Apex Triggers

Get Started with Apex Triggers

```
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

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

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

    for(Opportunity Opp:Trigger.New){
        if(Trigger.isInsert || Trigger.isUpdate)
          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 Tests

apex class

```
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;
   }
}
```

Test Apex Triggers

apex class

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

test class

```
@isTest
private class TestRestrictContactByName {

   @isTest static void testInvalidName() {
      //try inserting a Contact with INVALIDNAME
      Contact myConact = new Contact(LastName='INVALIDNAME');
      insert myConact;

      // Perform test
      Test.startTest();
      Database.SaveResult result = Database.insert(myConact, false);
      Test.stopTest();
      // Verify
      // In this case the creation should have been stopped by the trigger,
      // so verify that we got back an error.
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
      System.assertEquals('Cannot create contact with invalid last name.',
                           result.getErrors()[0].getMessage());

   }
}
```

Create Test Data for Apex Tests

apex class

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
      List<Contact> contactList = new List<Contact>();

      for(Integer i=0;i<numContactsToGenerate;i++) {
         Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
         contactList.add(c);
```

```
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }

}
```

Asynchronous Apex

Use Future Methods

apex class

```
public class AccountProcessor
{
  @future
  public static void countContacts(Set<id> setId)
  {
     List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from contacts )
from account where id in :setId ];
     for( Account acc : lstAccount )
     {
        List<Contact> lstCont = acc.contacts ;

        acc.Number_of_Contacts__c = lstCont.size();
     }
     update lstAccount;
  }
}
```

test class

```
@isTest
private class AccountProcessorTest {

    @isTest
    private static void countContactsTest() {
```

```
        List<Account> accounts=new List<Account>();
        for(Integer i=0;i<300;i++){
            accounts.add(new Account(Name='TestContact'+i));
        }
        insert accounts;

        List<Contact> contacts=new List<Contact>();
        List<Id> accountids=new List<Id>();
        for(Account acc:accounts){
            contacts.add(new
Contact(FirstName=acc.Name,LastName='TestContact',AccountId=acc.Id));
            accountids.add(acc.Id);
        }
        insert contacts;

        Test.startTest();
        AccountProcessor.countContacts(accountids);
        Test.stopTest();
    }

}
```

Use Batch Apex

apex class

```
public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name FROM Lead]);
    }
    public void execute(Database.BatchableContext dbc,List<Lead> leads){
        for(Lead l:leads){
            l.LeadSource='Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext dbc){
        System.debug('Done');
    }
}
```

test class

```
@isTest
private class LeadProcessorTest {
    @isTest
    private static void testBatchClass(){
        List<Lead> leads=new List<Lead>();
        for(Integer i=0;i<200;i++){
            leads.add(new Lead(LastName='Parichha',Company='Salesforce'));

        }
        insert leads;

        Test.startTest();
        LeadProcessor lp=new LeadProcessor();
        Id batchid=Database.executeBatch(lp,200);
        test.stopTest();

        List<Lead> updatedleads=[SELECT Id FROM Lead WHERE Leadsource='Dreamforce'];
        System.assertEquals(200, updatedleads.size());
    }
}
```

Control Process With Queueable Apex

apex class

```
public without sharing class AddPrimaryContact implements Queueable {
    private Contact contact;
    private String state;

    public AddPrimaryContact(Contact inputcontact,String inputstate){
        this.contact=inputcontact;
        this.state=inputstate;
    }
    public void execute(QueueableContext context){
        List<Contact> contacts=new List<Contact>();

        List<Account> accounts=[SELECT Id FROM Account WHERE BillingState= :state LIMIT 200];
```

```
        for (Account acc: accounts){
            Contact clonecontact=contact.clone();
            clonecontact.AccountId=acc.Id;
            contacts.add(clonecontact);
        }
        insert contacts;
    }

}
```

test class

```
@isTest
private class AddPrimaryContactTest {
    @isTest
    private static void testQueueableClass(){
        List<Account> accounts=new List<Account>();
        for(Integer i=0;i<500;i++){
            Account acc=new Account(Name='Test Account');
            if(i<250){
                acc.BillingState='NY';
            }
            else{
                acc.BillingState='CA';
            }
            accounts.add(acc);
        }
        insert accounts;


        Contact contact=new Contact(FirstName='Deependra',LastName='Parichha');
```

```
        insert contact;



        Test.startTest();

        Id jobId=System.enqueueJob(new AddPrimaryContact(contact,'CA'));

        Test.stopTest();



        List<Contact> contacts=[SELECT Id FROM Contact WHERE
Contact.Account.BillingState='CA' ];

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

    }

}
```

Schedule Jobe Using the Apex Scheduler

apex class

```
public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext ctx){
        List<Lead> leads=[SELECT id,LeadSource FROM Lead WHERE LeadSource=null LIMIT 200];
        for(Lead l:leads){
            l.LeadSource='Dreamforce';
        }
        update leads;
    }
}
```

test class

```
@isTest
private class DailyLeadProcessorTest {
    private static String CRON_EXP='0 0 0 ? * * *';

    @isTest
    private static void testSchedulabelClass(){
        List <Lead> leads=new List<Lead>();
```

```
        for(Integer i=0;i<500;i++){
            if(i<250){
                leads.add(new Lead(LastName='Parichha',Company='Salesforce'));
            }
            else{
                leads.add(new Lead(LastName='Parichha',Company='Salesforce',LeadSource='Other'));
            }

        }
        insert leads;

        Test.startTest();
        String jobid=System.schedule('Process Leads',CRON_EXP,new DailyLeadProcessor());
        Test.stopTest();

        List<Lead> updatedLeads=[SELECT ID,LeadSource FROM Lead WHERE
LeadSource='Dreamforce'];
        System.assertEquals(200,updatedLeads.size());

        List<CronTrigger> cts=[SELECT Id,TimesTriggered,NextFireTime FROM CronTrigger WHERE
Id=:jobid ];
        System.debug('Next Fire Time'+cts[0].NextFireTime);
    }
}
```

Apex Integration Services

Apex Rest Callouts

apex class

```
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) {
            // Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(res.getBody());


      animal = (Map<String,Object>) results.get('animal');
        }
return (String)animal.get('name');
    }
}
```

test class

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

unit tests

```
@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"]}');
        response.setStatusCode(200);
        return response;
    }

}
```

Apex Soap Callouts

apex class

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

test class

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

unit tests

```
@isTest
global class ParkServiceMock implements WebServiceMock {
  global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {
      // start - specify the response you want to send
      ParkService.byCountryResponse response_x=new ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
```

```
        // end
        response.put('response_x', response_x);
    }
}
```

Apex Web Services

apex class

```
@RestResource(urlMapping='/Account/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result =  [SELECT Id,Name,(SELECT Id,Name FROM Contacts)
                    FROM Account
                    WHERE Id = :accountId];
        return result;

    }
}
```

test class

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =

'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
```

```
    }

    // Helper method
    static Id createTestRecord() {
        // Create test record
        Account caseTest = new Account(
            Name='Test record');
        insert caseTest;
        Contact contactcase=new
Contact(FirstName='Deependra',LastName='Parichha',AccountId=casetest.Id);
        insert contactcase;
        return caseTest.Id;
    }
}
```

APEX SPECIALIST-SUPERBADGE

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));
            } else {
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            }

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedWPs = new
```

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

            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaintenanceRequest

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

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

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

   }

}
```

WarehouseCalloutService

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

```
    //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

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

```
        }
      }
    }

    public static void execute (QueueableContext context){
      runWarehouseEquipmentSync();
    }

}
```

WarehouseSyncSchedule

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

MaintenanceRequestHelperTest

```
@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

```
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

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

WarehouseCalloutService

```
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

```
@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]);
```

```
    }
}



@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

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

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

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":
"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
```

```
}

global class WarehouseCalloutServiceMock implements Schedulable {
   global void execute(SchedulableContext ctx) {

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

WarehouseSyncScheduleTest

```
@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 ');


   }
}
```