APEX TRIGGERS

Get Started with Apex Triggers

“AccountAddressTrigger.apxt “

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account a: Trigger.New)
    {
        if(a.Match_Billing_Address__c == True)
        {
            a.ShippingPostalCode=a.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

“ClosedOpportunityTrigger.apxt”

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


    List<Task> taskList = new List<Task>();
    for(Opportunity o : Trigger.New)
    {
        if(o.StageName == 'Closed Won')
        {
            taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = o.Id));
        }
    }
    if(taskList.size()>0)
    {
        insert taskList;
    }
    return;
}
```

APEX TESTING

Get Started with Apex Unit Tests

“VerifyDate.apxc”

```
public class VerifyDate {


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


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

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

        //method to return the end of the month of a given date
        @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 public class
TestVerifyDate {
    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('01/05/2022'));
        System.assertEquals(date.parse('01/05/2022'), D);
    }
    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('05/05/2022'));
        System.assertEquals(date.parse('01/31/2022'), D);
    }
```

```
    @isTest static void Test_DateWithin30Days_case1(){

        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('12/30/2021'));

        System.assertEquals(false, flag);

    }

    @isTest static void Test_DateWithin30Days_case2(){

        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('02/02/2022'));

        System.assertEquals(false, flag);

    }

    @isTest static void Test_DateWithin30Days_case3(){

        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('01/15/2022'));

        System.assertEquals(true, flag);

    }

    @isTest static void Test_SetEndOfMonthDate(){

        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));

    }

}
```

Test Apex Triggers

“RestrictContactByName.apxt”

```
public class RandomContactFactory{

public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){

   List<Contact> cnts = new List<Contact>();

 for(Integer i=0;i<numcnt;i++)

       {

           Contact cnt = new Contact(FirstName = 'Test'+i, LastName = lastname);

           cnts.add(cnt);

        }

        return cnts;

    }

}
```

ASYNCHRONOUS APEX

Use Future Methods

“AccountProcessorTest”

```
public class AccountProcessor {

  @future public static void countContacts(List<Id>

  accountIds) {

     List<Account> accountsToUpdate = new List<Account>();

     List<Account> accounts = [Select Id, Name,(Select Id from Contacts) from Account

                                                    Where Id IN :accountIds];

     // process account records to do awesome stuff

     For(Account acc:accounts){

        List<Contact> contactList = acc.Contacts;

        acc.Number_of_Contacts__c = contactList.size();

        accountsToUpdate.add(acc);

     }

     update accountsToUpdate;

  }

}
```

“AccountProcessorTest.apxc”

```
@isTest public class
AccountProcessorTest {
    @isTest private static void
    testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;
        Contact newContact1 = new Contact(FirstName='John',
                                          LastName='Doe',
                                          AccountId=newAccount.Id);
        insert newContact1;
        Contact newContact2 = new Contact(FirstName='Jane',
                                          LastName='Doe',
                                          AccountId=newAccount.Id);
        insert newContact2;
        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
                    Test.stopTest();
    }
}
```

Use Batch Apex

“LeadProcessor.apxc”

```
global class LeadProcessor implements Database.Batchable<SObject> {
   global Database.QueryLocator start(Database.BatchableContext bc) {
   return Database.getQueryLocator(
            'SELECT ID from Lead'
        );
   }
   global void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {
        lead.LeadSource = 'Dreamforce';
        leads.add(lead);
        }
        update leads;
   }
global void finish(Database.BatchableContext bc){
   }
}
```

“LeadProcessorTest.apxc”

```
@isTest private class
LeadProcessorTest {
    @testSetup static
    void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 10 accounts for (Integer i=0;i<200;i++) { leads.add(new
        Lead(LastName='Lead '+i,Company='Test Co'));
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

Control Processes with Queueable Apex

“AddPrimaryContact.apxc”

```
public class AddPrimaryContact implements Queueable
   { private Contact con; private String state; public
   AddPrimaryContact(Contact con, String state) {
   this.con = con; this.state = state;
   }
   public void execute(QueueableContext context) {
      List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id
from contacts)
                           from Account where BillingState = :state Limit 200];
      List<Contact> primaryContacts = new
      List<Contact>(); for(Account acc:accounts){ Contact c
      = con.clone();
         c.AccountId = acc.Id;
         primaryContacts.add(c);
      }
      if(primaryContacts.size() > 0)
      {
         insert primaryContacts;
      }
   }
}
```

“AddPrimaryContactTest.apxc”

```
@isTest public class
AddPrimaryContactTest { static
testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(integer i=0;i<50;i++)
        {
            testAccounts.add(new Account(Name='Account '+i, BillingState='CA'));
        }
        for(integer i=0;i<50;i++)
        {
            testAccounts.add(new Account(Name='Account '+i, BillingState='NY'));
        }
        insert testAccounts;
        Contact testContact = new Contact(FirstName='John', LastName='Doe');
        insert testContact;

        AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');
        Test.startTest();
        System.enqueueJob(addit);
        Test.stopTest();
        System.assertEquals(50,[select count() from Contact where accountId in (Select Id from
                                                    Account where BillingState='CA')]);
    }
}
```

Schedule Jobs Using Apex Scheduler

“DailyLeadProcessor.apxc”

```
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.apxc”

```
@isTest private class
DailyLeadProcessorTest {
    // Dummy CRON expression: midnight on March 15.
    // Because this is a test, job executes
    // immediately after Test.stopTest().
    public static String CRON_EXP = '0 0 0 15 3 ? 2023';
    static testmethod void testScheduledJob() {
        // Create some out of date Opportunity records
```

```
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();
// Schedule the test job
String jobId = System.schedule('ScheduledApexTest', CRON_EXP,
                        new DailyLeadProcessor());
Test.stopTest();
// Now that the scheduled job has executed,
// check that our tasks were created
List<Lead> checkleads = new List<Lead>();
checkleads = [SELECT Id
            FROM Lead
            WHERE LeadSource='Dreamforce'and Company='The Inc'];
System.assertEquals(200,
                checkleads.size(),
                'Lead were not created');
```

```
    }
}
```

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); string
        animalName; if (res.getStatusCode() == 200) {
            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) results.get('animal');
            animalName = string.valueOf(animal.get('name'));
        }
        return animalName;
    }
}
```

“AnimalLocatorTest.apxc”

```
@isTest private class
AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = (string) AnimalLocator.getAnimalNameById(1);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult);
    }
}
```

“AnimalLocatorMock.apxc”

```
@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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

“ParkLocator.apxc”

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

“ParkLocatorTest.apxc”

```
@isTest private class
ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        string country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<string>();
        parks.add('Yosemite'); parks.add('Yellowstone');
        parks.add('Another Park');
        // Verify that a fake result is returned
        System.assertEquals(parks, result);
    }
```

```
}
```

“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) {
       // start - specify the response you want to send
       List<String> parks = new List<string>();
              parks.add('Yosemite');
              parks.add('Yellowstone');
              parks.add('Another Park');
       ParkService.byCountryResponse response_x =
       new ParkService.byCountryResponse();
       response_x.return_x = parks;
       // end response.put('response_x',
       response_x);
   }
}
```

Apex Web Services

“AccountManager.apxc”

```
@RestResource(urlMapping='/Accounts/*/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;
    }
}
```

“AccountManagerTest.apxc”

```
@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 accountTest = new Account(
            NAme = 'Test record');
        insert accountTest;
        Contact contactTest = new Contact(
            FirstName='John',
            LastName='Doe',
            AccountId=accountTest.Id
        );
        insert contactTest;
        return accountTest.Id;
    }
}
```

VISUALFORCE BASIC

Create & Edit Visualforce pages

“DisplayImage.vfp”

```
<apex:page showHeader="false">
        <apex:image url="https://developer.salesforce.com/files/salesforce-developer-networklogo.png"/>
<apex:page standardController="Contact">
  <apex:pageBlock title="Contact Summary">
    <apex:pageBlockSection>
      First Name: {! Contact.FirstName } <br/>
      Last Name: {! Contact.LastName } <br/>
      Owner Email: {! Contact.Owner.Email } <br/>
    </apex:pageBlockSection>
</apex:page>
```

Use Simple Variables and Formulas

“DisplayUserInfo.vfp”

```
<apex:page >
  {! $User.FirstName}
</apex:page>
```

Use Standard Controllers

“ContactView.vfp”

```
    </apex:pageBlock>
</apex:page>
```

Display Records, Fields, and Tables

“OppView.vfp”

```
<apex:page standardController="Opportunity">
    <apex:outputField value="{!Opportunity.Name}"/>
    <apex:outputField value="{!Opportunity.Amount}"/>
    <apex:outputField value="{!Opportunity.CloseDate}"/>
    <apex:outputField value="{!Opportunity.Account.Name}"/>
</apex:page>
```

Input Data Using Forms

“CreateContact.vfp”

```
<apex:page standardController="Contact">
    <apex:form>
        <apex:inputField label="First Name" value="{!Contact.FirstName}"/>
        <apex:inputField label="Last Name" value="{!Contact.LastName}"/>
        <apex:inputField label="First Name" value="{!Contact.FirstName}"/>
        <apex:inputField label="Email" value="{!Contact.Email}"/>
        <apex:commandButton action="{!save}"/>
    </apex:form>
</apex:page>
```

Use Standard List Controllers

“AccountList.vfp”

```
<apex:page standardController="Account" recordSetVar="accounts">
    <apex:form>
        <apex:repeat var="a" value="{!accounts}">
            <apex:outputLink value="/{!a.id}">{!a.name}</apex:outputLink>
        </apex:repeat>
    </apex:form>
</apex:page>
```

Use Static Resources

“ShowImage.vfp”

```
<apex:page >
    <apex:image alt="cat" title="cat"
                url="{!URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg')}"/>
</apex:page>
```

Create & Use Custom Controllers

“NewCaseList.vfp”

```
<apex:page controller="NewCaseListController">
    <apex:repeat value="{!NewCases}" var="case">
        <li><apex:outputLink value="/{!case.id}">{!case.CaseNumber}</apex:outputLink></li>
    </apex:repeat>
</apex:page>
```

“NewCaseListController.apxc”

```
public class NewCaseListController {
    public List<Case> getNewCases() {
        List<Case> results = Database.query(
            'SELECT Id, CaseNumber from Case where Status = \'New\'');
        return results;
    }
}
```

Create a Visualforce Page

Create a Visualforce Page

“Hello.vfp”

```
<apex:page > Hello
</apex:page>
```

Add a Standard Controller to the Page

“ContactForm.vfp”

```
<apex:page standardController="Contact">
    <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1" />
      <title>Quick Start: Visualforce</title>
      <!-- Import the Design System style sheet -->
      <apex:slds />
    </head>
    <body>
      <apex:form>
      <apex:pageBlock title="New Contact">
        <!--Buttons -->
         <apex:pageBlockButtons>
```

```
          <apex:commandButton action="{!save}" value="Save"/>
        </apex:pageBlockButtons>
        <!--Input form -->
        <apex:pageBlockSection columns="1">
        <apex:inputField value="{!Contact.Firstname}"/>
        <apex:inputField value="{!Contact.Lastname}"/>
        <apex:inputField value="{!Contact.Email}"/>
       </apex:pageBlockSection>
     </apex:pageBlock>
     </apex:form>
   </body>
</apex:page>
```

SUPER BADGE :=> APEX SPECIALIST

CHALLENGE 2: Automate record creation

“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;
```

“MaintenanceRequest.apxt”

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

CHALLENGE 3: Synchronize Salesforce data with an external system

“WarehouseCalloutServices.apxc”

```
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();
    }
}
```

CHALLENGE 4: Schedule synchronization

“WarehouseSyncShedule.apxc”

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

CHALLENGE 5: Test automation logic

“MaintenanceRequestHelperTest.apxc”

```
@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);
    }
}
```

“MaintenanceRequest.apxt”

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

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

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

        }

}
```

CHALLENGE 6: Test callout logic

“WarehouseCalloutService.apxc”

```
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.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]);
    }
}
```

```
@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,"quantity":5,"nam

e":"Generator 1000

kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');

response.setStatusCode(200); return response;

    }

}
```

CHALLENGE 7: 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();
        //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 ');
    }
}
```