Salesforce Developer Catalyst

Apex Triggers

Get Started With Apex Triggers:

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:

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

        List<Task> tasklist = new List<Task>();

        for(Opportunity opp: Trigger.New){

                if(opp.StageName == 'Closed Won'){

                        tasklist.add(new Task(Subject = 'Follow Up Test Task',
WhatId=opp.Id));

                }

        }

        if(tasklist.size()>0){

                insert tasklist;

        }

}
```

Apex Testing

Get Started with Apex Unit Test:

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) {
        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;
        }


}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
    @isTest static void Test_CheckDats_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_CheckDats_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/20'));
        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('01/15/2020'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}
```

Test Apex Triggers:

RestrictContactByName.apxt

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

Create Test Data For Apex Tests:

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:

AccountProcessor.apxc

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){
               List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id
         from Contacts) from Account where Id in :accountIds];
                 For(Account acc : accList){
                  acc.Number_Of_Contacts__c = acc.Contacts.size();
                 }
              update accList;
   }
}
```

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
   public static testmethod void testAccountProcessor(){
      Account a = new Account();
      a.Name = 'Test Account';
      insert a;
      Contact con = new Contact();
      con.FirstName = 'Binary';
      con.LastName = 'Programming';
      con.AccountId = a.Id;
      insert con;
      List<Id> accListId = new List<Id>();
```

```
        accListId.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();
        Account acc = [Select Number_Of_Contacts__c from Account where Id = :a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }
}
```

Use Batch Apex:

LeadProcessor.apxc

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {
        global Integer recordsProcessed = 0;
        global Database.QueryLocator start(Database.BatchableContext bc) {
            return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
        }

        global void execute(Database.BatchableContext bc, List<Lead> scope){
                List<Lead> leads = new List<Lead>();
                for (Lead lead : scope) {
                        lead.LeadSource = 'Dreamforce';
                        recordsProcessed = recordsProcessed + 1;
                }
                update leads;
        }
        global void finish(Database.BatchableContext bc){
                System.debug(recordsProcessed + ' records processed. Shazam!');
        }
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
 @testSetup
   static void setup() {
      List<Lead> leads = new List<Lead>();
      for (Integer i=0;i<200;i++) {
         leads.add(new Lead(LastName='Lead '+i,
            Company='Lead', Status='Open - Not Contacted'));
      }
      insert leads;
   }
   static testmethod void test() {
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp, 200);
      Test.stopTest();
      System.assertEquals(200, [select count() from lead where LeadSource =
'Dreamforce']);
   }
}
```

}

Control Processes with Queueable Apex:

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id, FirstName,
LastName from contacts ) from ACCOUNT where BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
                 Contact cont = c.clone(false,false,false,false);
                 cont.AccountId =  acc.id;
                 lstContact.add( cont );
         }
         if(lstContact.size() >0 )
         {
             insert lstContact;
         }
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';
        AddPrimaryContact apc = new AddPrimaryContact(co, state);
        Test.startTest();
         System.enqueueJob(apc);
         Test.stopTest();
     }
 }
```

Schedule Jobs Using Apex Scheduler:

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable{
    public void execute(SchedulableContext sc){
        List<Lead> leadObj = [Select Id from Lead where LeadSource = null limit 200];
        for(Lead l : LeadObj){
           l.LeadSource = 'DreamForce';
            update l;
        }
    }
}
```

DailyLeadProcessorTest.apxc

```
@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());
        Test.stopTest();
    }
}
```

Apex Integration Services

Apex Rest Callouts:

AnimalLocator.apxc

```
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');
    }
}
```

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('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
```

```
        response.setStatusCode(200);

        return response;

    }

}
```

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:

ParkService.apxc

```
public class ParkService {

    public class byCountryResponse {

        public String[] return_x;

        private String[] return_x_type_info = new String[] {'return',
'http://parks.services/', null, '0', '-1', 'false'};

        private String[] apex_schema_type_info = new String[] {'http://parks.services/',
'false', 'false'};

        private String[] field_order_type_info = new String[]{'return_x'};

    }

    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[] {'arg0', 'http://parks.services/', null,
'0', '1', 'false'};
        private String[] apex_schema_type_info = new String[] {'http://parks.services/',
'false', 'false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
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;
        }
    }
}
```

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>{'Yellowstone', 'Mackinac National
Park', 'Yosemite'};
        response.put('response_x', response_x);
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

Apex Web Services:

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {
    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri = 'https://na1.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 TestAcc = new Account(Name='Test record');
            insert TestAcc;
            Contact TestCon= new Contact(LastName='Test',AccountId = TestAcc.id);
            return TestAcc.Id;
    }
}
```

Apex Specialist Superbadge

Automate Record Creation:

MaintenanceRequest.apxt

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

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
        if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id,
Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r) FROM
Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){

            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));

        }

            for(Case cc : closedCasesM.values()){

                Case nc = new Case (

                    ParentId = cc.Id,

                    Status = 'New',

                    Subject = 'Routine Maintenance',

                    Type = 'Routine Maintenance',

                    Vehicle__c = cc.Vehicle__c,

                    Equipment__c =cc.Equipment__c,

                    Origin = 'Web',

                    Date_Reported__c = Date.Today()

                );

                If (maintenanceCycles.containskey(cc.Id)){

                    nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));

                }

                newCases.add(nc);

            }

           insert newCases;

           List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();

           for (Case nc : newCases){

                for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
```

```
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);
            }
        }
        insert ClonedWPs;
    }
  }
}
```

Synchronization Salesforce Data With An External System:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {


    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    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);
      }
    }
  }
}
```

Schedule Synchronization Using Apex Code:

```
WarehouseSyncSchedule.apxc
global class WarehouseSyncSchedule implements Schedulable {
  global void execute(SchedulableContext ctx) {
    WarehouseCalloutService.runWarehouseEquipmentSync();
  }
}
```

Test Automation Logic:

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(equipmentLi
st.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);
    }
}
```

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {

    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {

        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){

            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

                    validIds.add(c.Id);

                }

            }

        }

        if (!validIds.isEmpty()){

            List<Case> newCases = new List<Case>();

            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id,
Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r) FROM
Case WHERE Id IN :validIds]);

            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c];

            for (AggregateResult ar : results){

                    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'),
(Decimal) ar.get('cycle'));

        }

            for(Case cc : closedCasesM.values()){

                Case nc = new Case (

                    ParentId = cc.Id,
                    Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()
                );
                If (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
                }
                newCases.add(nc);
            }
            insert newCases;
            List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);
                }
            }
            insert ClonedWPs;
        }
    }
}
```

MaintenanceRequest.apxt

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

Test Callout Logic:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';


    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);
            }
        }
    }
}
```

WarehouseCalloutServiceText.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.apxc

```
@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,"quanti
ty":5,"name":"Generator 1000 kW", "maintenanceperiod":365, "lifespan":120,
"cost":5000, "sku":"100003"}]');

        response.setStatusCode(200);

        return response;

    }

}
```

Test Scheduling Logic:

WarehouseSyncSchedule.apxc

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest.apxc

```
@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();
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
    }
}
```