Salesforce Developer Catalyst Self-Learning & Super Badges

- **APEX TRIGGERS**

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<Task> tasklist = new List<Task>();
    for(Opportunity opp : Trigger.new){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new task(Subject='Follow Up Test

        }
    }
    if (tasklist.size()>0){
        insert tasklist;
    }

}
```

- **Apex Testing**

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 < date1) { return false; }

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

 //method to return the end of the month of a given date
 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;
 }
```

```
30
31 }
```

➤ TestVerifyDate

```
1  @isTest
2  private class TestVerifyDate {
3      @isTest static void testInRange() {
4          Date day =
   VerifyDate.CheckDates(date.parse('2/2/22'),date.parse('2/1/22'));
5          System.assertEquals(date.parse('28/2/22'), day);
6      }
7      @isTest static void testOutOfRange() {
8          Date day =
   VerifyDate.CheckDates(date.parse('3/1/22'),date.parse('3/2/22'));
9          System.assertEquals(date.parse('3/2/22'),day);
10     }
11
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,
   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
   allowed for'
14                 +' DML',result.getErrors()[0].getMessage());
15     }
16
17 }
```

3. Create Test Data for Apex Tests

RandomContactFactory

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

```
        }
        return contacts;
    }

}
```

- **Asynchronous Apex**

1. Use Future Methods

➤ AccountProcessor

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

    List<Account> accounts = [Select Id, Name,(Select Id from
Contacts) from Account Where Id IN :accountIds];

    For(Account acc:accounts){
        List<Contact> 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='Test Account');
    insert newAccount;
    Contact newContact1 = new
Contact(FirstName='jane',LastName='flow',AccountId=newAccount.Id
);
    insert newContact1;
    Contact newContact2 = new
Contact(FirstName='charles',LastName='flow',AccountId=newAccount.
Id);
    insert newContact2;

    List<Id> accountIds =new List<Id>();
    accountIds.add(newAccount.Id);

    Test.startTest();
    AccountProcessor.countContacts(accountIds);
    Test.stopTest();
  }
}
```

2. Use Batch Apex

➤ LeadProcessor

```
public without sharing class LeadProcessor implements
Database.Batchable<sObject> {

   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;
    }
    public void finish(Database.BatchableContext dbc){
        system.debug('Done');
    }

}
```

➤ LeadProcessorTest

```
@isTest
private class LeadProcessorTest {
    @isTest
    private static void testBatchClass() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new
  Lead(Lastname='paren',Company='Saleyco'));
        }
        insert leads;
         Test.startTest();
         LeadProcessor lp = new LeadProcessor();
         Id batchId = Database.executeBatch(lp,200);
         Test.stopTest();
         List<Lead>updatedLeads =[SELECT Id FROM Lead WHERE
  LeadSource='Dreamforce'];
         System.assertEquals(200, updatedLeads.size(),'ERROR:At
  least 1 lead record not updated'
+' correctly');
    }
}
```

3. Control Processes with Queueable Apex

- AddPrimaryContact

```
public without sharing class AddPrimaryContact implements Queueable {
    private Contact contact;
    private String state;
    public AddPrimaryContact(Contact inputContact, String inputState) {
        this.contact = inputContact;
        this.state = inputState;
    }
    public void execute(QueueableContext context) {
       List<Account> accounts = [select id from account where billingstate = :state LIMIT 200];
        List<Contact> contacts= new List<Contact>();
        for (Account acc : accounts) {
           Contact contactClone = contact.clone();
           contactClone.AccountId =acc.Id;
           contacts.add(contactClone);
         }
         insert contacts;
     }
}
```

- AddPrimaryContactTest

```
@isTest
private class AddPrimaryContactTest {
    @isTest
    private static void testqueueableclass() {
        List<Account> accounts = new List<Account>();

        for (Integer i = 0; i < 500; i++) {
            Account acc=new Account(Name='Test account');
            if(i<250){
```

```
                acc.BillingState='NY';
            }else{
                acc.BillingState='CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact=new
Contact(Firstname='paren',Lastname='Saleyco');
        insert contact;

        Test.startTest();
        Id jobId =System.enqueueJob(new
AddPrimaryContact(contact,'CA'));
        Test.stopTest();

        List<Contact> contacts = [select id from Contact where
Contact.account.billingstate = :'CA'];
        System.assertEquals(200, contacts.size(),'ERROR:incorrect

    }
}
```

4. Schedule Jobs Using the Apex Scheduler

- DailyLeadProcessor

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

- DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 ? * * *';
    @isTest
    private static void testSchedulableClass() {

        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<500; i++) {
             if(i<250){
                 leads.add(new Lead(Lastname='paren',Company='Saleyco'));
             }else{
                 leads.add(new Lead(Lastname='paren',Company='Saleyco',LeadSource='Other'));
             }
         }
         insert leads;

         Test.startTest();
         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

```
public class AnimalLocator {
    public static string getAnimalNameById(Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-

        request.setMethod('GET');
        Map<String, Object> animal = new Map<String, Object>();
        HttpResponse response = http.send(request);
        if(response.getStatusCode()==200){
            Map<String, Object> result = (Map<String, Object>)
  JSON.deserializeUntyped(response.getBody());
      animal = (Map<String, Object>)result.get('animal');

        }

return string.valueOf(animal.get('name'));

    }

}
```

➤ AnimalLocatorTest

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

```
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          response.setStatusCode(200);
8          return response;
9      }
10 }
```

3. Apex SOAP Callouts

➤ ParkLocator

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

➤ 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',
   'India', 'Japan','United States'};
7          System.assertEquals(expectedParks,
   ParkLocator.country(country));
8      }
```

```
}
```

➤ ParkServiceMock

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
        // start - specify the response you want to send
        parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
        response_x.return_x = new List<String>{'Germany',
'India','Japan','United States'};
        response.put('response_x', response_x);
   }
}
```

4. Apex Web Services

➤ AccountManager

```
@RestResource(urlMapping='/Account/*/contacts')
global with sharing class AccountManager {

    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        String accountId =
  request.requestURI.substringBetween('Accounts/','/contacts');
        Account result =  [SELECT ID,Name,(SELECT
  ID,FirstName,LastName FROM Contacts)
```

```
                        FROM Account
                         WHERE Id = :accountId];
        return result;

    }
}
```

➤ AccountManagerTest

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetAccount() {
        Account a = new Account(Name='TestAccount');
        insert a;
        Contact c = new Contact(AccountId=a.Id,
  FirstName='Test',LastName='Test');
        insert c;

        RestRequest request = new RestRequest();
         request.requestUri
  ='https://yourInstance.my.salesforce.com/services/apexrest/Accoun

         request.httpMethod = 'GET';
         RestContext.request = request;

         Account myAcct = AccountManager.getAccount();
         System.assert(myAcct != null);
         System.assertEquals('Test Account', myAcct.Name);
    }
}
```

- **Apex Specialist SuperBadges**

Challenge 1-Automated Record Creation

- MaitenanceRequest

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

- MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine

                    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<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                                        (SELECT Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
                                                        FROM Case WHERE Id IN :validIds]);
```

```
18              Map<Id,Decimal> maintenanceCycles = new
    Map<ID,Decimal>();
19
20              //calculate the maintenance request due dates by
    using the maintenance cycle defined on the related equipment
    records.
21              AggregateResult[] results = [SELECT
    Maintenance_Request__c,
22
    MIN(Equipment__r.Maintenance_Cycle__c)cycle
23                                          FROM
    Equipment_Maintenance_Item__c
24                                          WHERE
    Maintenance_Request__c IN :ValidIds GROUP BY
    Maintenance_Request__c];
25
26              for (AggregateResult ar : results){
27                  maintenanceCycles.put((Id)
    ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
28              }
29
30              List<Case> newCases = new List<Case>();
31              for(Case cc : closedCases.values()){
32                  Case nc = new Case (
33                      ParentId = cc.Id,
34                      Status = 'New',
35                      Subject = 'Routine Maintenance',
36                      Type = 'Routine Maintenance',
37                      Vehicle__c = cc.Vehicle__c,
38                      Equipment__c =cc.Equipment__c,
39                      Origin = 'Web',
40                      Date_Reported__c = Date.Today()
41                  );
42
43                  //If multiple pieces of equipment are used in the
    maintenance request,
44                  //define the due date by applying the shortest
    maintenance cycle to today's date.
45                  //If (maintenanceCycles.containskey(cc.Id)){
46                      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 =
   (List<Object>)JSON.deserializeUntyped(response.getBody());
20             System.debug(response.getBody());
21
22             //class maps the following fields:
23             //warehouse SKU will be external ID for identifying
   which equipment records to update within Salesforce
24             for (Object jR : jsonResponse){
25                 Map<String,Object> mapJson =
   (Map<String,Object>)jR;
26                 Product2 product2 = new Product2();
27                 //replacement part (always true),
28                 product2.Replacement_Part__c = (Boolean)
   mapJson.get('replacement');
29                 //cost
30                 product2.Cost__c = (Integer) mapJson.get('cost');
31                 //current inventory
32                 product2.Current_Inventory__c = (Double)
   mapJson.get('quantity');
33                 //lifespan
34                 product2.Lifespan_Months__c = (Integer)
   mapJson.get('lifespan');
35                 //maintenance cycle
36                 product2.Maintenance_Cycle__c = (Integer)
   mapJson.get('maintenanceperiod');
37                 //warehouse SKU
38                 product2.Warehouse_SKU__c = (String)
   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

- ➤ WarehouseSyncShedule

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

```
35          return equipmentMaintenanceItem;
36      }
37
38      @isTest
39      private static void testPositive(){
40          Vehicle__c vehicle = createVehicle();
41          insert vehicle;
42          id vehicleId = vehicle.Id;
43
44          Product2 equipment = createEquipment();
45          insert equipment;
46          id equipmentId = equipment.Id;
47
48          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
49          insert createdCase;
50
51          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
52          insert equipmentMaintenanceItem;
53
54          test.startTest();
55          createdCase.status = 'Closed';
56          update createdCase;
57          test.stopTest();
58
59          Case newCase = [Select id,
60                          subject,
61                          type,
62                          Equipment__c,
63                          Date_Reported__c,
64                          Vehicle__c,
65                          Date_Due__c
66                         from case
67                         where status ='New'];
68
69          Equipment_Maintenance_Item__c workPart = [select id
70                                                    from
    Equipment_Maintenance_Item__c
71                                                    where
```

```
Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        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 = 'Working';
         update createdCase;
         test.stopTest();

         list<case> allCase = [select id from case];

         Equipment_Maintenance_Item__c equipmentMaintenanceItem =
[select id
                                                 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
    id
154                                                          from
    Equipment_Maintenance_Item__c
155                                                          where
    Maintenance_Request__c in: oldCaseIds];
156 
157         system.assert(newCase.size() == 300);
158 
159         list<case> allCase = [select id from case];
160         system.assert(allCase.size() == 600);
161     }
162 }
```

Challenge 5-Test callout logic

- 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<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);
    }
}
```

- WarehouseCalloutServiceMock

```
@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","replacement
```

```
                        '"name":"Generator 1000

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

                        '"name":"Fuse
,"cost":22,'+
                        '"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Challenge 6-Test scheduling logic

WarehouseSyncScheduleTest

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

         CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
=: jobId];
        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