Apex Triggers

GET STARTED WITH APEX TRIGGERS:

1.AccountAddressTrigger.apxt:

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

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

BULK APEX TRIGGERS:

1.ClosedOpportunityTrigger.apxt:

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

    for(Opportunity opp: Trigger.new){
        if(opp.StageName == 'ClosedWon'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId =
opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;

    }
}
```

Apex Testing

GET STARTED WITH APEX UNIT TEST:

1.VerifyDate.apxc:

```
---------------------------
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }
         @TestVisible private static Boolean DateWithin30Days(Date date1, Date
date2) {
                //check for date2 being in the past
        if( date2 < date1) { return false; }
        Date date30Days = date1.addDays(30);
if( date2 >= date30Days ) { return false; }
                else { return true; }
        }
         @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;
        }

}
```

2.TestVerifyDate.apxc:

```
--------------------------------
@isTest
private 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('12/30/2029'));
       System.assertEquals(false, flag);

   }
    @isTest static void Test_DateWithin30Days_case2(){
       Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2020'));
       System.assertEquals(false, flag);
    }

   @isTest static void Test_DateWithin30Days_case3(){
       Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('01/15/2020'));
       System.assertEquals(true, flag);
    }

   @isTest static void Test_SetEndOfMonthDatea(){
       Date  returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
   }
}
```

TEST APEX TRIGGERS:

1.RestrictContactByName.apxt:

--

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

        }



}
```

CREATE TEST DATA FOR APEX TESTS:

1.RandomContactFactory.apxc:

--

```
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

USE FUTURE METHODS:

1.AccountProcessor.apxc:

```
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;
   }



}
```

2.AccountProcessorTest.apxc:

```
@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();
   }
}
```

USE BATCH APEX:

1.LeadProcessor.apxc:

```
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);
    }

}
```

2.LeadProcessorTest.apxc:

```
@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();

    }
}
```

CONTROL PROCESSES WITH QUEUEABLE APEX:

1.AddPrimaryContact.apxc:

```
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;
        }
    }

}
```

2.AddPrimaryContactTest.apxc:

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

SCHEDULE JOBS USING APEX SCHEDULER:

1.DailyLeadProcessor.apxc:

--

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = ''];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

2.DailyLeadProcessorTest.apxc:

```
@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, LeadSource = '', 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

APEX REST CALLOUTS:

1.AnimalLocator.apxc:

```
public class AnimalLocator {

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

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



  }

}
```

2.AnimalLocatorMock.apxc:

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

3.AnimalLocatorTest.apxc:

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

APEX SOAP CALLOUTS:

1.ParkService.apxc:

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

2.ParkServiceMock.apxc:

```
@isTest
global class ParkserviceMock implements webserviceMock{
  global void doInvoke(
    object stub,
     object request,
     Map<String, object> response,
     String endpoint,
     string soapAction,
           String requestName,
           String responseNS,
     String responseName,
        String responseType){
           parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
           response_x.return_x = new List<String>{'Me','You','Her'};
            response.put('response_x', response_x);
       }
   }
```

3.ParkLocatorTest.apxc:

```
@isTest
public class ParkLocatorTest {
          @isTest
   static void testCallout(){
       Test.setMock(WebServiceMock.class, new ParkServiceMock());
       String country = 'USA';
       System.assertEquals(new List<String>{'Me','You','Him'},
ParkLocator.country(country));
   }
}
```

APEX WEB SERVICES:

1.AccountManager.apxc:

```
@RestResource(urlMapping='/Accounts/*/contacts')
Global with sharing class AccountManager {
   @HttpGet
   global static Account getAccount(){
       RestRequest request = RestContext.request;
  String accountId =
request.requestURI.substringBetween('Accounts/','/contacts');
       Account acc = [select Id,Name,(select Id,Name from Contacts) from Account
where Id = :accountId];
       system.debug('Account and Related Contacts->>>>'+acc);
       return acc;
   }
}
```

2.AccountManagerTest.apxc:

```
@isTest
private class AccountManagerTest {
   static Id createTestRecord(){
      Account TestAcc = new Account(Name='Test Account', Phone='8786757657');
      insert TestAcc;
      List<Contact> conList = new List<Contact>();
      Contact TestCon = new Contact();
      for(Integer i=1;i<=3;i++){
         TestCon.LastName = 'Test Contact'+i;
         TestCon.AccountId = TestAcc.Id;
         insert conList;//Its not best practice but I have use it for testing purposes
      }
      return TestAcc.Id;
   }
   @isTest static void getAccountTest(){
      Id recordId = createTestRecord();
      RestRequest request = new RestRequest();
      request.requestURI =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/' + recordId
+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisAcc = AccountManager.getAccount();
      system.assert(thisAcc != null);
      system.assertEquals('Test Account', thisAcc.Name);
   }
}
```

APEX SPECIALIST SUPERBADGE

AUTOMATE RECORD CREATION:

1)MaintenanceRequest.apxt:

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

2)MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = 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>();

            //calculate the maintenance request due dates by using the maintenance
cycle defined on the related equipment records.
            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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
           for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
              Equipment_Maintenance_Item__c item = clonedListItem.clone();
              item.Maintenance_Request__c = nc.Id;
              clonedList.add(item);
           }
```

```
            }
            insert clonedList;
        }
    }
}
```

SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

1)WarehouseCalloutService.apxc:

--

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';  @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                product2.Cost__c = (Integer) mapJson.get('cost');
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

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

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

}
```

TEST AUTOMATION LOGIC:

1)MaintenanceRequestHelperTest.apxc:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
   private static Vehicle__c createVehicle(){
      Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
      return vehicle;
   }
   private static Product2 createEquipment(){
      product2 equipment = new product2(name = 'Testing equipment',
                                        lifespan_months__c = 10,
                                        maintenance_cycle__c = 10,
                                    replacement_part__c = true);
        return equipment;
    }
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }
    private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }

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

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

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;
        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();

        Case newCase = [Select id,
                          subject,
                          type,
                          Equipment__c,
                          Date_Reported__c,
                          Vehicle__c,
                          Date_Due__c
                         from case
                         where status ='New'];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }

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

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

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

        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;

        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

        list<case> allCase = [select id from case];

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList =
new list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){
            
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipment
List.get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                              from case
                              where status ='New'];



        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c in:
oldCaseIds];

        system.assert(newCase.size() == 300);
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

2)MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }

        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = 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'));
            }
            List<Case> newCases = new List<Case>();
            for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
                //define the due date by applying the shortest maintenance cycle to
today’s date.
                //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> clonedList = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c item = clonedListItem.clone();
                    item.Maintenance_Request__c = nc.Id;
                    clonedList.add(item);
            }
          }
          insert clonedList;
        }
    }
}
```

3)MaintenanceRequest.apxt:

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

TEST CALLOUT LOGIC:

1)WarehouseCalloutService.apxc:

--

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

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

```
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            product2.Cost__c = (Integer) mapJson.get('cost');
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

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

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

}
```

2)WarehouseCalloutServiceTest.apxc:

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

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}
```

3)WarehouseCalloutServiceMock.apxc:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request) {

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"qu
antity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":
"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d6
6226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

TEST SCHEDULING LOGIC:

1)WarehouseSyncSchedule.apxc:

--

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

2)WarehouseSyncScheduleTest.apxc:

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

        Test.stopTest();
    }
}
```