Module:Apex integtration services

class name:AnimalLocator

1. public class AnimalLocator{
2. public static String getAnimalNameById(Integer x){
3. Http http=new Http();
4. HttpRequest req=new HttpRequest();
5. req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' +x);
6. req.setMethod('GET');
7. Map<String, Object> animal = new Map<String,Object>();
8. HttpResponse res = http.send(req);
9. if(res.getStatusCode() == 200){
10. Map<String, Object> results= (Map<String, Object>)JSON.deserializeUntyped(res.getBody());
11. animal = (Map<String, Object>) results.get('animal');
12. }
13. return (String)animal.get('name');
14. }
15. }

class name:AnimalLocatorTest

1. @isTest
2. private class AnimalLocatorTest{
3. @isTest static void AnimalLocatorMock1(){
4. try{
5. Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
6. string result = AnimalLocator.getAnimalNameById(1);
7. String expectedResult = 'fox';
8. System.assertEquals(result,expectedResult);
9. }
10. catch(exception e){
11. System.debug('The following exception has occurred: ' + e.getMessage());
12. }
13. }
14. }

class name:AnimalLocatorMock

1. @isTest
2. global class AnimalLocatorMock implements HttpCalloutMock{
3. global HTTPResponse respond(HTTPRequest request){
4. HttpResponse response = new HttpResponse();
5. response.setHeader('Content-Type','application/json');
6. response.setBody('{"animals":["lion","fox","bear","panda","snake","raccoon"]}');
7. response.setStatusCode(200);
8. return response;
9. }
10. }

classname: ParkLocator

1. public class ParkLocator {
2. public static string [] country ( string theCountry ) {
3. ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort();
4. return parkSvc.byCountry(theCountry);
5. }
6. }

classname:ParkLocatorTest

1. @isTest
2. private class ParkLocatorTest{
3. @isTest static void testCallout (){
4. Test.setMock(WebServiceMock.class, new ParkServiceMock ());
5. String country = ' United States';
6. List<String> result = ParkLocator.country (country);
7. List<String> parks = new List<String>{'Yellowstone','Mackinac National Park','Yosemite'};
8. System.assertEquals(parks, result);
9. }
10. }

classname:ParkServiceMock

1. @isTest
2. global class ParkServiceMock implements WebServiceMock{
3. global void doInvoke (
4. Object stub,
5. Object request,
6. Map<String, Object > response,
7. String endpoint,
8. String soapAction,
9. String requestName,
10. String responseNS,
11. String responseNam,
12. String responseType
13. ){
14. ParkService.byCountryResponse response\_x = new ParkService.byCountryResponse();
15. response\_x.return\_x = new List < String >{'Yellowstone','Mackinac National Park','Yosemite'};
16. response.put ('response\_x', response\_x) ;
17. }
18. }

classname:AccountManager

1. @RestResource(urlMapping = '/Accounts/\*/contacts')
2. global with sharing class AccountManager {
3. @HttpGet
4. global static Account getAccount(){
5. RestRequest request = RestContext.request;
6. string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
7. Account result = [SELECT Id, Name,(Select Id ,Name from Contacts) from Account where Id =:accountId Limit 1];
8. return result;
9. }
10. }

classname: AccountManagerTest

1. @IsTest
2. private class AccountManagerTest {
3. @isTest static void testGetContactsByAccountId (){
4. Id recordId = createTestRecord();
5. RestRequest request = new RestRequest () ;
6. request.requesturi = 'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
7. + recordId+'/contacts';
8. request.httpMethod = 'GET';
9. RestContext.request = request;
10. Account thisAccount = AccountManager.getAccount();
11. System.assert(thisAccount != null);
12. System.assertEquals ('Test record' , thisAccount.Name);
13. }
14. static Id createTestRecord (){
15. Account accountTest = new Account (
16. Name = 'Test record');
17. insert accountTest;
18. Contact contactTest = new Contact(
19. FirstName = 'John',
20. LastName = 'Doe',
21. AccountId = accountTest.Id
22. );
23. insert contactTest;
24. return accountTest.Id;
25. }
26. }

module:Apex testing

classname:VerifyDate

1. public class VerifyDate {
2. //method to handle potential checks against two dates
3. public static Date CheckDates(Date date1, Date date2) {
4. //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month
5. if(DateWithin30Days(date1,date2)) {
6. return date2;
7. } else {
8. return SetEndOfMonthDate(date1);
9. }
10. }
11. //method to check if date2 is within the next 30 days of date1
12. private static Boolean DateWithin30Days(Date date1, Date date2) {
13. //check for date2 being in the past
14. if( date2 < date1) { return false; }
15. //check that date2 is within (>=) 30 days of date1
16. Date date30Days = date1.addDays(30); //create a date 30 days away from date1
17. if( date2 >= date30Days ) { return false; }
18. else { return true; }
19. }
20. //method to return the end of the month of a given date
21. private static Date SetEndOfMonthDate(Date date1) {
22. Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
23. Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
24. return lastDay;
25. }
26. }

classname:TestVerifyDate

1. @isTest
2. public class TestVerifyDate
3. {
4. static testMethod void testMethod1()
5. {
6. Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
7. Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
8. }
9. }

classname:RestrictContactByName

1. trigger RestrictContactByName on Contact (before insert, before update) {
3. //check contacts prior to insert or update for invalid data
4. For (Contact c : Trigger.New) {
5. if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
6. c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
7. }
8. }
9. }

classname:TestRestrictContactByName

1. @isTest
2. private class TestRestrictContactByName {
3. static testMethod void metodoTest()
4. {
5. List<Contact> listContact= new List<Contact>();
6. Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' , email='Test@test.com');
7. Contact c2 = new Contact(FirstName='Francesco1', LastName = 'INVALIDNAME',email='Test@test.com');
8. listContact.add(c1);
9. listContact.add(c2);
10. Test.startTest();
11. try
12. {
13. insert listContact;
14. }
15. catch(Exception ee)
16. {
17. }
18. Test.stopTest();
19. }
20. }

classname:LeadProcessor

1. public class LeadProcessor implements Database.Batchable<sObject> {
2. public Database.QueryLocator start(Database.BatchableContext bc) {
3. // collect the batches of records or objects to be passed to execute
4. return Database.getQueryLocator([Select LeadSource From Lead ]);
5. }
6. public void execute(Database.BatchableContext bc, List<Lead> leads){
7. // process each batch of records
8. for (Lead Lead : leads) {
9. lead.LeadSource = 'Dreamforce';
10. }
11. update leads;
12. }
13. public void finish(Database.BatchableContext bc){
14. }
15. }

classname:LeadProcessorTest

1. @isTest
2. public class LeadProcessorTest {
3. @testSetup
4. static void setup() {
5. List<Lead> leads = new List<Lead>();
6. for(Integer counter=0 ;counter <200;counter++){
7. Lead lead = new Lead();
8. lead.FirstName ='FirstName';
9. lead.LastName ='LastName'+counter;
10. lead.Company ='demo'+counter;
11. leads.add(lead);
12. }
13. insert leads;
14. }
15. @isTest static void test() {
16. Test.startTest();
17. LeadProcessor leadProcessor = new LeadProcessor();
18. Id batchId = Database.executeBatch(leadProcessor);
19. Test.stopTest();
20. }
21. }

classname:AddPrimaryContact

1. public class AddPrimaryContact implements Queueable
2. {
3. private Contact c;
4. private String state;
5. public AddPrimaryContact(Contact c, String state)
6. {
7. this.c = c;
8. this.state = state;
9. }
10. public void execute(QueueableContext context)
11. {
12. List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
13. List<Contact> lstContact = new List<Contact>();
14. for (Account acc:ListAccount)
15. {
16. Contact cont = c.clone(false,false,false,false);
17. cont.AccountId = acc.id;
18. lstContact.add( cont );
19. }
20. if(lstContact.size() >0 )
21. {
22. insert lstContact;
23. }
24. }
25. }

classname:AddPrimaryContactTest

1. @isTest
2. public class AddPrimaryContactTest
3. {
4. @isTest static void TestList()
5. {
6. List<Account> Teste = new List <Account>();
7. for(Integer i=0;i<50;i++)
8. {
9. Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
10. }
11. for(Integer j=0;j<50;j++)
12. {
13. Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
14. }
15. insert Teste;
16. Contact co = new Contact();
17. co.FirstName='demo';
18. co.LastName ='demo';
19. insert co;
20. String state = 'CA';
21. AddPrimaryContact apc = new AddPrimaryContact(co, state);
22. Test.startTest();
23. System.enqueueJob(apc);
24. Test.stopTest();
25. }
26. }

classname:DailyLeadProcessor

1. public class DailyLeadProcessor implements Schedulable {
2. Public void execute(SchedulableContext SC){
3. List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
4. for(Lead l:LeadObj){
5. l.LeadSource='Dreamforce';
6. update l;
7. }
8. }
9. }

classname:DailyLeadProcessorTest

1. @isTest
2. private class DailyLeadProcessorTest {
3. static testMethod void testDailyLeadProcessor() {
4. String CRON\_EXP = '0 0 1 \* \* ?';
5. List<Lead> lList = new List<Lead>();
6. for (Integer i = 0; i < 200; i++) {
7. lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open - Not Contacted'));
8. }
9. insert lList;
11. Test.startTest();
12. String jobId = System.schedule('DailyLeadProcessor', CRON\_EXP, new DailyLeadProcessor());
13. }
14. }

# module:apex superbadge(Automate record creation)

classname:MaintenanceRequestHelper

1. public with sharing class MaintenanceRequestHelper {
2. public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
3. Set<Id> validIds = new Set<Id>();
4. For (Case c : updWorkOrders){
5. if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
6. if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
7. validIds.add(c.Id);
8. }
9. }
10. }
11. if (!validIds.isEmpty()){
12. List<Case> newCases = new List<Case>();
13. 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)
14. FROM Case WHERE Id IN :validIds]);
15. Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
16. 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];
17. for (AggregateResult ar : results){
18. maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));
19. }
20. for(Case cc : closedCasesM.values()){
21. Case nc = new Case (
22. ParentId = cc.Id,
23. Status = 'New',
24. Subject = 'Routine Maintenance',
25. Type = 'Routine Maintenance',
26. Vehicle\_\_c = cc.Vehicle\_\_c,
27. Equipment\_\_c =cc.Equipment\_\_c,
28. Origin = 'Web',
29. Date\_Reported\_\_c = Date.Today()
30. );
31. If (maintenanceCycles.containskey(cc.Id)){
32. nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
33. } else {
34. nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);
35. }
36. newCases.add(nc);
37. }
38. insert newCases;
39. List<Equipment\_Maintenance\_Item\_\_c> clonedWPs = new List<Equipment\_Maintenance\_Item\_\_c>();
40. for (Case nc : newCases){
41. for (Equipment\_Maintenance\_Item\_\_c wp : closedCasesM.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){
42. Equipment\_Maintenance\_Item\_\_c wpClone = wp.clone();
43. wpClone.Maintenance\_Request\_\_c = nc.Id;
44. ClonedWPs.add(wpClone);
45. }
46. }
47. insert ClonedWPs;
48. }
49. }
50. }

classname:MaintenanceRequest

1. trigger MaintenanceRequest on Case (before update, after update) {
2. if(Trigger.isUpdate && Trigger.isAfter){
3. MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
4. }
5. }

# Synchronize Salesforce data with an external system:

classname:WarehouseCalloutService

1. public with sharing class WarehouseCalloutService implements Queueable {
2. private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';
3. //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
4. //The callout’s JSON response returns the equipment records that you upsert in Salesforce.
5. @future(callout=true)
6. public static void runWarehouseEquipmentSync(){
7. Http http = new Http();
8. HttpRequest request = new HttpRequest();
9. request.setEndpoint(WAREHOUSE\_URL);
10. request.setMethod('GET');
11. HttpResponse response = http.send(request);
12. List<Product2> warehouseEq = new List<Product2>();
13. if (response.getStatusCode() == 200){
14. List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
15. System.debug(response.getBody());
16. //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
17. //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
18. for (Object eq : jsonResponse){
19. Map<String,Object> mapJson = (Map<String,Object>)eq;
20. Product2 myEq = new Product2();
21. myEq.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');
22. myEq.Name = (String) mapJson.get('name');
23. myEq.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');
24. myEq.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');
25. myEq.Cost\_\_c = (Integer) mapJson.get('cost');
26. myEq.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');
27. myEq.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');
28. myEq.ProductCode = (String) mapJson.get('\_id');
29. warehouseEq.add(myEq);
30. }
31. if (warehouseEq.size() > 0){
32. upsert warehouseEq;
33. System.debug('Your equipment was synced with the warehouse one');
34. }
35. }
36. }
37. public static void execute (QueueableContext context){
38. runWarehouseEquipmentSync();
39. }
40. }
41. ----------------------------------------------------------------
42. After saving the code open execute anonymous window ( CTRl+E ) and run this method ,
43. -------------------------------------------
44. System.enqueueJob(new WarehouseCalloutService());

# Schedule synchronization:

classname:WarehouseSyncSchedule

1. global with sharing class WarehouseSyncSchedule implements Schedulable{
2. global void execute(SchedulableContext ctx){
3. System.enqueueJob(new WarehouseCalloutService());
4. }
5. }

# Test automation logic:

classname:MaintenanceRequestHelperTest

1. @istest
2. public with sharing class MaintenanceRequestHelperTest {
3. private static final string STATUS\_NEW = 'New';
4. private static final string WORKING = 'Working';
5. private static final string CLOSED = 'Closed';
6. private static final string REPAIR = 'Repair';
7. private static final string REQUEST\_ORIGIN = 'Web';
8. private static final string REQUEST\_TYPE = 'Routine Maintenance';
9. private static final string REQUEST\_SUBJECT = 'Testing subject';
10. PRIVATE STATIC Vehicle\_\_c createVehicle(){
11. Vehicle\_\_c Vehicle = new Vehicle\_\_C(name = 'SuperTruck');
12. return Vehicle;
13. }
14. PRIVATE STATIC Product2 createEq(){
15. product2 equipment = new product2(name = 'SuperEquipment',
16. lifespan\_months\_\_C = 10,
17. maintenance\_cycle\_\_C = 10,
18. replacement\_part\_\_c = true);
19. return equipment;
20. }
21. PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
22. case cs = new case(Type=REPAIR,
23. Status=STATUS\_NEW,
24. Origin=REQUEST\_ORIGIN,
25. Subject=REQUEST\_SUBJECT,
26. Equipment\_\_c=equipmentId,
27. Vehicle\_\_c=vehicleId);
28. return cs;
29. }
30. PRIVATE STATIC Equipment\_Maintenance\_Item\_\_c createWorkPart(id equipmentId,id requestId){
31. Equipment\_Maintenance\_Item\_\_c wp = new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipmentId,
32. Maintenance\_Request\_\_c = requestId);
33. return wp;
34. }
35. @istest
36. private static void testMaintenanceRequestPositive(){
37. Vehicle\_\_c vehicle = createVehicle();
38. insert vehicle;
39. id vehicleId = vehicle.Id;
40. Product2 equipment = createEq();
41. insert equipment;
42. id equipmentId = equipment.Id;
43. case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
44. insert somethingToUpdate;
45. Equipment\_Maintenance\_Item\_\_c workP = createWorkPart(equipmentId,somethingToUpdate.id);
46. insert workP;
47. test.startTest();
48. somethingToUpdate.status = CLOSED;
49. update somethingToUpdate;
50. test.stopTest();
51. Case newReq = [Select id, subject, type, Equipment\_\_c, Date\_Reported\_\_c, Vehicle\_\_c, Date\_Due\_\_c
52. from case
53. where status =:STATUS\_NEW];
54. Equipment\_Maintenance\_Item\_\_c workPart = [select id
55. from Equipment\_Maintenance\_Item\_\_c
56. where Maintenance\_Request\_\_c =:newReq.Id];
57. system.assert(workPart != null);
58. system.assert(newReq.Subject != null);
59. system.assertEquals(newReq.Type, REQUEST\_TYPE);
60. SYSTEM.assertEquals(newReq.Equipment\_\_c, equipmentId);
61. SYSTEM.assertEquals(newReq.Vehicle\_\_c, vehicleId);
62. SYSTEM.assertEquals(newReq.Date\_Reported\_\_c, system.today());
63. }
64. @istest
65. private static void testMaintenanceRequestNegative(){
66. Vehicle\_\_C vehicle = createVehicle();
67. insert vehicle;
68. id vehicleId = vehicle.Id;
69. product2 equipment = createEq();
70. insert equipment;
71. id equipmentId = equipment.Id;
72. case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
73. insert emptyReq;
74. Equipment\_Maintenance\_Item\_\_c workP = createWorkPart(equipmentId, emptyReq.Id);
75. insert workP;
76. test.startTest();
77. emptyReq.Status = WORKING;
78. update emptyReq;
79. test.stopTest();
80. list<case> allRequest = [select id
81. from case];
82. Equipment\_Maintenance\_Item\_\_c workPart = [select id
83. from Equipment\_Maintenance\_Item\_\_c
84. where Maintenance\_Request\_\_c = :emptyReq.Id];
85. system.assert(workPart != null);
86. system.assert(allRequest.size() == 1);
87. }
88. @istest
89. private static void testMaintenanceRequestBulk(){
90. list<Vehicle\_\_C> vehicleList = new list<Vehicle\_\_C>();
91. list<Product2> equipmentList = new list<Product2>();
92. list<Equipment\_Maintenance\_Item\_\_c> workPartList = new list<Equipment\_Maintenance\_Item\_\_c>();
93. list<case> requestList = new list<case>();
94. list<id> oldRequestIds = new list<id>();
95. for(integer i = 0; i < 300; i++){
96. vehicleList.add(createVehicle());
97. equipmentList.add(createEq());
98. }
99. insert vehicleList;
100. insert equipmentList;
101. for(integer i = 0; i < 300; i++){
102. requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
103. }
104. insert requestList;
105. for(integer i = 0; i < 300; i++){
106. workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
107. }
108. insert workPartList;
109. test.startTest();
110. for(case req : requestList){
111. req.Status = CLOSED;
112. oldRequestIds.add(req.Id);
113. }
114. update requestList;
115. test.stopTest();
116. list<case> allRequests = [select id
117. from case
118. where status =: STATUS\_NEW];
119. list<Equipment\_Maintenance\_Item\_\_c> workParts = [select id
120. from Equipment\_Maintenance\_Item\_\_c
121. where Maintenance\_Request\_\_c in: oldRequestIds];
122. system.assert(allRequests.size() == 300);
123. }
124. }

classname:MaintenanceRequestHelper

1. public with sharing class MaintenanceRequestHelper {
2. public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
3. Set<Id> validIds = new Set<Id>();
4. For (Case c : updWorkOrders){
5. if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
6. if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
7. validIds.add(c.Id);
8. }
9. }
10. }
11. if (!validIds.isEmpty()){
12. List<Case> newCases = new List<Case>();
13. 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)
14. FROM Case WHERE Id IN :validIds]);
15. Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
16. 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];
17. for (AggregateResult ar : results){
18. maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));
19. }
20. for(Case cc : closedCasesM.values()){
21. Case nc = new Case (
22. ParentId = cc.Id,
23. Status = 'New',
24. Subject = 'Routine Maintenance',
25. Type = 'Routine Maintenance',
26. Vehicle\_\_c = cc.Vehicle\_\_c,
27. Equipment\_\_c =cc.Equipment\_\_c,
28. Origin = 'Web',
29. Date\_Reported\_\_c = Date.Today()
30. );
31. If (maintenanceCycles.containskey(cc.Id)){
32. nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
33. }
34. newCases.add(nc);
35. }
36. insert newCases;
37. List<Equipment\_Maintenance\_Item\_\_c> clonedWPs = new List<Equipment\_Maintenance\_Item\_\_c>();
38. for (Case nc : newCases){
39. for (Equipment\_Maintenance\_Item\_\_c wp : closedCasesM.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){
40. Equipment\_Maintenance\_Item\_\_c wpClone = wp.clone();
41. wpClone.Maintenance\_Request\_\_c = nc.Id;
42. ClonedWPs.add(wpClone);
43. }
44. }
45. insert ClonedWPs;
46. }
47. }
48. }

# Test callout logic:

classname:WarehouseCalloutServiceMock

1. @isTest
2. global class WarehouseCalloutServiceMock implements HttpCalloutMock {
3. // implement http mock callout
4. global static HttpResponse respond(HttpRequest request){
5. System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment', request.getEndpoint());
6. System.assertEquals('GET', request.getMethod());
7. // Create a fake response
8. HttpResponse response = new HttpResponse();
9. response.setHeader('Content-Type', 'application/json');
10. response.setBody('[{"\_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
11. response.setStatusCode(200);
12. return response;
13. }
14. }

classname:WarehouseCalloutService

1. public with sharing class WarehouseCalloutService {
2. private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';
3. //@future(callout=true)
4. public static void runWarehouseEquipmentSync(){
5. Http http = new Http();
6. HttpRequest request = new HttpRequest();
7. request.setEndpoint(WAREHOUSE\_URL);
8. request.setMethod('GET');
9. HttpResponse response = http.send(request);
10. List<Product2> warehouseEq = new List<Product2>();
11. if (response.getStatusCode() == 200){
12. List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
13. System.debug(response.getBody());
14. for (Object eq : jsonResponse){
15. Map<String,Object> mapJson = (Map<String,Object>)eq;
16. Product2 myEq = new Product2();
17. myEq.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');
18. myEq.Name = (String) mapJson.get('name');
19. myEq.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');
20. myEq.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');
21. myEq.Cost\_\_c = (Decimal) mapJson.get('lifespan');
22. myEq.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');
23. myEq.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');
24. warehouseEq.add(myEq);
25. }
26. if (warehouseEq.size() > 0){
27. upsert warehouseEq;
28. System.debug('Your equipment was synced with the warehouse one');
29. System.debug(warehouseEq);
30. }
31. }
32. }
33. }

classname:WarehouseCalloutServiceTest

1. @isTest
2. private class WarehouseCalloutServiceTest {
3. @isTest
4. static void testWareHouseCallout(){
5. Test.startTest();
6. // implement mock callout test here
7. Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
8. WarehouseCalloutService.runWarehouseEquipmentSync();
9. Test.stopTest();
10. System.assertEquals(1, [SELECT count() FROM Product2]);
11. }
12. }

# Test scheduling logic:

classname:WarehouseSyncSchedule

1. global class WarehouseSyncSchedule implements Schedulable {
2. global void execute(SchedulableContext ctx) {
3. WarehouseCalloutService.runWarehouseEquipmentSync();
4. }
5. }

classname:WarehouseSyncScheduleTest

1. @isTest
2. public class WarehouseSyncScheduleTest {
3. @isTest static void WarehousescheduleTest(){
4. String scheduleTime = '00 00 01 \* \* ?';
5. Test.startTest();
6. Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
7. String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSyncSchedule());
8. Test.stopTest();
9. //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems.
10. // This object is available in API version 17.0 and later.
11. CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
12. System.assertEquals(jobID, a.Id,'Schedule ');
13. }
14. }