Apex Trigger
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
1.Get Started with Apex Unit Tests
VerifyDate Class

```
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 Class

```
@IsTest
public class TestVerifyDate {
@isTest static void date2within30daydate1() {
Date returnDate1 = VerifyDate.CheckDates(date.valueOf(&#39;2022-06-
14&#39;),date.valueOf(&#39;2022-06-24&#39;));
System.assertEquals(date.valueOf(&#39;2022-06-24&#39;), returnDate1);
}
@isTest static void date2NOTwithin30daydate1() {
Date returnDate2 = VerifyDate.CheckDates(date.valueOf(&#39;2022-06-
14&#39;),date.valueOf(&#39;2022-07-24&#39;));
System.assertEquals(date.valueOf(&#39;2022-06-29&#39;), returnDate2);
}
}
```

2.Test Apex Triggers

RestrictContactByName Trigger

```
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 Test Class
@IsTest
public class TestRestrictContactByName {
@IsTest static void createBadContact(){
Contact c = new Contact(FirstName = &#39;John&#39;, LastName =
&#39;INVALIDNAME&#39;);
Test.startTest();
Database.SaveResult result = Database.insert(c, false);
Test.stopTest();
System.assert(!result.isSuccess());
}
}
```

3.Create Test Data for Apex Tests

RandomContactFactory Class

```
public class RandomContactFactory {
public static List&lt;Contact&gt; generateRandomContacts(Integer num,String
lastname){
List&lt;Contact&gt; contactList = new List&lt;Contact&gt;();
for(Integer i = 1;i&lt;=num;i++){
Contact ct = new Contact(FirstName = &#39;Test&#39;+i,LastName =lastName);
contactList.add(ct);
}
return contactList;
}
}
```

Asynchronous Apex

1.Use Future Methods

AccountProcessor Class

```
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 Class

```
@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 Class
public without sharing class LeadProcessor implements
Database.Batchable<sobject>, Database.Stateful {
public Integer recordCount =0;
public Database.QueryLocator start(Database.BatchableContext dbc) {
return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
}
public void execute(database.BatchableContext dbc, List<Lead> leads) {
for(Lead l : leads) {
l.LeadSource = 'Dreamforce';
}
update leads;
recordCount = recordCount + leads.size();
}
public void finish (Database.BatchableContext dbc) {
System.debug('Total records processed' + recordCount);
}
}
LeadProcessorTest Class
@IsTest
private class LeadProcessorTest {

@isTest
private static void testBatchClass() {
//Load test Data
List<Lead> leads = new List<Lead>();
for (Integer i =0; i<200; i++) {
leads.add(new Lead(LastName='Connock', Company =
'Salesforce'));
}
insert leads;
//Perform the Test
Test.startTest();
LeadProcessor lp = new LeadProcessor();

```
Id batchId = Database.executeBatch(lp,200);
Test.stopTest();
//Check the Result
List&lt;Lead&gt; updatedLeads = [SELECT Id FROM Lead WHERE Leadsource =
&#39;Dreamforce&#39;];
System.assertEquals(200, updatedLeads.size(), &#39;ERROR: At Least 1 lead record not
updated correctly&#39;);
}
}
```

3.Control Processes with Queueable Apex

AddPrimaryContact Class

```
public class AddPrimaryContact implements Queueable{
private Contact c;
private String state;
public AddPrimaryContact(Contact c,String state){
this.c =c;
this.state = state;
}
public void execute(QueueableContext context){
List&lt;Account&gt; ListAccount = [SELECT Id, Name, (SELECT Id, FirstName, LastName
FROM Contacts) FROM Account
WHERE BillingState =:
state LIMIT 200];
List&lt;Contact&gt; lstContact = new List&lt;Contact&gt;();
for (Account acc:ListAccount){
Contact cont = c.clone(false, false, false, false);
cont.Accountid = acc.id;
lstContact.add(cont);
}
if(lstcontact.size()&gt;0){
insert lstcontact;
}
}
}
```

AddPrimaryContactTest Class

```
@isTest
public class AddPrimaryContactTest {
```

@isTest static void TestList(){
List<Account> Teste = new List<Account>();
for(Integer i=0;i<50;i++){
teste.add(new Account(BillingState ='CA', name = 'Test' +i));
}
for(Integer j=0;j<50;j++){
Teste.add(new Account(BillingState ='NY', name= 'Test'+ j));
}
insert Teste;

Contact co = new Contact();
co.FirstName ='demo';
co.LastName ='demo';
insert co;
String state = 'CA';
AddPrimaryContact apc = new AddPrimaryContact(co, state);
Test.startTest();
System.enqueueJob(apc);
Test.stopTest();
}
}

4. Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor Class

public class DailyLeadProcessor implements Schedulable{
Public void execute(SchedulableContext SC){
List<Lead> LeadObj= [SELECT Id from Lead where LeadSource=null limit 200];
for(Lead l:LeadObj){
l.LeadSource='Dreamforce';
update l;
}
}
}

DailyLeadProcessorTest Class

@isTest
public class DailyLeadProcessorTest {
static testMethod void testDailyLeadProcessor(){
String CRON_EXP ='0 0 1 * * ?';

List<Lead> lList = new List<Lead>();
for (Integer i=0;i<200;i++){
lList.add(new Lead(LastName = 'Dreamforce'+i, Company ='Test1 Inc.',
status='Open - Not Connected'));
}
insert lList;
Test.startTest();
string jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new DailyLeadProcessor());
}
}

AnimalLocator Class
public class AnimalLocator {
public static String getAnimalNameById(Integer animalId) {
String animalName;
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+animalId);
request.setMethod('GET');
HttpResponse response = http.send(request);
// If the request is successful, parse the JSON response.
if(response.getStatusCode() == 200) {
Map<String, Object> r = (Map<String, Object>) JSON.deserializeUntyped(response.getbody());
Map<String, Object> animal = (Map<String, Object>)r.get('animal');
animalName = string.valueOf(animal.get('name'));
}
return animalName;
}
}
AnimalLocatorMock Mock Class
@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":0,"name":&quo
t;","eats":"","says":""}}');
response.setStatusCode(200);
return response;
}
}
AnimalLocatorTest Class
@isTest
private class AnimalLocatorTest{
@isTest static void getAnimalNameByIdTest() {
// Set mock callout class
Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
// This causes a fake response to be sent
// from the class that implements HttpCalloutMock.
String response = AnimalLocator.getAnimalNameById(1);
// Verify that the response received contains fake values
System.assertEquals('chicken', response);
}
}
2.Apex SOAP Callouts
ParkService Class
//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[]{&#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;
}
}
}
```

ParkLocator Class

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

ParkServiceMock Class

```
@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;Gir National Park&#39;);
parks.add(&#39;Jim Corbett National Park&#39;);
parks.add(&#39;Ranthambore National Park&#39;);
ParkService.byCountryResponse response_x =
```

```
new ParkService.byCountryResponse();
response_x.return_x = parks;
// end
response.put(&#39;response_x&#39;, response_x);
}
}
ParkLocatorTest Class
@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;India&#39;;
List&lt;String&gt; result = ParkLocator.country(country);
List&lt;String&gt; parks = new List&lt;String&gt;();
parks.add(&#39;Gir National Park&#39;);
parks.add(&#39;Jim Corbett National Park&#39;);
parks.add(&#39;Ranthambore National Park&#39;);
// Verify that a fake result is returned
System.assertEquals(parks, result);
}
}
```

3.Apex Web Services

AccountManager Class

```
@RestResource(urlMapping=&#39;/Accounts/*/contacts&#39;)
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(&#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 Class
@IsTest
private class AccountManagerTest {
@isTest static void testGetContactsByAccountId() {
Id recordId = createTestRecord();
// Set up a test request
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;
// Call the method to test
Account thisAccount = AccountManager.getAccount();
// Verify results
System.assert(thisAccount != null);
System.assertEquals(&#39;Test record&#39;, thisAccount.Name);
}
// Helper method
static Id createTestRecord() {
// Create test record
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

STEP 2: Automate record creation
MaintenanceRequest Trigger

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

}
}
```

MaintenanceRequestHelper Class

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List&amp;lt;Case&amp;gt; updWorkOrders,
Map&amp;lt;Id,Case&amp;gt; nonUpdCaseMap) {
Set&amp;lt;Id&amp;gt; validIds = new Set&amp;lt;Id&amp;gt;();
For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != &amp;#39;Closed&amp;#39; &amp;amp;&amp;amp; c.Status ==
&amp;#39;Closed&amp;#39;){
if (c.Type == &amp;#39;Repair&amp;#39; || c.Type == &amp;#39;Routine Maintenance&amp;#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&amp;lt;Id,Case&amp;gt; closedCases = new Map&amp;lt;Id,Case&amp;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&amp;lt;Id,Decimal&amp;gt; maintenanceCycles = new Map&amp;lt;ID,Decimal&amp;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(&amp;#39;Maintenance_Request__c&amp;#39;), (Decimal)
ar.get(&amp;#39;cycle&amp;#39;));
}
List&amp;lt;Case&amp;gt; newCases = new List&amp;lt;Case&amp;gt;();
for(Case cc : closedCases.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = &amp;#39;New&amp;#39;,
Subject = &amp;#39;Routine Maintenance&amp;#39;,
Type = &amp;#39;Routine Maintenance&amp;#39;,
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = &amp;#39;Web&amp;#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&amp;lt;Equipment_Maintenance_Item__c&amp;gt; clonedList = new
List&amp;lt;Equipment_Maintenance_Item__c&amp;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;
}
}
}
```

STEP 3: Synchronize Salesforce data with an external system

WarehouseCalloutService Class

```
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<String,Object> mapJson = (Map<String,Object>)jR;
Product2 product2 = new Product2();
//replacement part (always true),
product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
//cost
product2.Cost__c = (Integer) mapJson.get('cost');

//current inventory
product2.Current_Inventory__c = (Double) mapJson.get('quantity');
//lifespan
product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
//maintenance cycle
product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
//warehouse SKU
product2.Warehouse_SKU__c = (String) mapJson.get('sku');
product2.Name = (String) mapJson.get('name');
product2.ProductCode = (String) mapJson.get('_id');
product2List.add(product2);
}
if (product2List.size() > 0){
upsert product2List;
System.debug('Your equipment was synced with the warehouse one');
}
}
}
public static void execute (QueueableContext context){
System.debug('start runWarehouseEquipmentSync');
runWarehouseEquipmentSync();
System.debug('end runWarehouseEquipmentSync');
}
}
STEP 4: Schedule synchronization
WarehouseSyncSchedule Class
global with sharing class WarehouseSyncSchedule implements Schedulable {
// implement scheduled code here

```
global void execute (SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService());
}
}
```

STEP 5: Test automation logic

MaintenanceRequest Trigger

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

MaintenanceRequestHelper Class

```
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;
}
}
}
```

MaintenanceRequestHelperTest Class

```
@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(equipmentList.
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);
}
}

STEP 6: Test callout logic

WarehouseCalloutService Class

public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';
//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('go into runWarehouseEquipmentSync');
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);

request.setMethod('GET');
HttpResponse response = http.send(request);
List<Product2> product2List = new List<Product2>();
System.debug(response.getStatusCode());
if (response.getStatusCode() == 200){
List<Object> jsonResponse =
(List<Object>)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<String,Object> mapJson = (Map<String,Object>)jR;
Product2 product2 = new Product2();
//replacement part (always true),
product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
//cost

product2.Cost__c = (Integer) mapJson.get('cost');
//current inventory
product2.Current_Inventory__c = (Double) mapJson.get('quantity');
//lifespan
product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
//maintenance cycle
product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
//warehouse SKU
product2.Warehouse_SKU__c = (String) mapJson.get('sku');
product2.Name = (String) mapJson.get('name');
product2.ProductCode = (String) mapJson.get('_id');
product2List.add(product2);
}
if (product2List.size() > 0){
upsert product2List;
System.debug('Your equipment was synced with the warehouse one');
}
}
}
public static void execute (QueueableContext context){
System.debug('start runWarehouseEquipmentSync');
runWarehouseEquipmentSync();
System.debug('end runWarehouseEquipmentSync');
}
}
WarehouseCalloutServiceMock Class
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request) {
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741",&q
uot;replacement":false,"quantity":5,"name":"Generator
1000
kW","maintenanceperiod":365,"lifespan":120,"cost&qu

ot;:5000,"sku":"100003"},{"_id":"55d66226726b6
11100aaf742","replacement
":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":3
00,"sku":"100004"},{"_id":"55d66226726b611100
aaf743","replacement":true
,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":2
2,"sku":"100005"}]');
response.setStatusCode(200);

return response;
}
}

WarehouseCalloutServiceTest Class
@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<Product2> product2List = new List<Product2>();
product2List = [SELECT ProductCode FROM Product2];
System.assertEquals(3, product2List.size());
System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
}
}

STEP 7: Test scheduling logic

WarehouseCalloutServiceMock Class

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request) {
HttpResponse response = new HttpResponse();
response.setHeader(&amp;#39;Content-Type&amp;#39;, &amp;#39;application/json&amp;#39;);
response.setBody(&amp;#39;[{&amp;quot;_id&amp;quot;:&amp;quot;55d66226726b611100aaf741&amp;quot;,&amp;quot;replacement&amp;quot;:false,&amp;quot;quantity&amp;quot;:5,&amp;quot;name&amp;quot;:&amp;quot;Generator 1000
kW&amp;quot;,&amp;quot;maintenanceperiod&amp;quot;:365,&amp;quot;lifespan&amp;quot;:120,&amp;quot;cost&amp;quot;:5000,&amp;quot;sku&amp;quot;:&amp;quot;100003&amp;quot;},{&amp;quot;_id&amp;quot;:&amp;quot;55d66226726b611100aaf742&amp;quot;,&amp;quot;replacement
&amp;quot;:true,&amp;quot;quantity&amp;quot;:183,&amp;quot;name&amp;quot;:&amp;quot;Cooling Fan&amp;quot;,&amp;quot;maintenanceperiod&amp;quot;:0,&amp;quot;lifespan&amp;quot;:0,&amp;quot;cost&amp;quot;:300,&amp;quot;sku&amp;quot;:&amp;quot;100004&amp;quot;},{&amp;quot;_id&amp;quot;:&amp;quot;55d66226726b611100aaf743&amp;quot;,&amp;quot;replacement&amp;quot;:true
,&amp;quot;quantity&amp;quot;:143,&amp;quot;name&amp;quot;:&amp;quot;Fuse 20A&amp;quot;,&amp;quot;maintenanceperiod&amp;quot;:0,&amp;quot;lifespan&amp;quot;:0,&amp;quot;cost&amp;quot;:22,&amp;quot;sku&amp;quot;:&amp;quot;100005&amp;quot;}]&amp;#39;);
response.setStatusCode(200);
return response;
}
}
```

WarehouseSyncSchedule Class

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

WarehouseSyncScheduleTest Class

```
@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();
}
}
```