Apex Triggers :

Get Started with Apex Triggers -

CHALLENGE :

- Soloution : **`AccountAddressTrigger`**

```
trigger AccountAddressTrigger on Account (before insert , before update) {

    for(Account a:Trigger.New){
        if(a.Match_Billing_Address__c == true){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }

}
```

Bulk Apex Triggers -

Soloution : 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

                                  WhatId = opp.Id));

        }
    }
    if (taskList.size() > 0){
        insert taskList;
    }
```

```
14 }
```

Apex Testing :

Get Started with Apex Unit Tests -

CHALLENGE :

- Soloution :

Apex class - 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.  Otherwise use the end of the month
6          if(DateWithin30Days(date1,date2)) {
7              return date2;
8          } else {
9              return SetEndOfMonthDate(date1);
10         }
11     }
12
13     //method to check if date2 is within the next 30 days of
   date1
14     @TestVisible private static Boolean DateWithin30Days(Date
   date1, Date date2) {
15         //check for date2 being in the past
16     if( date2 < date1) { return false; }
17
18     //check that date2 is within (>=) 30 days of date1
19     Date date30Days = date1.addDays(30); //create a date 30 days
   away from date1
20         if( date2 >= date30Days ) { return false; }
21         else { return true; }
```

```
22    }
23
24    //method to return the end of the month of a given date
25    @TestVisible private static Date SetEndOfMonthDate(Date
   date1) {
26            Integer totalDays = Date.daysInMonth(date1.year(),
   date1.month());
27            Date lastDay = Date.newInstance(date1.year(),
   date1.month(), totalDays);
28            return lastDay;
29    }
30
31 }
```

Testing : TestVerifyDate

```
1  @isTest
2
3  public class TestVerifyDate {
4
5      @isTest static void Test_CheckDates_case1() {
6            Date D = VerifyDate.CheckDates(
   date.parse('01/01/2020'), date.parse('01/05/2020')) ;
7            System.assertEquals(date.parse('01/05/2020'), D)
   ;
8
9      }
10     @isTest static void Test_CheckDates_case2() {
11         Date D =
   VerifyDate.CheckDates(date.parse('01/01/2020'),
   date.parse('05/05/2020') ) ;
12         System.assertEquals(date.parse('01/31/2020 '), D )
   ;
13     }
14  @isTest static void Test_DateWithin30Days_case1() {
15        Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
```

```
   date.parse('12/30/2019')) ;
16        System.assertEquals(false, flag ) ;
17    }
18
19    @isTest static void Test_DateWithin30Days_case2() {
20        Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
   date.parse('02/02/2019')) ;
21        System.assertEquals(false, flag ) ;
22    }
23    @isTest static void Test_DateWithin30Days_case3() {
24        Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
   date.parse('01/15/2019')) ;
25        System.assertEquals(true, flag ) ;
26    }
27    @isTest static void Test_SetEndOfMonthDate() {
28         Date returndate =
   VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020')) ;
29    }
30 }
```

Test Apex Triggers :

:

Apex class - 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+'" is not

7             }
8              }
9   }
```

Testing : **TestRestrictContactByName**

```
1  @isTest
2  public class TestRestrictContactByName {
3
4      @isTest static void Test_insertupdateContact(){
5          Contact cnt = new Contact();
6          cnt.LastName = 'INVALIDNAME';
7
8          Test.startTest();
9          Database.SaveResult result =
   Database.insert(cnt,false);
10
11         System.assert(!result.isSuccess());
12         System.assert(result.getErrors().size() > 0);
13         System.assertEquals('The Last Name "INVALIDNAME"

   ,result.getErrors()[0].getMessage());
14     }
15
16 }
```

Create Test Data for Apex Tests :

CHALLENGE :

- Soloution :

Apex class - RandomContactFactory

```apex
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt,string lastname){
        List <Contact>contacts = new List <Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test' +i,LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }


```

Asynchronous Apex :

Use Future Methods -

CHALLENGE :

- Soloution :

Apex class - AccountProcessor

```apex
public without sharing class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds){
        List <Account> accounts = [SELECT Id,(SELECT Id FROM Contacts)  FROM Account WHERE Id IN : accountIds];

                                    for (Account acc : accounts){

acc.Number_Of_Contacts__c = acc.Contacts.size();

                                    }
                                    update accounts;
    }
```

```
13
14 }
```

Testing – **AccountProcessorTest**

```
1  @isTest
2  private class AccountProcessorTest {
3
4      @isTest
5      private static void countContactsTest(){
6
7          // Load Test Data
8                List<Account> accounts = new
   List<Account>();
9          for (Integer i=0; i<300; i++){
10             accounts.add(new Account (Name = 'Test

11         }
12         insert accounts;
13
14          List<Contact> contacts = new List<Contact>();
15          List<Id> accountIds = new List<Id>();
16         for (Account acc : accounts){
17             contacts.add(new Contact(FirstName = acc.Name
   , LastName = 'TestContact',AccountId = acc.Id));
18             accountIds.add(acc.Id);
19         }
20         insert contacts;
21
22         // Do The Test
23         Test.startTest();
24         AccountProcessor.countContacts(accountIds);
25         Test.stopTest();
26
27         // Check result
28         List<Account> accs = [SELECT Id ,
   Number_of_Contacts__c FROM Account];
29             for(Account acc : accs){
30                 System.assertEquals(1 ,
   acc.Number_Of_Contacts__c , 'ERROR : AT Least 1 Account
```

```
            }
    }
}
```

Use Batch Apex :

CHALLENGE :

- Soloution :

Apex class - LeadProcessor

```
/*public without sharing class LeadProcessor implements Database.Batchable<sObject>,Datebase.Stateful {

    public Integer recordCount = 0;
    public Database.QueryLocator start (Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name FROM Lead]);
    }
    public void execute (Databse.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);
    }
}*/
global class LeadProcessor implements Database.Batchable<sObject> {
       global Integer count =0 ;

    global Database.QueryLocator
```

```
   start(Database.BatchableContext bc ) {
22         return Database.getQueryLocator('SELECT ID,

23     }
24
25     global void execute (Database.BatchableContext bc,
   List<Lead> L_list ) {
26         List<Lead> L_list_new = new List<lead>( ) ;
27         for(lead L:L_list) {
28             L.leadsource ='Dreamforce' ;
29             L_list_new.add(L) ;
30             count += 1 ;
31         }
32         update L_list_new ;
33     }
34     global void finish(Database.BatchableContext bc ) {
35         system.debug('count = ' +count ) ;
36     }
37
38 }
```

Testing : LeadProcessorTest

```
1 @isTest
2 public class LeadProcessorTest {
3
4         @isTest
5     public static void testing( ) {
6         List<lead> L_list = new List<lead>
   ( ) ;
7
8         for(Integer i=0; i<200; i++ ) {
9             Lead L = new lead( ) ;
10             L.LastName = 'name' +i ;
```

```
11              L.Company = 'Company' ;
12              L.Status = 'Random Status' ;
13              L_list.add(L) ;
14          }
15          insert L_list ;
16
17          Test.startTest() ;
18          LeadProcessor Ip = new
  LeadProcessor( );
19          Id batchId =
  Database.executeBatch(Ip) ;
20              Test.stopTest() ;
21      }
22
23 }
```

Control Processes with Queueable Apex :

CHALLENGE :

- Soloution :

Apex class - 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 ;
19                  lstContact.add( cont );
20          }
21
22          if(lstContact.size() >0 )
23          {
24              insert lstContact;
25          }
26
27     }
28
29 }
```

Testing :

```
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));
```

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

Schedule Jobs Using the Apex Scheduler :

Apex class: DailyLeadProcessor

```
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 ;
        }
    }
}
```

Testing : `DailyLeadProcessorTest`

```
@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          }
10         insert lList ;
11
12         Test.startTest() ;
13         String jobId = System.schedule('DailyLeadProcessor' ,
   CRON_EXP , new DailyLeadProcessor( )) ;
14     }
15 }
```

Apex Integration Services :

Apex REST Callouts -

Apex class : 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-

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,
11 Object>)JSON.deserializeUntyped(res.getBody());
12 animal = (Map<String, Object>) results.get('animal');
```

```
13 }
14 return (String)animal.get('name');
15 }
16  }
```

Testing -AnimalLocatorTest

```
1  AnimalLocatorTest
2  @isTest
3  private class AnimalLocatorTest{
4  @isTest static void AnimalLocatorMock1() {
5  Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
6  string result = AnimalLocator.getAnimalNameById(3);
7  String expectedResult = 'chicken';
8  System.assertEquals(result,expectedResult );
9  }
10 }
```

Apex SOAP Callouts :

Apex Class : ParkLocator

```
1  public class ParkLocator {
2  public static string[] country(string theCountry) {
3  ParkService.ParksImplPort parkSvc = new
   ParkService.ParksImplPort(); // remove
4  space
5  return parkSvc.byCountry(theCountry);
6  }
7  }
```

Testing - 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

8  'Yosemite'};
9  System.assertEquals(parks, result);
10 }
11 }
```

Apex Web Services

Apex Class - AccountManager

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

Testing -

```
1  @IsTest
2  private class AccountManagerTest {
3  @isTest static void testGetContactsByAccountId(){ Id recordId
   = createTestRecord();
4  RestRequest request = new RestRequest(); request.requestUri =
5  'https://yourInstance.my.salesforce.com/services/apexrest/Acco

6  request.httpMethod = 'GET';
7  RestContext.request = request;
```

```
8  Account thisAccount = AccountManager.getAccount();
   System.assert(thisAccount != null); System.assertEquals('Test

9  }
10 static Id createTestRecord(){
11 Account accountTest = new Account(
12 Name ='Test record'); insert accountTest;
13 Contact contactTest = new Contact( FirstName='John',
14 LastName = 'Doe',
15 AccountId = accountTest.Id
16 );
17 insert contactTest;
18 return accountTest.Id;
19 } }
```

Apex Specialist :

Automate Record Creation -

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
```

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

7                       validIds.add(c.Id);
8                   }
9               }
10          }
11
12          //When an existing maintenance request of type Repair
    or Routine Maintenance is closed,
13          //create a new maintenance request for a future
    routine checkup.
14          if (!validIds.isEmpty()){
15              Map<Id,Case> closedCases = new
    Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
    Equipment__r.Maintenance_Cycle__c,
16
    (SELECT Id,Equipment__c,Quantity__c FROM
    Equipment_Maintenance_Items__r)
17                                                          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>();
```

```
            for(Case cc : closedCases.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                    Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    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));
                }

                newCases.add(nc);
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedList = new List<Equipment_Maintenance_Item__c>();
            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;
        }
    }
}
```

Sychronize Salseforce Data With an External System :

```
1  public with sharing class WarehouseCalloutService implements
   Queueable {
2      private static final String WAREHOUSE_URL = 'https://th-

3
4      //Write a class that makes a REST callout to an external
   warehouse system to get a list of equipment that needs to be
   updated.
5      //The callout's JSON response returns the equipment
   records that you upsert in Salesforce.
6
7      @future(callout=true)
8      public static void runWarehouseEquipmentSync(){
9          System.debug('go into runWarehouseEquipmentSync');
10         Http http = new Http();
11         HttpRequest request = new HttpRequest();
12
13         request.setEndpoint(WAREHOUSE_URL);
14         request.setMethod('GET');
15         HttpResponse response = http.send(request);
16
17         List<Product2> product2List = new List<Product2>();
18         System.debug(response.getStatusCode());
19         if (response.getStatusCode() == 200){
20             List<Object> jsonResponse =
   (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 which equipment records to update within
   Salesforce
25             for (Object jR : jsonResponse){
26                 Map<String,Object> mapJson =
   (Map<String,Object>)jR;
27                 Product2 product2 = new Product2();
28                 //replacement part (always true),
29                 product2.Replacement_Part__c = (Boolean)
   mapJson.get('replacement');
30                 //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

            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

Schedule Synchroniztion :

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
```

```
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

Test automation logic :

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing

        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing

                                          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='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }

    // 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
```

```
                                                    from
Equipment_Maintenance_Item__c
                                                    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

        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];

```

```
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(createEquipmentMaintenanceIte

136         }
137         insert equipmentMaintenanceItemList;
138
```

```
        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                                    from case
                                    where status ='New'];


        list<Equipment_Maintenance_Item__c> workParts =
  [select id
                                                          from
  Equipment_Maintenance_Item__c
  where Maintenance_Request__c in: oldCaseIds];

        system.assert(newCase.size() == 300);

        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

Test callout logic :

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');
```

```
8
   response.setBody('[{"_id":"55d66226726b611100aaf741","replacem
9          response.setStatusCode(200);
10
11         return response;
12     }
13 }
```

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 }
```

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

        Test.stopTest();
    }
}
```

WarehouseSyncSchedule

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

WarehouseCalloutServiceMock -

```
 @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
   response.setBody('[{"_id":"55d66226726b611100aaf741","replacem

9          response.setStatusCode(200);
10
11         return response;
12     }
13 }
```
