APEX TRIGGERS

Link: Apex Triggers | Salesforce Trailhead

1. Get Started with Apex Triggers :

AccountAddressTrigger Code:

```
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 :

```
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

Link: Apex Testing | Salesforce Trailhead

1. Get Started with Apex Unit Tests:

verifyDate code:

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

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

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

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

TestVerifyDate Code:

```
@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);
   }


}
```

2.Test Apex Triggers :

RestrictContactByName:

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

TestRestrictContactByName :

```
@isTest
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 DML',
res.getErrors()[0].getMessage());

   }

}
```

3. Create Test Data for Apex Tests :

RandomContactFactory code:

```
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;
   }

}
```

ASYNCHRONOUS APEX

Link: Asynchronous Apex | Salesforce Trailhead

1. Asynchronous Processing Basics : Quiz
2 . Use Future Methods :

AccountProcessor code :

```
public class AccountProcessor {

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

      List<Account> accounts = [SELECT Id, Name, Number_of_Contacts__c, (SELECT Id, Name
from Contacts) from Account WHERE Id in :accountIds];

      for (Account a: accounts) {
         a.Number_of_Contacts__c = a.Contacts.size();
      }

      update accounts;
   }
}
```

AccountProcessorTest code:

```
@isTest
public class AccountProcessorTest {

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

```
        Account newAccount = new Account(Name = 'Test Account 1');
        insert newAccount;
        Contact contact1 = new Contact(Lastname = 'ContactAccount1', AccountId =
newAccount.Id);
        insert contact1;
        Contact contact2 = new Contact(Lastname = 'ContactAccount2', AccountId =
newAccount.Id);
        insert contact2;
        System.debug(newAccount);
        System.debug(contact1);
        System.debug(contact2);

        Account newAccount2 = new Account(Name = 'Test Account 2');
        insert newAccount2;
        Contact contact3 = new Contact(Lastname = 'ContactAccount3', AccountId =
newAccount2.Id);
        insert contact3;
        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        accountIds.add(newAccount2.Id);

        System.debug(accountIds);

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

        List<Account> accountsUpdated = [SELECT Id, Name, Number_of_Contacts__c, (SELECT
Id, Name from Contacts) from Account WHERE Id in :accountIds];

        System.debug(accountsUpdated);

        System.assertEquals(accountsUpdated.get(0).Number_of_Contacts__c, 2);
        System.assertEquals(accountsUpdated.get(1).Number_of_Contacts__c, 1);

    }
}
```

3. Use Batch Apex :

LeadProcessor code :

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

   // instance member to retain state across transactions
   public Integer recordsProcessed = 0;

   public Database.QueryLocator start(Database.BatchableContext bc) {
      return Database.getQueryLocator('SELECT Id, LeadSource from Lead');
   }

   public void execute(Database.BatchableContext bc, List<Lead> leads) {
      for (Lead l: leads) {
         l.LeadSource = 'Dreamforce';
         recordsProcessed = recordsProcessed + 1;
      }
      update leads;
   }

   public void finish(Database.BatchableContext bc) {
      System.debug(recordsProcessed + ' records processed.');
   }
}
```

LeadProcessorTest code:

```
@isTest
public class LeadProcessorTest {

   @TestSetup
   static void makeData(){
      List<Lead> theLeads = new List<Lead>();
```

```
        for (Integer i = 0; i < 200; i++) {
            theLeads.add(new Lead(LastName='Lastname'+i, Company='Company'+i));
        }
        insert theLeads;
    }

    @isTest
    public static void test() {
        Test.startTest();
        LeadProcessor lP = new LeadProcessor();
        Id batchId = Database.executeBatch(lP);
        Test.stopTest();

        System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
    }
}
```

4. Control Processes with Queueable Apex :

AddPrimaryContact code :

```
public class AddPrimaryContact implements Queueable {

    private Contact myContact;
    private String state;

    public AddPrimaryContact(Contact myContact, String state) {
        this.myContact = myContact;
        this.state = state;
    }

    public void execute(QueueableContext context) {
        List<Account> accounts = [SELECT Id, Name from Account WHERE BillingState = :state
LIMIT 200];

        List<Contact> contactsToAdd = new List<Contact>();
```

```
        for (Account a: accounts) {
            Contact c = myContact.clone();
            c.AccountId = a.Id;
            contactsToAdd.add(c);
        }

        insert contactsToAdd;
    }

}
```

AddPrimaryContactTest code :

```
@isTest
public class AddPrimaryContactTest {

    @TestSetup
    static void makeData(){

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

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

    @isTest
    static void testQueueable() {
        Contact caContact = new Contact(LastName = 'CA Contact');
        AddPrimaryContact apc = new AddPrimaryContact(caContact, 'CA');
```

```
        Test.startTest();
        System.enqueueJob(apc);
        Test.stopTest();

        List<Account> CAAccounts = [select Id, (select Id from Contacts) from Account where
BillingState = 'CA'];
        Integer countContacts = 0;
        for (Account a: CAAccounts) {
            for (Contact c: a.Contacts) {
                if (c != null) {
                    countContacts = countContacts + 1;
                }
            }
        }

        System.assertEquals(50, countContacts);
    }
}
```

5. Schedule Jobs Using the Apex Scheduler :

DailyLeadProcessor code :

```
public class DailyLeadProcessor implements Schedulable {

 public void execute(SchedulableContext ctx) {

        List<Lead> leads = [select Id, LeadSource from Lead where LeadSource = '' LIMIT 200];
        if(leads.size()>0){

            List<Lead>newLeads= new List<Lead>();
            for(Lead lead: leads){

                lead.LeadSource='DreamForce';
                newLeads.add(lead);
```

```
 }
        update newLeads;
      }
   }
}
```

DailyLeadProcessorTest code :

```
@isTest
private class DailyLeadProcessorTest {
   public static String CRON_EXP= '0 0 0 2 6 ? 2022';
   static testmethod void testScheduledJob(){
      List<Lead>leads = new List<Lead>();
      for(Integer i=0; i<200;i++){
         Lead lead= new Lead(LastName='Test'+i,Company='Test Company'+i,Status='Open-Not
Contacted');
         leads.add(lead);

      }
      insert leads;
      Test.startTest();
      String jobId= System.schedule('Update LeadSource to DreamForce', CRON_EXP,new
DailyLeadProcessor());
      Test.stopTest();

   }
}
```

APEX INTEGRATION SERVICES

Link: Apex Integration Services | Salesforce Trailhead

1. Apex Integration Overview : Quiz

2. Apex REST Callouts :

AnimalLocator code :

```
public class AnimalLocator {
   public static String getAnimalNameById (Integer i) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
      request.setMethod('GET');
      HttpResponse response = http.send(request);


         Map<String, Object> r= (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
         Map<String, Object> animal= (Map<String, Object>)r.get('animal');
         System.debug('name:'+string.valueOf(animal.get('name')));

         return string.valueOf(animal.get('name'));




 }

}
```

AnimalLocatorTest code :

```
@isTest
public class AnimalLocatorTest {

    @isTest
    static void animalLoacatorTest1(){
        Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
        String actual=AnimalLocator.getAnimalNameById(1);
        String expected='moose';
        System.assertEquals(actual,expected);
    }
}
```

AnimalLocatorMock code :

```
@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. Apex SOAP Callouts :

ParkService code :

```
//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;
    }
  }
}
```

ParkLocator code :

```
public class ParkLocator {
  public static List<String> country(String country){
    ParkService.ParksImplPort parkservice=new ParkService.ParksImplPort();
    return parkservice.byCountry('country');
  }

}
```

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
      String country = 'United States';
      List<String> result = ParkLocator.country(country);
      List<String> parks = new List<String>();

      parks.add('Yellowstone');
      parks.add('Yosemite');
      parks.add('Highland Valley');

      System.assertEquals(parks, 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

         List<String> parks = new List<String>();
         parks.add('Yellowstone');
         parks.add('Yosemite');
         parks.add('Highland Valley');

         ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();

         response_x.return_x = parks;

         //end

         response.put('response_x', response_x);

        }
}
```

4. Apex Web Services :

AccountManager code :

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

```
    }

}
```

AccountManagerTest code :

```
@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();
        // Set request properties

        request.requestUri
='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request= request;
        Account myAcct=AccountManager.getAccount();
        System.assert(myAcct!=null);

        System.assertEquals('TestAccount',myAcct.Name);
```

APEX SPECIALIST SUPERBADGE

Link: Apex Specialist | Salesforce Trailhead

1. Credentials security : Quiz

2. Automate Record Creation :

MaintenanceRequestHelper code :

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


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



            }
         }
      }

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

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

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

            );

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

            newCases.add(nc);
        }

        insert newCases;

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

            }
        }
        insert ClonedWPs;
```

```
        }
    }
}
```

MaintenanceRequest code :

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

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

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

    }

}
```

3. Synchronize Salesforce Data with an External System :

WarehouseCalloutService code :

```
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);
            }

        }
    }
}
```

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

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

// Now Run the code ane Check the Challege

4. Schedule Synchronization :

WarehouseSyncSchedule code :

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

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

5. Test Automation Logic :

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 code :

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


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

```
            }
        }
    }

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

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

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

            );

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

```
            newCases.add(nc);
        }

        insert newCases;

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

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaintenanceRequest code :

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

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

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

  }

}
```

// NOW run all code and check the Challenge

6. Test Callout Logic :

WarehouseCalloutService code :

```
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 code :

```
@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,"na
me":"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 ');


    }
}
```

THE END