SALESFORCE DEVELOPER

CATALYST

APEX TRIGGERS &gt;
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 (after insert, after update)
{
List&lt;Task&gt; taskList = new List&lt;Task&gt;();
for(Opportunity opp : Trigger.New){
if(opp.StageName == &#39;Closed Won&#39;){
taskList.add(new Task(Subject = &#39;Follow Up Test Task&#39;, WhatId =
opp.ID));
}
}
if(taskList.size()&gt;0){
insert taskList;
}

}
```

//
Apex Testing &gt;
1.Get Started With Apex Unit Tests
VerifyDate

```
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
private static Boolean DateWithin30Days(Date date1, Date date2) {
//check for date2 being in the past
if( date2 &lt; date1) { return false; }
//check that date2 is within (&gt;=) 30 days of date1
Date date30Days = date1.addDays(30); //create a date 30 days away
from date1

if( date2 &gt;= date30Days ) { return false; }
else { return true; }
}
//method to return the end of the month of a given date
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

@isTest
public class TestVerifyDate {
@isTest static void Test_CheckDates_case1(){
Date D = VerifyDate.CheckDates(date.parse(&#39;01/01/2020&#39;),
date.parse(&#39;01/05/2020&#39;));

System.assertEquals(date.parse(&#39;01/05/2020&#39;), D);

}
@isTest static void Test_CheckDates_case2(){
Date D = VerifyDate.CheckDates(date.parse(&#39;01/01/2020&#39;),
date.parse(&#39;05/05/2020&#39;));

System.assertEquals(date.parse(&#39;01/31/2020&#39;), D);

}
@isTest static void Test_DateWithin30Days_case1(){
Boolean flag =
VerifyDate.DateWithin30Days(date.parse(&#39;01/01/2020&#39;),
date.parse(&#39;12/30/2019&#39;));

System.assertEquals(false, flag);

}
@isTest static void Test_DateWithin30Days_case2(){
Boolean flag =
VerifyDate.DateWithin30Days(date.parse(&#39;01/01/2020&#39;),
date.parse(&#39;02/02/2020&#39;));

System.assertEquals(false, flag);

}
```

```
@isTest static void Test_DateWithin30Days_case3(){
Boolean flag =
VerifyDate.DateWithin30Days(date.parse(&#39;01/01/2020&#39;),
date.parse(&#39;01/15/2020&#39;));

System.assertEquals(false, flag);

}
@isTest static void Test_SetEndOfMonthDate(){
Date returndate =
VerifyDate.SetEndOfMonthDate(date.parse(&#39;01/01/2020&#39;));
}
}
```

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 == &#39;INVALIDNAME&#39;) { //invalidname is invalid
c.AddError(&#39;The Last Name &quot;&#39;+c.LastName+&#39;&quot; is not

allowed for DML&#39;);
}}}
```


TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
@isTest static void Test_insertupdateContact(){
Contact cnt = new Contact();
cnt.LastName = &#39;INVALIDNAME&#39;;
Test.startTest();
Database.SaveResult result = Database.insert(cnt, false);
Test.stopTest();

System.assert(!result.isSuccess());
System.assert(result.getErrors().size() &gt; 0);
System.assertEquals(&#39;The Last Name &quot;INVALIDNAME&quot; is not allowed
for DML&#39;, result.getErrors()[0].getMessage());
}}
```

3.Create Test Data for Apex Tests
RandomContactFactory

```
public class RandomContactFactory {
public static List&lt;Contact&gt; generateRandomContacts(Integer numcnt,
string lastname){
List&lt;Contact&gt; contacts = new List&lt;Contact&gt;();
for(Integer i=0;i&lt;numcnt;i++){
Contact cnt = new Contact(FirstName = &#39;Test&#39;+i, LastName =
lastname);
contacts.add(cnt);
}
return contacts;
}
}
```

//
Asynchronous Apex&gt;
1.Use Future Methods

AccountProcessor

```
public class AccountProcessor {
@future
public static void countContacts(List&lt;Id&gt; accountIds){
List&lt;Account&gt; accountsToUpdate = new List&lt;Account&gt;();

List&lt;Account&gt; accounts = [Select Id, Name, (Select Id from Contacts)
from Account Where Id in :accountIds];
For(Account acc:accounts){
List&lt;Contact&gt; contactList = acc.Contacts;
acc.Number_Of_Contacts__c = contactList.size();
accountsToUpdate.add(acc);
}
update accountsToUpdate;
}
}
```


AccountProcessorTest

```
@IsTest
private class AccountProcessorTest {
@IsTest
private static void testcountContacts(){
Account newAccount = new Account(Name=&#39;Test Account&#39;);
insert newAccount;
Contact newContact1 = new
Contact(FirstName=&#39;John&#39;,LastName=&#39;Doe&#39;,AccountId =
newAccount.Id);
insert newContact1;
Contact newContact2 = new
Contact(FirstName=&#39;Jane&#39;,LastName=&#39;Doe&#39;,AccountId =
newAccount.Id);
insert newContact2;
List&lt;Id&gt; accountIds = new List&lt;Id&gt;();
accountIds.add(newAccount.Id);
Test.startTest();
AccountProcessor.countContacts(accountIds);
Test.stopTest();
}
}
```

2.Use Batch Apex

LeadProcessor

```
global class LeadProcessor implements Database.Batchable&lt;sObject&gt; {
```

```
global Integer count = 0;
global Database.QueryLocator start(Database.BatchableContext bc){
return Database.getQueryLocator(&#39;SELECT ID, LeadSource FROM
Lead&#39;);
}
global void execute (Database.BatchableContext bc, List&lt;Lead&gt; L_list){
List&lt;lead&gt; L_list_new = new List&lt;lead&gt;();
for(lead L:L_list){
L.leadsource = &#39;Dreamforce&#39;;
L_list_new.add(L);
count +=1;
}
update L_list_new;
}
global void finish(Database.BatchableContext bc){
system.debug(&#39;count = &#39; + count);
}
}
```


LeadProcessorTest

```
@isTest
public class LeadProcessorTest {
@isTest
public static void testit(){
List&lt;lead&gt; L_list = new List&lt;lead&gt;();
for(Integer i=0; i&lt;200; i++){

Lead L = new lead();
L.LastName = &#39;name&#39; +i;
L.Company =&#39;Company&#39;;
L.Status = &#39;Random Status&#39;;
L_list.add(L);
}
insert L_list;
Test.startTest();
LeadProcessor lp = new LeadProcessor();
Id batchId = Database.executeBatch(lp);
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&lt;Account&gt; accounts = [Select Id, Name, (Select FirstName,
LastName, Id from contacts) from Account where BillingState = :state Limit
200];
List&lt;Contact&gt; primaryContacts= new List&lt;Contact&gt;();
for(Account acc:accounts){
Contact c = con.clone();
c.AccountId = acc.Id;
primaryContacts.add(c);

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

AddPrimaryContactTest

```
@isTest
```

```
public class AddPrimaryContactTest {
static testmethod void testQueueable(){
List&lt;Account&gt; testAccounts = new List&lt;Account&gt;();
for(Integer i=0;i&lt;500;i++){
testAccounts.add(new Account(Name =
&#39;Account&#39;+i,Billingstate=&#39;CA&#39;));
}
for(Integer j=0;j&lt;50;j++){
testAccounts.add(new Account(Name=&#39;Account
&#39;+j,BillingState=&#39;NY&#39;));
}
insert testAccounts;
Contact testContact = new Contact(FirstName =&#39;John&#39;,LastName
=&#39;Doe&#39;);
insert testContact;
AddPrimaryContact addit = new addPrimaryContact(testContact,&#39;CA&#39;);
Test.startTest();
system.enqueueJob(addit);
Test.stopTest();
System.assertEquals(50,[Select count() from Contact where accountId
in (Select Id from Account where BillingState=&#39;CA&#39;)]);
}
```

4.Schedule Jods Using The Apex Scheduler

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
global void execute(SchedulableContext ctx){
List&lt;lead&gt; leadstoupdate = new List&lt;lead&gt;();
List&lt;Lead&gt; leads = [Select id from Lead Where LeadSource = NULL
Limit 200];
for(Lead l:leads){
l.LeadSource = &#39;Dreamforce&#39;;
leadstoupdate.add(l);
}
update leadstoupdate;
}
}
----------------------------------------------------------------------------------------------
--------------------
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
public static String CRON_EXP = &#39;0 0 0 15 7 ? 2022&#39;;
static testmethod void testScheduledJob(){
List&lt;Lead&gt; leads = new List&lt;lead&gt;();
for (Integer i=0; i&lt;200; i++){
Lead l = new Lead(
FirstName = &#39;First &#39;+i,
LastName = &#39;LastName&#39;,
Company = &#39;The Inc&#39;
);
leads.add(l);
}
insert leads;
Test.startTest();

String jobId = System.schedule(&#39;ScheduledApexTest&#39;, CRON_EXP,
new DailyLeadProcessor());
Test.stopTest();
List&lt;Lead&gt; checkleads = new List&lt;Lead&gt;();
checkleads = [Select Id From Lead Where LeadSource = &#39;Dreamforce&#39;
and Company = &#39;The Inc&#39;];
System.assertEquals(200, checkleads.size(),&#39;Leads were not
created&#39;);
}
}
////////////////////////////////////////////////////////////////////////////////////////////////
```

Apex Integration Services&gt;
1. Apex Rest Callouts

AnimalLocator

```
public class AnimalLocator {
public static String getAnimalNameById(Integer animalId) {
String animalName;
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(&#39;https://th-apex-http-
callout.herokuapp.com/animals/&#39;+animalId);
request.setMethod(&#39;GET&#39;);
HttpResponse response = http.send(request);
if (response.getStatusCode() == 200){
Map&lt;String, Object&gt; r = (Map&lt;String, Object&gt;)
JSON.deserializeUntyped(response.getBody());
Map&lt;String, Object&gt; animal= (Map&lt;String, Object&gt;)r.get(&#39;animal&#39;);

animalName = string.valueOf(animal.get(&#39;name&#39;));

}
return animalName;
}

}
```


AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {
@isTest static void getAnimalNameByIdTest() {
Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
string response = AnimalLocator.getAnimalNameById(1);
System.assertEquals(&#39;chicken&#39;, response);
}
}
```

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(&#39;Content-Type&#39;, &#39;application/json&#39;);
response.setBody(&#39;{&quot;animal&quot;:{&quot;id&quot;:1,&quot;name&quot;:&quo
t;chicken&quot;,&quot;eats&quot;:&quot;chicken
food&quot;,&quot;says&quot;:&quot;cluck cluck&quot;}}&#39;);
response.setStatusCode(200);
return response;
}
}
```

2. Apex SOAP Callouts

ParkLocator

```
public class ParkLocator {

public static List&lt;String&gt; country(String country) {
ParkService.ParksImplPort parkservice =
new parkService.ParksImplPort();
return parkservice.byCountry(country);
}
}
```

ParkLocatorTest

```
@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 = &#39;United States&#39;;
List&lt;String&gt; result = ParkLocator.country(country);
List&lt;String&gt; parks = new List&lt;String&gt;();

parks.add(&#39;Yosemite&#39;);
parks.add(&#39;Yellowstone&#39;);
parks.add(&#39;Another Park&#39;);
// Verify that a fake result is returned
System.assertEquals(parks, result);
}
}
```

ParkServiceMock

```
@isTest
global class ParkServiceMock implements WebServiceMock {
global void doInvoke(
Object stub,
Object request,
Map&lt;String, Object&gt; response,
String endpoint,
String soapAction,

String requestName,
String responseNS,
String responseName,
String responseType) {
// start - specify the response you want to send
List&lt;String&gt; parks = new List&lt;string&gt;();
parks.add(&#39;Yosemite&#39;);
parks.add(&#39;Yellowstone&#39;);
parks.add(&#39;Another Park&#39;);
ParkService.byCountryResponse response_x =
new ParkService.byCountryResponse();
response_x.return_x = parks;
// end
response.put(&#39;response_x&#39;, response_x);
```

```
}
}
-----------------------------------------------------------------------------------------------
--------------------

ParkService

//Generated by wsdl2apex
public class ParkService {
public class byCountryResponse {
public String[] return_x;
private String[] return_x_type_info = new
String[]{&#39;return&#39;,&#39;http://parks.services/&#39;,null,&#39;0&#39;,&#39;-
1&#39;,&#39;false&#39;};
private String[] apex_schema_type_info = new
String[]{&#39;http://parks.services/&#39;,&#39;false&#39;,&#39;false&#39;};
private String[] field_order_type_info = new String[]{&#39;return_x&#39;};
}
public class byCountry {
public String arg0;
private String[] arg0_type_info = new
String[]{&#39;arg0&#39;,&#39;http://parks.services/&#39;,null,&#39;0&#39;,&#39;1&#39
;,&#39;false&#39;};
private String[] apex_schema_type_info = new
String[]{&#39;http://parks.services/&#39;,&#39;false&#39;,&#39;false&#39;};
private String[] field_order_type_info = new String[]{&#39;arg0&#39;};
}
public class ParksImplPort {

public String endpoint_x = &#39;https://th-apex-soap-
service.herokuapp.com/service/parks&#39;;
public Map&lt;String,String&gt; inputHttpHeaders_x;
public Map&lt;String,String&gt; 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[]{&#39;http://parks.services/&#39;, &#39;ParkService&#39;};
public String[] byCountry(String arg0) {
ParkService.byCountry request_x = new ParkService.byCountry();
request_x.arg0 = arg0;
ParkService.byCountryResponse response_x;
Map&lt;String, ParkService.byCountryResponse&gt; response_map_x =
new Map&lt;String, ParkService.byCountryResponse&gt;();
response_map_x.put(&#39;response_x&#39;, response_x);
WebServiceCallout.invoke(
this,
request_x,
response_map_x,
new String[]{endpoint_x,
&#39;&#39;,
&#39;http://parks.services/&#39;,
&#39;byCountry&#39;,
&#39;http://parks.services/&#39;,
&#39;byCountryResponse&#39;,
&#39;ParkService.byCountryResponse&#39;}
);
response_x = response_map_x.get(&#39;response_x&#39;);
return response_x.return_x;
}
}
}
```

3. Apex Web Servcies

AccountManager

```
@RestResource(urlMapping = &#39;/Accounts/*/contacts&#39;)
global with sharing class AccountManager {
@HttpGet
global static Account getAccount(){
RestRequest request = RestContext.request;
String accountId =
request.requestURI.substringBetween(&#39;Accounts/&#39;,&#39;/contacts&#39;);
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 testGetContactsByAccountId(){
Id recordId = createTestRecord();
RestRequest request = new RestRequest();
request.requestUri =
&#39;https://yourInstance.my.salesforce.com/services/apexrest/Accounts/&#39;

+ recordId+&#39;/contacts&#39;;
request.httpMethod = &#39;GET&#39;;
RestContext.request = request;
Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount != null);
System.assertEquals(&#39;Test record&#39;, thisAccount.Name);
}
static Id createTestRecord(){
Account accountTest = new Account(

Name =&#39;Test record&#39;);

insert accountTest;
Contact contactTest = new Contact(

FirstName=&#39;John&#39;,
LastName=&#39;Doe&#39;,

AccountId=accountTest.Id
);
insert contactTest;
return accountTest.Id;
}
}
////////////////////////////////////////////////////////////////////////////////////////////
```

Apex Specialist SuperBadge&gt;

1.Automates Record Creation
MaintenanceRequest

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

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List&lt;Case&gt; updWorkOrders,
Map&lt;Id,Case&gt; nonUpdCaseMap) {
Set&lt;Id&gt; validIds = new Set&lt;Id&gt;();
For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != &#39;Closed&#39; &amp;&amp; c.Status ==
&#39;Closed&#39;){
if (c.Type == &#39;Repair&#39; || c.Type == &#39;Routine Maintenance&#39;){
validIds.add(c.Id);
}
}

}
//When an existing maintenance request of type Repair or Routine
Maintenance is closed,
//create a new maintenance request for a future routine checkup.
```

```
if (!validIds.isEmpty()){
Map&lt;Id,Case&gt; closedCases = new Map&lt;Id,Case&gt;([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&lt;Id,Decimal&gt; maintenanceCycles = new Map&lt;ID,Decimal&gt;();
//calculate the maintenance request due dates by using the
maintenance cycle defined on the related equipment records.
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle
FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c];
for (AggregateResult ar : results){
maintenanceCycles.put((Id) ar.get(&#39;Maintenance_Request__c&#39;),
(Decimal) ar.get(&#39;cycle&#39;));
}
List&lt;Case&gt; newCases = new List&lt;Case&gt;();
for(Case cc : closedCases.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = &#39;New&#39;,
Subject = &#39;Routine Maintenance&#39;,
Type = &#39;Routine Maintenance&#39;,
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = &#39;Web&#39;,
Date_Reported__c = Date.Today()
);

//If multiple pieces of equipment are used in the maintenance
request,
//define the due date by applying the shortest maintenance cycle
to today’s date.
//If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
//} else {
// nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
//}
newCases.add(nc);
}
insert newCases;
List&lt;Equipment_Maintenance_Item__c&gt; clonedList = new
List&lt;Equipment_Maintenance_Item__c&gt;();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c item =
clonedListItem.clone();
item.Maintenance_Request__c = nc.Id;
clonedList.add(item);
}
}
insert clonedList;
}
}
}
```

2.Synchronize Salesforce data with an external

system

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable
{

private static final String WAREHOUSE_URL = &#39;https://th-superbadge-
apex.herokuapp.com/equipment&#39;;
system to get a list of equipment that needs to be updated.

@future(callout=true)
```

```
public static void runWarehouseEquipmentSync(){
System.debug(&#39;go into runWarehouseEquipmentSync&#39;);
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod(&#39;GET&#39;);
HttpResponse response = http.send(request);
List&lt;Product2&gt; product2List = new List&lt;Product2&gt;();
System.debug(response.getStatusCode());
if (response.getStatusCode() == 200){
List&lt;Object&gt; jsonResponse =
(List&lt;Object&gt;)JSON.deserializeUntyped(response.getBody());
System.debug(response.getBody());

for (Object jR : jsonResponse){
Map&lt;String,Object&gt; mapJson = (Map&lt;String,Object&gt;)jR;
Product2 product2 = new Product2();
product2.Replacement_Part__c = (Boolean)
mapJson.get(&#39;replacement&#39;);
product2.Cost__c = (Integer) mapJson.get(&#39;cost&#39;);
product2.Current_Inventory__c = (Double)
mapJson.get(&#39;quantity&#39;);
product2.Lifespan_Months__c = (Integer)
mapJson.get(&#39;lifespan&#39;);

product2.Maintenance_Cycle__c = (Integer)
mapJson.get(&#39;maintenanceperiod&#39;);
product2.Warehouse_SKU__c = (String) mapJson.get(&#39;sku&#39;);
product2.Name = (String) mapJson.get(&#39;name&#39;);
product2.ProductCode = (String) mapJson.get(&#39;_id&#39;);
product2List.add(product2);
}
if (product2List.size() &gt; 0){
upsert product2List;
System.debug(&#39;Your equipment was synced with the warehouse
one&#39;);
}
}
}
public static void execute (QueueableContext context){
System.debug(&#39;start runWarehouseEquipmentSync&#39;);
runWarehouseEquipmentSync();
System.debug(&#39;end runWarehouseEquipmentSync&#39;);
}
}
```

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 {
// createVehicle
private static Vehicle__c createVehicle(){
Vehicle__c vehicle = new Vehicle__C(name = &#39;Testing Vehicle&#39;);
return vehicle;
}
// createEquipment
private static Product2 createEquipment(){
product2 equipment = new product2(name = &#39;Testing equipment&#39;,
lifespan_months__c = 10,
maintenance_cycle__c = 10,
replacement_part__c = true);
return equipment;
```

```
}
// createMaintenanceRequest
private static Case createMaintenanceRequest(id vehicleId, id
equipmentId){
case cse = new case(Type=&#39;Repair&#39;,
Status=&#39;New&#39;,
Origin=&#39;Web&#39;,
Subject=&#39;Testing subject&#39;,
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cse;
}
// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return equipmentMaintenanceItem;

}
@isTest
private static void testPositive(){
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
Product2 equipment = createEquipment();
insert equipment;
id equipmentId = equipment.Id;
case createdCase =
createMaintenanceRequest(vehicleId,equipmentId);
insert createdCase;
Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
insert equipmentMaintenanceItem;
test.startTest();
```

```
createdCase.status = &#39;Closed&#39;;
update createdCase;
test.stopTest();
Case newCase = [Select id,
subject,
type,
Equipment__c,
Date_Reported__c,
Vehicle__c,
Date_Due__c
from case
where status =&#39;New&#39;];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c
=:newCase.Id];

list&lt;case&gt; allCase = [select id from case];
system.assert(allCase.size() == 2);
system.assert(newCase != null);
system.assert(newCase.Subject != null);
system.assertEquals(newCase.Type, &#39;Routine Maintenance&#39;);
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}
@isTest
private static void testNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
product2 equipment = createEquipment();
insert equipment;
id equipmentId = equipment.Id;
case createdCase =
createMaintenanceRequest(vehicleId,equipmentId);
insert createdCase;
```

```
Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
insert workP;
test.startTest();
createdCase.Status = &#39;Working&#39;;
update createdCase;
test.stopTest();
list&lt;case&gt; allCase = [select id from case];
Equipment_Maintenance_Item__c equipmentMaintenanceItem =
[select id
from Equipment_Maintenance_Item__c

where Maintenance_Request__c =
:createdCase.Id];
system.assert(equipmentMaintenanceItem != null);
system.assert(allCase.size() == 1);
}
@isTest
private static void testBulk(){
list&lt;Vehicle__C&gt; vehicleList = new list&lt;Vehicle__C&gt;();
list&lt;Product2&gt; equipmentList = new list&lt;Product2&gt;();
list&lt;Equipment_Maintenance_Item__c&gt;
equipmentMaintenanceItemList = new
list&lt;Equipment_Maintenance_Item__c&gt;();
list&lt;case&gt; caseList = new list&lt;case&gt;();
list&lt;id&gt; oldCaseIds = new list&lt;id&gt;();
for(integer i = 0; i &lt; 300; i++){
vehicleList.add(createVehicle());
equipmentList.add(createEquipment());
}
insert vehicleList;
insert equipmentList;
for(integer i = 0; i &lt; 300; i++){
caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
}
insert caseList;
```

```
for(integer i = 0; i &lt; 300; i++){
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equ
ipmentList.get(i).id, caseList.get(i).id));
}
insert equipmentMaintenanceItemList;
test.startTest();
for(case cs : caseList){
cs.Status = &#39;Closed&#39;;

oldCaseIds.add(cs.Id);
}
update caseList;
test.stopTest();
list&lt;case&gt; newCase = [select id
from case
where status =&#39;New&#39;];

list&lt;Equipment_Maintenance_Item__c&gt; workParts = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in:
oldCaseIds];
system.assert(newCase.size() == 300);
list&lt;case&gt; allCase = [select id from case];
system.assert(allCase.size() == 600);
}
}
```


```
MaintenanceRequestHelper
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List&lt;Case&gt; updWorkOrders,
Map&lt;Id,Case&gt; nonUpdCaseMap) {
Set&lt;Id&gt; validIds = new Set&lt;Id&gt;();
For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != &#39;Closed&#39; &amp;&amp; c.Status ==
&#39;Closed&#39;){
```

```
if (c.Type == &#39;Repair&#39; || c.Type == &#39;Routine Maintenance&#39;){
validIds.add(c.Id);
}
}
}

//When an existing maintenance request of type Repair or Routine
Maintenance is closed,
//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
Map&lt;Id,Case&gt; closedCases = new Map&lt;Id,Case&gt;([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&lt;Id,Decimal&gt; maintenanceCycles = new Map&lt;ID,Decimal&gt;();
//calculate the maintenance request due dates by using the
maintenance cycle defined on the related equipment records.
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle
FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c];
for (AggregateResult ar : results){
maintenanceCycles.put((Id) ar.get(&#39;Maintenance_Request__c&#39;),
(Decimal) ar.get(&#39;cycle&#39;));
}
List&lt;Case&gt; newCases = new List&lt;Case&gt;();
for(Case cc : closedCases.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = &#39;New&#39;,
Subject = &#39;Routine Maintenance&#39;,
Type = &#39;Routine Maintenance&#39;,
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = &#39;Web&#39;,
Date_Reported__c = Date.Today()
);
//If multiple pieces of equipment are used in the maintenance
request,

//define the due date by applying the shortest maintenance cycle
to today’s date.
//If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
//} else {
// nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
//}
newCases.add(nc);
}
insert newCases;
List&lt;Equipment_Maintenance_Item__c&gt; clonedList = new
List&lt;Equipment_Maintenance_Item__c&gt;();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c item =
clonedListItem.clone();
item.Maintenance_Request__c = nc.Id;
clonedList.add(item);
}
}
insert clonedList;
}
}
}
```


MaintenanceRequest

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

5.Test Callout Logic

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable
{
private static final String WAREHOUSE_URL = &#39;https://th-superbadge-
apex.herokuapp.com/equipment&#39;;
//Write a 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(){
System.debug(&#39;go into runWarehouseEquipmentSync&#39;);
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod(&#39;GET&#39;);
HttpResponse response = http.send(request);
List&lt;Product2&gt; product2List = new List&lt;Product2&gt;();
System.debug(response.getStatusCode());
if (response.getStatusCode() == 200){
List&lt;Object&gt; jsonResponse =
(List&lt;Object&gt;)JSON.deserializeUntyped(response.getBody());
System.debug(response.getBody());
//class maps the following fields:
//warehouse SKU will be external ID for identifying which equipment
records to update within Salesforce
for (Object jR : jsonResponse){
```

```
Map&lt;String,Object&gt; mapJson = (Map&lt;String,Object&gt;)jR;
Product2 product2 = new Product2();
//replacement part (always true),

product2.Replacement_Part__c = (Boolean)
mapJson.get(&#39;replacement&#39;);
//cost
product2.Cost__c = (Integer) mapJson.get(&#39;cost&#39;);
//current inventory
product2.Current_Inventory__c = (Double)
mapJson.get(&#39;quantity&#39;);
//lifespan
product2.Lifespan_Months__c = (Integer)
mapJson.get(&#39;lifespan&#39;);
//maintenance cycle
product2.Maintenance_Cycle__c = (Integer)
mapJson.get(&#39;maintenanceperiod&#39;);
//warehouse SKU
product2.Warehouse_SKU__c = (String) mapJson.get(&#39;sku&#39;);
product2.Name = (String) mapJson.get(&#39;name&#39;);
product2.ProductCode = (String) mapJson.get(&#39;_id&#39;);
product2List.add(product2);
}
if (product2List.size() &gt; 0){
upsert product2List;
System.debug(&#39;Your equipment was synced with the warehouse
one&#39;);
}
}
}
public static void execute (QueueableContext context){
System.debug(&#39;start runWarehouseEquipmentSync&#39;);
runWarehouseEquipmentSync();
System.debug(&#39;end runWarehouseEquipmentSync&#39;);
}
}
```

WArehouseCalloutServiceTest

```
@IsTest
private class WarehouseCalloutServiceTest {

// implement your mock callout test here

@isTest

static void testWarehouseCallout() {
test.startTest();
test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
WarehouseCalloutService.execute(null);
test.stopTest();
List&lt;Product2&gt; product2List = new List&lt;Product2&gt;();
product2List = [SELECT ProductCode FROM Product2];
System.assertEquals(3, product2List.size());
System.assertEquals(&#39;55d66226726b611100aaf741&#39;,
product2List.get(0).ProductCode);
System.assertEquals(&#39;55d66226726b611100aaf742&#39;,
product2List.get(1).ProductCode);
System.assertEquals(&#39;55d66226726b611100aaf743&#39;,
product2List.get(2).ProductCode);
}
}
```


WarehouseCalloutServiceMock

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

response.setHeader(&#39;Content-Type&#39;, &#39;application/json&#39;);
response.setBody(&#39;[{&quot;_id&quot;:&quot;55d66226726b611100aaf741&quot;,&q
uot;replacement&quot;:fal
se,&quot;quantity&quot;:5,&quot;name&quot;:&quot;Generator 1000
kW&quot;,&quot;maintenanceperiod&quot;:365,&quot;lifespan&quot;:120,&quot;cost&qu
ot;:5000,&quot;sku&quot;:&quot;100003&quot;},{&quot;
_id&quot;:&quot;55d66226726b611100aaf742&quot;,&quot;replacement&quot;:true,&qu
ot;quantity&quot;:183,&quot;nam
e&quot;:&quot;Cooling
Fan&quot;,&quot;maintenanceperiod&quot;:0,&quot;lifespan&quot;:0,&quot;cost&quot;:3
00,&quot;sku&quot;:&quot;100004&quot;},{&quot;_id&quot;:&quot;

55d66226726b611100aaf743&quot;,&quot;replacement&quot;:true,&quot;quantity&quot;
:143,&quot;name&quot;:&quot;F
use
20A&quot;,&quot;maintenanceperiod&quot;:0,&quot;lifespan&quot;:0,&quot;cost&quot;:2
2,&quot;sku&quot;:&quot;100005&quot;}]&#39;);
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 with sharing class WarehouseSyncScheduleTest {
// implement scheduled code here
//
@isTest static void test() {
String scheduleTime = &#39;00 00 00 * * ? *&#39;;
Test.startTest();
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
String jobId = System.schedule(&#39;Warehouse Time to Schedule to test&#39;,
scheduleTime, new WarehouseSyncSchedule());
CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =:
jobId];
System.assertEquals(&#39;WAITING&#39;, String.valueOf(c.State), &#39;JobId does
not match&#39;);
Test.stopTest();

}
}
//////////////////////////////////////THE END/////////////////////////////////////////
```