SALESFORCE DEVELOPER CATALYST

APEX TRIGGERS >

1.Get Started with Apex Triggers

AccountAddressTrigger

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

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

2.Bulk Apex Triggers

ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (before 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 >

1.Get Started With Apex Unit Tests

VerifyDate

TestVerifyDate

```
@isTest
public class TestVerifyDate {
   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
                System.assertEquals(date.parse('01/05/2020'), D);
   }
   @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
                System.assertEquals(date.parse('01/31/2020'), D);
   }
   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
                System.assertEquals(false, flag);
   }
   @isTest static void Test_DateWithin30Days_case2(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
                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(false, flag);
   }
   @isTest static void Test_SetEndOfMonthDate(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
```

```
    }
}
```

2.Test Apex Triggers

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert, before update) {

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

TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
```

```
   }}
```

3.Create Test Data for Apex Tests

RandomContactFactory

```
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>

1.Use Future Methods

AccountProcessor

```
public class AccountProcessor {

@future
        public static void countContacts (List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN:
accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for (Account account: accounts){
                account.Number_of_Contacts__c = [Select count() from Contact Where
AccountId =: account.Id];
                System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
```

```
updatedAccounts.add(account);

}

update updatedAccounts;

}

}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {
        @isTest
        public static void testcountContacts(){
                Account a = new Account();
                a.Name = 'Test Account';
                Insert a;

                Contact c = new Contact();
                c.FirstName = 'Bob';
                c.LastName = 'Willie';
      c.AccountId = a.Id;

                Contact c2 = new Contact();
                c2.FirstName = 'Tom';
                c2.LastName = 'Cruise';
                c2.AccountId = a.Id;

                List<Id> acctIds = new List<Id>();
                acctIds.add(a.Id);

                Test.startTest();
                AccountProcessor.countContacts(acctIds);
                Test.stopTest();
        }
```

}

2.Use Batch Apex

LeadProcessor

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

        public Database.QueryLocator start(Database.BatchableContext bc) {

                // collect the batches of records or objects to be passed to execute
                return Database.getQueryLocator([Select LeadSource From Lead ]);
        }
        public void execute(Database.BatchableContext bc, List<Lead> leads){

                // process each batch of records
                  for (Lead Lead : leads) {
          lead.LeadSource = 'Dreamforce';

        }
                update leads;
         }

         public void finish (Database.BatchableContext bc){

         }

}
```

LeadProcessorTest

```
@isTest

public class LeadProcessorTest {

   @testSetup

        static void setup() {
```

```
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName = 'LastName'+counter;
            lead.Company ='demo'+counter;
            leads.add(lead);
        }
    insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId= Database.executeBatch(leadProcessor);
  Test.stopTest();
 }
}
```

3.Control Processes With Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable{
```

```
    private Contact con;
    private String state;

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

    }
    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from
contacts) from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts= new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);

        }
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }
}
```

AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<500;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')]);

    }
```

4.Schedule Jods Using The Apex Scheduler

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
          global void execute(SchedulableContext ctx){
        List<lead>leadstoupdate = new List<lead>();
        List<Lead> leads = [Select id from Lead Where LeadSource = NULL Limit 200];

        for(Lead l:leads){
           l.LeadSource = 'Dreamforce';
           leadstoupdate.add(l);
           }
        update leadstoupdate;
    }
}
```

DailyLeadProcessorTest

```
@isTest
```

```
private class DailyLeadProcessorTest{
   public static String CRON_EXP = '0 0 15 7 ? 2022';

   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();

      for(Integer i=0;i<200;i++){
         Lead l = new Lead(FirstName = 'First' + i, LastName = 'LastName', Company =
'The Inc');
         leads.add(l);
      }

      insert leads;

      Test.startTest();
      String testScheduledJob = System.schedule('ScheduledApexTest', CRON_EXP, new
DailyLeadProcessor());
      Test.stopTest();

      List<Lead> checkleads = new List<Lead>();
      checkleads = [Select Id from Lead Where LeadSource = 'Dreamforce' and Company
='The Inc'];

      System.assertEquals(200,checkleads.size(),'Leads were not creted');
   }
}
```

//

Apex Integration Services>

1. Apex Rest Callouts

AnimalLocator

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

AnimalLocatorTest

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

AnimalLocatorMock

```
@isTest
```

```
global class AnimalLocatorMock implements HttpCalloutMock {

// Implement this interface method

        global HTTPResponse respond(HTTPRequest request) {

                // Create a fake response

                HttpResponse response = new HttpResponse();

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

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

                response.setStatusCode(200);

                return response;

        }

}
```

2. Apex SOAP Callouts

ParkLocator

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

ParkLocatorTest

```
@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>{'Yellowatone','Mackinac National
Park','Yosemite'};
                System.assertEquals(parks, result);
    }
}
```

ParkServiceMock

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

ParkService

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

3. Apex Web Servcies

AccountManager

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {

  @HttpGet
  global static Account getAccount(){
    RestRequest request = RestContext.request;
    String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
    Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where Id=:accountId Limit 1];
    return result;
  }
}
```

AccountManagerTest

```
@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

      // Call the method to test
      Account  acc = AccountManager.getAccount();

      // Verify results
      System.assert(acc != null);
   }

   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;

      Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
      Insert con;

      return acc.Id;
   }
}
```

//

Apex Specialist SuperBadge>

1.Automates Record Creation

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after update) {
   //ToDo: Call MaintenanceRequestHelper.updateWorkOrders
   if(trigger.isAfter){
      MaintenanceRequestHelper.updateWorkOrders();
   }
}
```

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
   public static void updateWorkOrders() {
      List<case> newCaseList = new List<case>();
      Integer avgAmount=10000;

      List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
      List<case> caseList = [SELECT id,Vehicle__c,Subject,ProductID,Product__c, (SELECT
id from Equipment_Maintenance_Items__r) from case where status='closed' and Type
IN ('Repair', 'Routine Maintenance') and ID IN :Trigger.new LIMIT 200];
      Map<id,Equipment_Maintenance_Item__c> equip = new
map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c,
Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle__c from
Equipment_Maintenance_Item__c ]);
      for(case c: caseList){
         case newCase = new Case();
         newCase.Type = 'Routine Maintenance';
         newCase.Status = 'New';
         newCase.Vehicle__c = c.Vehicle__c;
         newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request' :
c.Subject;
         newCase.Date_Reported__c = Date.today();
         newCase.ProductId = c.ProductId;
         newCase.Product__c = c.Product__c;
         newCase.parentID = c.Id;
```

```
        for(Equipment_Maintenance_Item__c emi : c.Equipment_Maintenance_Items__r ){
            avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment__r.Maintenance_Cyc
le__c));
            newEMI.add(new Equipment_Maintenance_Item__c(
              Equipment__c = equip.get(emi.id).Equipment__c,
              Maintenance_Request__c = c.id,
              Quantity__c = equip.get(emi.id).Quantity__c));
          }
          Date dueDate = date.TODAY().adddays(avgAmount);
          newCase.Date_Due__c =dueDate;
          newCaseList.add(newCase);

      }
      if(newCaseList.size()>0){
        Database.insert(newCaseList);
      }

      for(Case c2: newCaseList){
        for(Equipment_Maintenance_Item__c emi2 : newEmi){
          if(c2.parentID == emi2.Maintenance_Request__c){
            emi2.Maintenance_Request__c = c2.id;
          }
        }
      }

      if(newEmi.size()>0){
        Database.insert(newEmi);
      }
    }
}
```

2.Synchronize Salesforce data with an external system

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();
  }

}
```

3.Schedule synchronization using Apex code

WarehouseSyncSchedule

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

4.Test Automation Logic

MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest {
  @istest
  public static void BulkTesting(){
```

```
    product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10,
Replacement_Part__c = true);

    Database.insert(pt2);


    List<case> caseList = new List<case>();
    for(Integer i=0;i<300;i++){
      caseList.add(new case(
        Type = 'Routine Maintenance',
        Status = 'Closed',
        Subject = 'testing',
        Date_Reported__c = Date.today(),
        ProductId = pt2.id
      ));
    }
    if(caseList.size()>0){
      Database.insert(caseList);
      System.debug(pt2.id);
      System.debug(caseList.size());
    }


    List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
    for(Integer i=0;i<5;i++){
      newEMI.add(new Equipment_Maintenance_Item__c(
        Equipment__c = pt2.id,
        Maintenance_Request__c = caseList[1].id,
        Quantity__c = 10));
    }
    if(newEmi.size()>0){
      Database.insert(newEmi);
    }

    for(case c :caseList){
      c.Subject = 'For Testing';
    }
```

```
        Database.update(caseList);
        Integer newcase = [Select count() from case where ParentId = :caseList[0].id];
        System.assertEquals(1, newcase);

    }

    @istest
    public static void positive(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
        insert pt2;

        Case cParent = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c =
Date.today(),
                                ProductId = pt2.id);
        insert cParent;
        Case cChild = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c =
Date.today(),
                               ProductId = pt2.id,parentID = cParent.ParentId);
        insert cChild;

        cParent.subject = 'child refrecer record';
        update cParent;

        Integer newcase = [Select count() from case where ParentId = :cParent.id];
        System.assertEquals(1, newcase);

    }
    @istest public static void negetive(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
        insert pt2;

        Case c = new Case(Type = 'Repair',status = 'New',Date_Reported__c = Date.today(),
                          ProductId = pt2.id);
        insert c;

        c.Status = 'Working';
        update c;
```

```
        Integer newcase = [Select count() from case where ParentId = :c.id];
        System.assertEquals(0, newcase);
    }



}
```

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders() {
        List<case> newCaseList = new List<case>();
        Integer avgAmount=10000;

        List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
        List<case> caseList = [SELECT id,Vehicle__c,Subject,ProductID,Product__c, (SELECT
id from Equipment_Maintenance_Items__r) from case where status='closed' and Type
IN ('Repair', 'Routine Maintenance') and ID IN :Trigger.new LIMIT 200];
        Map<id,Equipment_Maintenance_Item__c> equip = new
map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c,
Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle__c from
Equipment_Maintenance_Item__c ]);
        for(case c: caseList){
            case newCase = new Case();
            newCase.Type = 'Routine Maintenance';
            newCase.Status = 'New';
            newCase.Vehicle__c = c.Vehicle__c;
            newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request' :
c.Subject;
            newCase.Date_Reported__c = Date.today();
```

```
        newCase.ProductId = c.ProductId;
        newCase.Product__c = c.Product__c;
        newCase.parentID = c.Id;


        for(Equipment_Maintenance_Item__c emi : c.Equipment_Maintenance_Items__r ){
          avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment__r.Maintenance_Cyc
le__c));
          newEMI.add(new Equipment_Maintenance_Item__c(
            Equipment__c = equip.get(emi.id).Equipment__c,
            Maintenance_Request__c = c.id,
            Quantity__c = equip.get(emi.id).Quantity__c));
        }
        Date dueDate = date.TODAY().adddays(avgAmount);
        newCase.Date_Due__c =dueDate;
        newCaseList.add(newCase);

      }
      if(newCaseList.size()>0){
        Database.insert(newCaseList);
      }

      for(Case c2: newCaseList){
        for(Equipment_Maintenance_Item__c emi2 : newEmi){
          if(c2.parentID == emi2.Maintenance_Request__c){
            emi2.Maintenance_Request__c = c2.id;
          }
        }
      }

      if(newEmi.size()>0){
        Database.insert(newEmi);
      }
    }
}
```

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after update) {
   //ToDo: Call MaintenanceRequestHelper.updateWorkOrders
   if(trigger.isAfter){
      MaintenanceRequestHelper.updateWorkOrders();
   }
}
```

5.Test Callout Logic

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();
  }

}
```

WarehouseCalloutServiceMock

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

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":true,"quantity":5,"
name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"220000"}]');
        response.setStatusCode(200);
        return response;
    }

}
```

6.Test Scheduling Logic

WarehouseSyncSchedule

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

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 ');
    }
}

////////////////////////////    $THE END$  ////////////////////////////////////////
```