

Nandini Agarwalla

Salesforce Developer Catalyst Self-Learning & Super Badges

- APEX TRIGGERS

1. Get Started With Apex Triggers

AccountAddressTrigger

```
1 trigger AccountAddressTrigger on Account (before insert,before
  update) {
2     for (Account account:trigger.new){
3         if(account.Match_Billing_Address__c==true){
4             account.ShippingPostalCode=account.BillingPostalCode;
5         }
6     }
7
8 }
```

2. Bulk Apex Triggers

ClosedOpportunityTrigger

```
1 trigger ClosedOpportunityTrigger on Opportunity (after
  insert,after update) {
2     List<Task> tasklist = new List<Task>();
3     for(Opportunity opp : Trigger.new){
4         if(opp.StageName == 'Closed Won'){
5             tasklist.add(new task(Subject='Follow Up Test
6
7         }
8     }
9     if (tasklist.size()>0){
10        insert tasklist;
11    }
```

11

12 }

- Apex Testing

1. Get Started with Apex Unit Tests

► VerifyDate

```
1 public class VerifyDate {
2
3     //method to handle potential checks against two dates
4     public static Date CheckDates(Date date1, Date date2) {
5         //if date2 is within the next 30 days of date1, use date2.
6         //Otherwise use the end of the month
7         if(DateWithin30Days(date1,date2)) {
8             return date2;
9         } else {
10            return SetEndOfMonthDate(date1);
11        }
12    }
13
14    //method to check if date2 is within the next 30 days of date1
15    private static Boolean DateWithin30Days(Date date1, Date date2)
16    {
17        //check for date2 being in the past
18        if( date2 < date1) { return false; }
19
20        //check that date2 is within (>=) 30 days of date1
21        Date date30Days = date1.addDays(30); //create a date 30 days
22        //away from date1
23        if( date2 >= date30Days ) { return false; }
24        else { return true; }
25    }
26
27    //method to return the end of the month of a given date
28    private static Date SetEndOfMonthDate(Date date1) {
29        Integer totalDays = Date.daysInMonth(date1.year(),
30        date1.month());
31        Date lastDay = Date.newInstance(date1.year(),
32        date1.month(), totalDays);
33        return lastDay;
34    }
35 }
```

```
30
31 }
```

➤ TestVerifyDate

```
1  @isTest
2  private class TestVerifyDate {
3      @isTest static void testInRange() {
4          Date day =
5              VerifyDate.CheckDates(date.parse('2/2/22'),date.parse('2/1/22'));
6          System.assertEquals(date.parse('28/2/22'), day);
7      }
8      @isTest static void testOutOfRange() {
9          Date day =
10             VerifyDate.CheckDates(date.parse('3/1/22'),date.parse('3/2/22'));
11             System.assertEquals(date.parse('3/2/22'),day);
12     }
```

2. Test Apex Triggers

➤ RestrictContactByName

```
1  trigger RestrictContactByName on Contact (before insert, before
   update) {
2
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+'"'
   not allowed for'
7                  +' DML');
8          }
9      }
```

```
10 }
```

► TestRestrictContactByName

```
1 @istest
2 public class TestRestrictContactByName {
3     @isTest static void TestNewContact() {
4         Contact cont = new Contact();
5         cont.LastName = 'INVALIDNAME' ;
6
7         Test.startTest();
8         Database.SaveResult result = Database.insert(cont,
9             false);
9         Test.stopTest();
10
11         System.assert(!result.isSuccess());
12         System.assert(result.getErrors().size() > 0);
13         System.assertEquals('The Last Name "INVALIDNAME" is not
14             allowed for'
15             + ' DML', result.getErrors()[0].getMessage());
16     }
17 }
```

3. Create Test Data for Apex Tests

RandomContactFactory

```
1 public class RandomContactFactory {
2     Public Static List < Contact > generateRandomContacts(Integer
3         NumCont, string lastname){
4         List<Contact> contacts =new List<Contact>();
5         for(Integer i=0;i<NumCont;i++){
6             Contact cont=new Contact(FirstName = 'abc'+i, LastName =
7                 lastname);
8             contacts.add(cont);
9         }
10    }
```

```
7     }
8     return contacts;
9 }
10
11 }
```

- **Asynchronous Apex**

1. Use Future Methods

➤ AccountProcessor

```
1 public class AccountProcessor {
2     @future
3     public static void countContacts(List<Id> accountIds) {
4         List<Account> accountsToUpdate = new List<Account>();
5
6         List<Account> accounts = [Select Id, Name, (Select Id from
Contacts) from Account Where Id IN :accountIds];
7
8         For(Account acc:accounts){
9             List<Contact> contactList = acc.Contacts;
10            acc.Number_of_Contacts__c = contactList.size();
11            accountsToUpdate.add(acc);
12        }
13        update accountsToUpdate;
14    }
15
16 }
```

➤ AccountProcessorTest

```
1 @IsTest
2 private class AccountProcessorTest {
3     @IsTest
4     private static void testCountContacts() {
5         Account newAccount = new Account(Name='Test Account');
6         insert newAccount;
7         Contact newContact1 = new
Contact(FirstName='jane',LastName='flow',AccountId=newAccount.Id
);
8         insert newContact1;
9         Contact newContact2 = new
Contact(FirstName='charles',LastName='flow',AccountId=newAccount.
Id);
10        insert newContact2;
11
12        List<Id> accountIds =new List<Id>();
13        accountIds.add(newAccount.Id);
14
15        Test.startTest();
16        AccountProcessor.countContacts(accountIds);
17        Test.stopTest();
18    }
19 }
```

2. Use Batch Apex

➤ LeadProcessor

```
1public without sharing class LeadProcessor implements
Database.Batchable<SObject> {
2
3    public Database.QueryLocator start(Database.BatchableContext
dbc) {
4        return Database.getQueryLocator([SELECT ID,Name FROM
Lead]);
5    }
```

```

5     }
6     public void execute(Database.BatchableContext
    dbc,List<Lead>leads){
7         for(Lead l: leads){
8             l.LeadSource='Dreamforce';
9         }
10        update leads;
11    }
12    public void finish(Database.BatchableContext dbc){
13        system.debug('Done');
14    }
15
16 }

```

➤ LeadProcessorTest

```

1 @isTest
2 private class LeadProcessorTest {
3     @isTest
4     private static void testBatchClass() {
5         List<Lead> leads = new List<Lead>();
6         for (Integer i=0;i<200;i++) {
7             leads.add(new
8             Lead(Lastname='paren',Company='Saleyc0'));
9         }
10        insert leads;
11        Test.startTest();
12        LeadProcessor lp = new LeadProcessor();
13        Id batchId = Database.executeBatch(lp,200);
14        Test.stopTest();
15        List<Lead>updatedLeads =[SELECT Id FROM Lead WHERE
16        LeadSource='Dreamforce'];
17        System.assertEquals(200, updatedLeads.size(),'ERROR:At
18        least 1 lead record not updated'
19        + ' correctly');
20    }
21 }

```


3. Control Processes with Queueable Apex

➤ AddPrimaryContact

```
1 public without sharing class AddPrimaryContact implements
  Queueable {
2     private Contact contact;
3     private String state;
4     public AddPrimaryContact(Contact inputContact, String
      inputState) {
5         this.contact = inputContact;
6         this.state = inputState;
7     }
8     public void execute(QueueableContext context) {
9         List<Account> accounts = [select id from account where
      billingstate = :state LIMIT 200];
10        List<Contact> contacts= new List<Contact>();
11        for (Account acc : accounts) {
12            Contact contactClone = contact.clone();
13            contactClone.AccountId =acc.Id;
14            contacts.add(contactClone);
15        }
16        insert contacts;
17    }
18 }
```

➤ AddPrimaryContactTest

```
1 @isTest
2 private class AddPrimaryContactTest {
3     @isTest
4     private static void testqueueableclass() {
5         List<Account> accounts = new List<Account>();
6
7         for (Integer i = 0; i < 500; i++) {
8             Account acc=new Account(Name='Test account');
9             if(i<250){
```

```
10         acc.BillingState='NY';
11     }else{
12         acc.BillingState='CA';
13     }
14     accounts.add(acc);
15 }
16 insert accounts;
17
18     Contact contact=new
19     Contact(Firstname='paren',Lastname='Saleyc');
20     insert contact;
21
22     Test.startTest();
23     Id jobId =System.enqueueJob(new
24     AddPrimaryContact(contact,'CA'));
25     Test.stopTest();
26
27     List<Contact> contacts = [select id from Contact where
28     Contact.account.billingstate = : 'CA'];
29     System.assertEquals(200, contacts.size(),'ERROR:incorrect
30
31 }
32 }
```

4. Schedule Jobs Using the Apex Scheduler

➤ DailyLeadProcessor

```
1 public without sharing class DailyLeadProcessor implements
   Schedulable {
2     public void execute(SchedulableContext ctx) {
3         List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
   LeadSource = null Limit 200];
4         for(lead l:leads){
5             l.LeadSource='Dreamforce';
6         }
7         update leads;
8     }
9 }
```

➤ DailyLeadProcessorTest

```
1 @isTest
2 private class DailyLeadProcessorTest {
3
4     public static String CRON_EXP = '0 0 0 ? * * *';
5     @isTest
6     private static void testSchedulableClass() {
7
8         List<Lead> leads = new List<Lead>();
9         for (Integer i=0; i<500; i++) {
10             if(i<250){
11                 leads.add(new
   Lead(Lastname='paren',Company='Saleco'));
12             }else{
13                 leads.add(new
   Lead(Lastname='paren',Company='Saleco',LeadSource='Other'));
14             }
15         }
16         insert leads;
17
18         Test.startTest();
19         String jobId = System.schedule('Process
```

```
20         Test.stopTest();
21
22         List<Lead> updatedLeads = [SELECT Id,LeadSource FROM Lead
WHERE LeadSource='Dreamforce'];
23         System.assertEquals(200, updatedLeads.size(),'ERROR:At
least 1 lead record not updated'
24 +' correctly');
25
26         List<CronTrigger>cts = [SELECT
Id,TimesTriggered,NextFireTime FROM CronTrigger WHERE Id=
:jobId];
27         System.debug('Next fire time'+cts[0].NextFireTime);
28     }
29 }
```

- Apex Integration Services

2. Apex REST Callouts

► AnimalLocator

```
1 public class AnimalLocator {
2     public static string getAnimalNameById(Integer i) {
3         Http http = new Http();
4         HttpRequest request = new HttpRequest();
5         request.setEndpoint('https://th-apex-http-
6
6         request.setMethod('GET');
7         Map<String, Object> animal = new Map<String, Object>();
8         HttpResponse response = http.send(request);
9         if(response.getStatusCode()==200){
10             Map<String, Object> result = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
11             animal = (Map<String, Object>)result.get('animal');
12
13         }
14
15     return string.valueOf(animal.get('name'));
16
17     }
18
19 }
```

► AnimalLocatorTest

```
1 @isTest
2 private class AnimalLocatorTest {
3     @isTest
4     static void testanimallocator() {
5         Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
6         String actual = AnimalLocator.getAnimalNameById(1);
7         String expected = 'moose';
8         System.assertEquals(actual,expected);
9     }
```

```
10 }
```

➤ AnimalLocatorMock

```
1 @isTest
2 global class AnimalLocatorMock implements HttpCalloutMock{
3     global HTTPResponse respond(HTTPRequest request) {
4         HTTPResponse response = new HTTPResponse();
5         response.setHeader('contentType', 'application/json');
6         response.setBody('{"animals": {"id":1,
7             }}}');
8         response.setStatusCode(200);
9         return response;
10    }
```

3. Apex SOAP Callouts

➤ ParkLocator

```
1 public class ParkLocator {
2     public static List<String> country(String country) {
3         ParkService.ParksImplPort prkSvc = new
4         ParkService.ParksImplPort();
5         return prkSvc.byCountry(country);
6     }
7 }
```

➤ ParkLocatorTest

```
1 @isTest
2 public class ParkLocatorTest {
3     @isTest static void testCallout() {
4         Test.setMock(WebServiceMock.class, new ParkServiceMock());
5         String country = 'United States';
6         List<String> expectedParks = new List<String>{'Germany',
7             'India', 'Japan', 'United States'};
8         System.assertEquals(expectedParks,
9             ParkLocator.country(country));
10    }
```

```
9 }
```

➤ 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 responseName,
12        String responseType) {
13        // start - specify the response you want to send
14        parkService.byCountryResponse response_x = new
15        parkService.byCountryResponse();
16        response_x.return_x = new List<String>{'Germany',
17        'India','Japan','United States'};
18        response.put('response_x', response_x);
19    }
20 }
```

4. Apex Web Services

➤ AccountManager

```
1 @RestResource(urlMapping='/Account/*/contacts')
2 global with sharing class AccountManager {
3
4     @HttpGet
5     global static Account getAccount() {
6         RestRequest request = RestContext.request;
7         String accountId =
8         request.requestURI.substringBetween('Accounts/', '/contacts');
9         Account result = [SELECT ID,Name,(SELECT
10         ID,FirstName,LastName FROM Contacts)
11     }
```

```

9             FROM Account
10             WHERE Id = :accountId];
11     return result;
12
13 }
14 }

```

► AccountManagerTest

```

1 @IsTest
2 private class AccountManagerTest {
3     @isTest static void testGetAccount() {
4         Account a = new Account(Name='TestAccount');
5         insert a;
6         Contact c = new Contact(AccountId=a.Id,
7             FirstName='Test', LastName='Test');
8         insert c;
9
10        RestRequest request = new RestRequest();
11        request.requestUri
12        ='https://yourInstance.my.salesforce.com/services/apexrest/AccountManager/getAccount';
13
14        request.httpMethod = 'GET';
15        RestContext.request = request;
16
17        Account myAcct = AccountManager.getAccount();
18        System.assert(myAcct != null);
19        System.assertEquals('Test Account', myAcct.Name);
20    }
21 }

```


- Apex Specialist SuperBadges

Challenge 1-Automated Record Creation

➤ MaintenanceRequest

```
1 trigger MaintenanceRequest on Case (before update, after update) {
2     if (Trigger.isUpdate && Trigger.isAfter) {
3         MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
4             Trigger.OldMap);
5     }
6 }
```

➤ MaintenanceRequestHelper

```
1 public with sharing class MaintenanceRequestHelper {
2     public static void updateWorkOrders(List<Case> updWorkOrders,
3         Map<Id,Case> nonUpdCaseMap) {
4         Set<Id> validIds = new Set<Id>();
5         For (Case c : updWorkOrders) {
6             if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
7                 c.Status == 'Closed') {
8                 if (c.Type == 'Repair' || c.Type == 'Routine
9
10                    validIds.add(c.Id);
11                }
12            }
13        }
14
15        //When an existing maintenance request of type Repair or
16        //Routine Maintenance is closed,
17        //create a new maintenance request for a future routine
18        //checkup.
19        if (!validIds.isEmpty()) {
20            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT
21                Id, Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,
22                (SELECT
23                    Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
24                FROM
25                Case WHERE Id IN :validIds]);
26        }
27    }
28 }
```

```

18         Map<Id,Decimal> maintenanceCycles = new
19         Map<ID,Decimal>();
20         //calculate the maintenance request due dates by
21         using the maintenance cycle defined on the related equipment
22         records.
23         AggregateResult[] results = [SELECT
24         Maintenance_Request__c,
25         MIN(Equipment__r.Maintenance_Cycle__c)cycle
26         FROM
27         Equipment_Maintenance_Item__c
28         WHERE
29         Maintenance_Request__c IN :ValidIds GROUP BY
30         Maintenance_Request__c];
31         for (AggregateResult ar : results){
32             maintenanceCycles.put((Id)
33             ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
34         }
35
36         List<Case> newCases = new List<Case>();
37         for(Case cc : closedCases.values()){
38             Case nc = new Case (
39                 ParentId = cc.Id,
40                 Status = 'New',
41                 Subject = 'Routine Maintenance',
42                 Type = 'Routine Maintenance',
43                 Vehicle__c = cc.Vehicle__c,
44                 Equipment__c =cc.Equipment__c,
45                 Origin = 'Web',
46                 Date_Reported__c = Date.Today()
47             );
48
49             //If multiple pieces of equipment are used in the
50             maintenance request,
51             //define the due date by applying the shortest
52             maintenance cycle to today's date.
53             //If (maintenanceCycles.containsKey(cc.Id)){
54                 nc.Date_Due__c =

```

```

    Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
47         //} else {
48         //    nc.Date_Due__c =
    Date.today().addDays((Integer)
    cc.Equipment__r.maintenance_Cycle__c);
49         //}
50
51         newCases.add(nc);
52     }
53
54     insert newCases;
55
56     List<Equipment_Maintenance_Item__c> clonedList = new
    List<Equipment_Maintenance_Item__c>();
57     for (Case nc : newCases){
58         for (Equipment_Maintenance_Item__c clonedListItem
    : closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
59             Equipment_Maintenance_Item__c item =
    clonedListItem.clone();
60             item.Maintenance_Request__c = nc.Id;
61             clonedList.add(item);
62         }
63     }
64     insert clonedList;
65 }
66 }
67 }

```

Challenge 2-Synchronize Salesforce data with an external system

➤ WarehouseCalloutService

```

1 public with sharing class WarehouseCalloutService implements
    Queueable {
2
3     private static final String WAREHOUSE_URL =
4     'https://th-superbadge-apex.herokuapp.com/equipment';
5
6     @future(callout=true)

```

```

7     public static void runWarehouseEquipmentSync(){
8         System.debug('go into runWarehouseEquipmentSync');
9         Http http = new Http();
10        HttpRequest request = new HttpRequest();
11
12        request.setEndpoint(WAREHOUSE_URL);
13        request.setMethod('GET');
14        HttpResponse response = http.send(request);
15
16        List<Product2> product2List = new List<Product2>();
17        System.debug(response.getStatusCode());
18        if (response.getStatusCode() == 200){
19            List<Object> jsonResponse =
20            (List<Object>)JSON.deserializeUntyped(response.getBody());
21            System.debug(response.getBody());
22
23            //class maps the following fields:
24            //warehouse SKU will be external ID for identifying
25            which equipment records to update within Salesforce
26            for (Object jR : jsonResponse){
27                Map<String,Object> mapJson =
28                (Map<String,Object>)jR;
29                Product2 product2 = new Product2();
30                //replacement part (always true),
31                product2.Replacement_Part__c = (Boolean)
32                mapJson.get('replacement');
33                //cost
34                product2.Cost__c = (Integer) mapJson.get('cost');
35                //current inventory
36                product2.Current_Inventory__c = (Double)
37                mapJson.get('quantity');
38                //lifespan
39                product2.Lifespan_Months__c = (Integer)
40                mapJson.get('lifespan');
41                //maintenance cycle
42                product2.Maintenance_Cycle__c = (Integer)
43                mapJson.get('maintenanceperiod');
44                //warehouse SKU
45                product2.Warehouse_SKU__c = (String)
46                mapJson.get('sku');

```

```

39
40         product2.Name = (String) mapJson.get('name');
41         product2.ProductCode = (String)
mapJson.get('_id');
42         product2List.add(product2);
43     }
44
45     if (product2List.size() > 0){
46         upsert product2List;
47         System.debug('Your equipment was synced with the
48     }
49 }
50 }
51
52 public static void execute (QueueableContext context){
53     System.debug('start runWarehouseEquipmentSync');
54     runWarehouseEquipmentSync();
55     System.debug('end runWarehouseEquipmentSync');
56 }
57
58 }

```

Challenge 3-Schedule synchronization using Apex code

➤ WarehouseSyncSchedule

```

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

```

Challenge 4-Test automation logic

➤ MaintenanceRequestHelperTest

```

1 @isTest
2 public with sharing class MaintenanceRequestHelperTest {

```

```

3
4 // createVehicle
5 private static Vehicle__c createVehicle(){
6     Vehicle__c vehicle = new Vehicle__C(name = 'Testing

7     return vehicle;
8 }
9
10 // createEquipment
11 private static Product2 createEquipment(){
12     product2 equipment = new product2(name = 'Testing

13                                     lifespan_months__c =
14     10,
15                                     maintenance_cycle__c =
16                                     10,
17                                     replacement_part__c =
18     true);
19     return equipment;
20 }
21 // createMaintenanceRequest
22 private static Case createMaintenanceRequest(id vehicleId, id
equipmentId){
23     case cse = new case(Type='Repair',
24                         Status='New',
25                         Origin='Web',
26                         Subject='Testing subject',
27                         Equipment__c=equipmentId,
28                         Vehicle__c=vehicleId);
29     return cse;
30 }
31 // createEquipmentMaintenanceItem
32 private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
33     Equipment_Maintenance_Item__c equipmentMaintenanceItem =
new Equipment_Maintenance_Item__c(
34     Equipment__c = equipmentId,
35     Maintenance_Request__c = requestId);

```



```

Maintenance_Request__c =:newCase.Id];
72     list<case> allCase = [select id from case];
73     system.assert(allCase.size() == 2);
74
75     system.assert(newCase != null);
76     system.assert(newCase.Subject != null);
77     system.assertEquals(newCase.Type, 'Routine Maintenance');
78     SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
79     SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
80     SYSTEM.assertEquals(newCase.Date_Reported__c,
system.today());
81 }
82
83 @isTest
84 private static void testNegative(){
85     Vehicle__C vehicle = createVehicle();
86     insert vehicle;
87     id vehicleId = vehicle.Id;
88
89     product2 equipment = createEquipment();
90     insert equipment;
91     id equipmentId = equipment.Id;
92
93     case createdCase =
createMaintenanceRequest(vehicleId,equipmentId);
94     insert createdCase;
95
96     Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
97     insert workP;
98
99     test.startTest();
100     createdCase.Status = 'Working';
101     update createdCase;
102     test.stopTest();
103
104     list<case> allCase = [select id from case];
105
106     Equipment_Maintenance_Item__c equipmentMaintenanceItem =
[select id
107                                     from

```



```

Equipment_Maintenance_Item__c
108                                     where
Maintenance_Request__c = :createdCase.Id];
109
110     system.assert(equipmentMaintenanceItem != null);
111     system.assert(allCase.size() == 1);
112 }
113
114 @isTest
115 private static void testBulk(){
116     list<Vehicle__C> vehicleList = new list<Vehicle__C>();
117     list<Product2> equipmentList = new list<Product2>();
118     list<Equipment_Maintenance_Item__c>
equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
119     list<case> caseList = new list<case>();
120     list<id> oldCaseIds = new list<id>();
121
122     for(integer i = 0; i < 300; i++){
123         vehicleList.add(createVehicle());
124         equipmentList.add(createEquipment());
125     }
126     insert vehicleList;
127     insert equipmentList;
128
129     for(integer i = 0; i < 300; i++){
130
caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
131     }
132     insert caseList;
133
134     for(integer i = 0; i < 300; i++){
135
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(e
136     )
137     insert equipmentMaintenanceItemList;
138
139     test.startTest();

```

```
140         for(case cs : caseList){
141             cs.Status = 'Closed';
142             oldCaseIds.add(cs.Id);
143         }
144         update caseList;
145         test.stopTest();
146
147         list<case> newCase = [select id
148                             from case
149                             where status = 'New'];
150
151
152
153         list<Equipment_Maintenance_Item__c> workParts = [select
154             id
155             from
156             Equipment_Maintenance_Item__c
157             where
158             Maintenance_Request__c in: oldCaseIds];
159
160         system.assert(newCase.size() == 300);
161
162         list<case> allCase = [select id from case];
163         system.assert(allCase.size() == 600);
164     }
165 }
```

Challenge 5-Test callout logic

► WarehouseCalloutServiceTest

```
1 @IsTest
2 private class WarehouseCalloutServiceTest {
3     // implement your mock callout test here
4     @isTest
5     static void testWarehouseCallout() {
6         test.startTest();
7         test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
8         WarehouseCalloutService.execute(null);
9         test.stopTest();
10
11         List<Product2> product2List = new List<Product2>();
12         product2List = [SELECT ProductCode FROM Product2];
13
14         System.assertEquals(3, product2List.size());
15         System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
16         System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
17         System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
18     }
19 }
```

► WarehouseCalloutServiceMock

```
1 @isTest
2 global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
3     // implement http mock callout
4     global static HttpResponse respond(HttpRequest request) {
5
6         HttpResponse response = new HttpResponse();
7         response.setHeader('Content-Type', 'application/json');
8
9         response.setBody(' [{"_id": "55d66226726b611100aaf741", "replacement
```

```

9             "name":"Generator 1000

10         "cost":5000,"sku":"100003"},{"_id":"55d66226726b611100aaf742",'+
11         "replacement":true,"quantity":183,"name":"Cooling Fan",'+
12         "maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},'+
13         '{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":

14             "name":"Fuse
15         ,"cost":22,'+
16             "sku":"100005"}]');
17     response.setStatusCode(200);
18     return response;
19 }

```

Challenge 6-Test scheduling logic

WarehouseSyncScheduleTest

```

1 @isTest
2 public with sharing class WarehouseSyncScheduleTest {
3     // implement scheduled code here
4     //
5     @isTest static void test() {
6         String scheduleTime = '00 00 00 * * ? *';
7         Test.startTest();
8         Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
9         String jobId = System.schedule('Warehouse Time to Schedule

10         CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
=: jobId];
11         System.assertEquals('WAITING', String.valueOf(c.State),
'JobId does not match');

```

```
12
13     Test.stopTest();
14 }
15 }
```

In the Apex Specialist Superbadge, the MaintenanceRequestHelper code block creates and manages different aspects of a maintenance request for routine maintenance as well as service requests for broken or malfunctioning equipment of a vehicle for a company named HowWeRoll. The WarehouseCalloutService code block gets information from the provided url of an external site and upserts into "product2" using certain keys. MaintenanceRequestHelperTest and WarehouseCalloutServiceTest are test cases to test the coverage of their respective code blocks