert ClonedWPs;
      }
   }
}
```

MaitenanceRequest Code :

```
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 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();
     }

}
```

4. Schedule Synchronization

WarehouseSyncSchedule Code :

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

5. Test AutomaticLogic

MaintenanceRequestHelperTest Code :

```
@istest
public with sharing class MaintenanceRequestHelperTest {
    private static final string STATUS_NEW = 'New';
   private static final string WORKING = 'Working';
   private static final string CLOSED = 'Closed';
   private static final string REPAIR = 'Repair';
   private static final string REQUEST_ORIGIN = 'Web';
   private static final string REQUEST_TYPE = 'Routine Maintenance';
   private static final string REQUEST_SUBJECT = 'Testing subject';

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

PRIVATE STATIC Product2 createEq(){
   product2 equipment = new product2(name = 'SuperEquipment',
                lifespan_months__C = 10,
                maintenance_cycle__C = 10,
                replacement_part__c = true);
return equipment;
}

PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
  case cs = new case(Type=REPAIR,
                 Status=STATUS_NEW,
                Origin=REQUEST_ORIGIN,
                Subject=REQUEST_SUBJECT,
                Equipment__c=equipmentId,
                Vehicle__c=vehicleId);
return cs;
}
```

```
  PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
   Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item_c(Equipment_c = equipmentId,
                                                Maintenance_Request__c = requestId);
return wp;
}


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

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

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

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

     test.startTest();
     somethingToUpdate.status = CLOSED;
     update somethingToUpdate;
     test.stopTest();
     Case newReq = [Select id, subject, type, Equipment_c, Date_Reported_c,
Vehicle_c, Date_Due_c
     from case
     where status =:STATUS_NEW];
```

```
Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newReq.Id];
      system.assert(workPart != null);
      system.assert(newReq.Subject != null);
      system.assertEquals(newReq.Type, REQUEST_TYPE);
      SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
      SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
      SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

@istest
private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

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

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

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

    list<case> allRequest = [select id
                from case];
```

```
Equipment_Maintenance_Item__c workPart = [select id
                                  from Equipment_Maintenance_Item__c
                                   where Maintenance_Request__c = :emptyReq.Id];

       system.assert(workPart != null);
       system.assert(allRequest.size() == 1);
}

@istest
private static void testMaintenanceRequestBulk(){
    list<Vehicle_C> vehicleList = new list<Vehicle_C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
    list<case> requestList = new list<case>();
    list<id> oldRequestIds = new list<id>();

    for(integer i = 0; i < 300; i++){
      vehicleList.add(createVehicle());
      equipmentList.add(createEq());
   }
  insert vehicleList;
  insert equipmentList;

  for(integer i = 0; i < 300; i++){
        requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
   }
      insert requestList;

      for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
   }
  insert workPartList;
  test.startTest();
```

```
for(case req : requestList){
     req.Status = CLOSED;
    oldRequestIds.add(req.Id);
}
update requestList;
test.stopTest();

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

list<Equipment_Maintenance_Item__c> workParts = [select id
              from Equipment_Maintenance_Item__c
              where Maintenance_Request__c in: oldRequestIds];
system.assert(allRequests.size() == 300);
  }
}
```

MaintenanceRequestHelper.apxc :-

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


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


            }
         }
      }
```

```
if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment_c, Equipmentr.Maintenance_Cycle_c,(SELECT
Id,Equipment_c,Quantityc FROM Equipment_Maintenance_Items_r)
                                        FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
        AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment_r.Maintenance_Cycle_c)cycle FROM
Equipment_Maintenance_Item_c WHERE Maintenance_Request_c IN :ValidIds GROUP
BY Maintenance_Request__c];

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

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

          );

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

          newCases.add(nc);
```

```
}

        insert newCases;

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

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

6. TestCallout 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){
             upsert warehouseEq;
             System.debug('Your equipment was synced with the warehouse one');
             System.debug(warehouseEq);
```

```
        }

      }
   }
}
```

WarehouseCalloutServiceTest.apxc :-

```
@isTest
private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
         Test.startTest();
         // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
   }
}
```

WarehouseCalloutServiceMock Code :

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

       System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
       System.assertEquals('GET', request.getMethod());

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity
":5,"name":"Generator 1000
```

```
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

7. Test Scheduling Logic

WarehouseSyncSchedule Code :

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

       WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest Code

```
@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 ');
    }
}
```