## CODES FOR HANDS-ON CHALLENGES IN SALESFORCE SELF LEARNING:

## APEX TRIGGERS MODULE

### Get started with Apex Triggers

*1) AccountAddressTrigger.apxt*

```
 trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

### Bulk Apex Triggers Challenge

*2) ClosedOpportunityTrigger.apxt*

```
2)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 MODULE

### Get started with Apex Unit Tests

*1) VerifyDate.apxc*

```
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
  @TestVisible 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
  @TestVisible 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;
  }

}
```

*2) TestVerifyDate.apxc*

```
@isTest
private class TestVerifyDate{
```

```
3    @isTest static void Test_CheckDates_case1(){
4    Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
5          System.assertEquals(date.parse('01/05/2020'), D);
6    }
7    @isTest static void Test_CheckDates_case2(){
8          Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
   date.parse('05/05/2020'));
9          System.assertEquals(date.parse('01/31/2020'), D);
10   }
11   @isTest static void Test_Datekithin30Days_case1(){
12         Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
   date.parse('12/30/2019'));
13         System.assertEquals(false, flag);
14   }
15   @isTest static void Test_Datewithin30Days_case2(){
16         Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
   date.parse('02/02/2019'));
17          System.assertEquals(false, flag);
18      }
19      @isTest static void Test_Datewithin30Days_case3(){
20         Boolean flag =
   VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
   date.parse('01/15/2020'));
21          System.assertEquals(true, flag);
22      }
23      @isTest static void Test_SetEndOfMonthDate(){
24          Date returndate =
   VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
25      }
26 }
27
```

**Test Apex Triggers**

*3) RestrictContactByName.apxt*

```
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 allowed for DML');
7            }
8
9    }
10 }
```

*4) RestrictContactByName.apxc*

```
1  @isTest
2  public class TestRestrictContactByName {
3      @isTest static void Test_insertupdateContact(){
4          Contact cnt = new Contact();
5          cnt.LastName = 'INVALIDNAME';
6
7          Test.startTest();
8          Database.SaveResult result = Database.insert(cnt, 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     }
15 }
```

**Create Test Data for Apex Tests**

*5) RandomContactFactory.apxc*

```
1  public class RandomContactFactory {
2      public static List<Contact> generateRandomContacts(Integer
   num, String lastname){
```

```
3           List<Contact> contactList = new List<Contact>();
4           for(Integer i = 1; i<=num; i++){
5               Contact ct = new Contact(FirstName = 'Test '+i,
    LastName = lastName);
6               contactList.add(ct);
7           }
8           return contactList;
9       }
10  }
```

**ASYNCHRONOUS APEX MODULE**

**Use future methods**

*1) AccountProcessor.apxc*

```
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];
        // process account records to do awesome stuff
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;

    }

}
```

*2) AccountProcessorTest.apxc*

```
@IsTest
private class AccountProcessorTest{
  @IsTest
  private static void testCountContacts() {
      Account newAccount = new Account(Name='Test Account');
      insert newAccount;

      Contact newContact1 = new Contact(FirstName='John',
                                        LastName='Doe',
                                        AccountId=newAccount.Id);
      insert newContact1;
      Contact newContact2 = new Contact(FirstName='Jane',
                                        LastName='Doe',
                                        AccountId=newAccount.Id);
```

```
      insert newContact2;
      List<Id> accountIds = new List<Id>();
    accountIds.add(newAccount.Id);
    AccountProcessor.countContacts(accountIds);

      Test.startTest();
      AccountProcessor.countContacts(accountIds);

      Test.stopTest();

  }
}

```

**Use Batch Apex**

*1) LeadProcessor.apxc*

```
global class LeadProcessor implements Database.Batchable<sObject>
{
    global Integer count = 0;
    global Database.queryLocator start(Database.BatchableContext
bc){
        return Database.getQueryLocator('SELECT ID, LeadSource

    }

    global void execute (Database.BatchableContext bc, List<Lead>
L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;

        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
```

```
        system.debug('count = '+ count);
    }
}
```

*2) LeadProcessorTest.apxc*

```
@isTest
public class LeadProcessorTest {

  @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0; i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.company = 'company';
            L.status = 'Random Status';
            L_list.add(L);
        }
        insert l_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

**Control Processes with Queuable Apex**

*1) AddPrimaryContact.apxc*

```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
```

```
    }
    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select
FirstName, LastName, Id from contacts)
                                  from Account where BillingState
= :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();
        for(Account acc:accounts){
            contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }
        if(primaryContacts.size()>0){
            insert primaryContacts;
        }
    }

}
```

*2) AddPrimaryContactTest.apxc*

```
@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0; i<50; i++){
            testAccounts.add(new Account(Name = 'Account

        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name = 'Account

        }
        insert testAccounts;
        Contact testContact = new Contact(FirstName = 'John',
LastName = 'Doe');
        insert testContact;

        AddPrimaryContact addit = new
```

```
    addPrimaryContact(testContact, 'CA');
17
18          Test.startTest();
19          system.enqueueJob(addit);
20          Test.stopTest();
21          System.assertEquals(50,[Select count() from Contact where
    accountId in (Select Id from Account where BillingState='CA')]);
22      }
23  }
```

**Schedule Jobs Using the Apex Scheduler**

*1) DailyLeadProcessor.apxc*

```
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;
    }

}
```

*2) DailyLeadProcessorTest.apxc*

```
@isTest
public class DailyLeadProcessorTest {
    private 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='Connock', Company='Salesforce'));
            }
            else{
                leads.add(new Lead(LastName='Connock', Company='Salesforce', LeadSource='Other'));
            }
        }
        insert leads;

        Test.startTest();
```

```
        String jobId = System.schedule('Process Leads', CRON_EXP, new DailyLeadProcessor());
        Test.stopTest();

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At


        List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger WHERE Id= :jobId];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);

    }

}
```

**Apex REST Callouts**

*1) DailyLeadProcessor.apxc*

```
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;
    }

}
```

*2) DailyLeadProcessorTest.apxc*

```
@isTest
```

```
public class DailyLeadProcessorTest {
    private 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='Connock',
Company='Salesforce'));
            }
            else{
                leads.add(new Lead(LastName='Connock',
Company='Salesforce', LeadSource='Other'));
            }
        }
        insert leads;

        Test.startTest();
        String jobId = System.schedule('Process Leads', CRON_EXP,
new DailyLeadProcessor());
        Test.stopTest();

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM
Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At


        List<CronTrigger> cts = [SELECT Id, TimesTriggered,
NextFireTime FROM CronTrigger WHERE Id= :jobId];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);

    }

}
```

*3) AnimalLocatorMock.apxc*

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
```

```
3
4       global HttpResponse respond(HttpRequest request){
5           HttpResponse response = new HttpResponse();
6           response.setHeader('contentType', 'application/json');
7
    response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants

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

**Apex SOAP Callouts**

*1) ParkService.apxc*

```
1   //Generated by wsdl2apex
2
3   public class ParkService {
4       public class byCountryResponse {
5           public String[] return_x;
6           private String[] return_x_type_info = new
    String[]{'return','http://parks.services/',null,'0','-

7           private String[] apex_schema_type_info = new
    String[]{'http://parks.services/','false','false'};
8           private String[] field_order_type_info = new
    String[]{'return_x'};
9       }
10      public class byCountry {
11          public String arg0;
12          private String[] arg0_type_info = new
    String[]{'arg0','http://parks.services/',null,'0','1','false'};
13          private String[] apex_schema_type_info = new
    String[]{'http://parks.services/','false','false'};
14          private String[] field_order_type_info = new
    String[]{'arg0'};
15      }
16      public class ParksImplPort {
```

```
        public String endpoint_x = 'https://th-apex-soap-

        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new
ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse>
response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

*2) ParkLocator.apxc*

```
public class ParkLocator {
```

```
2
3       public static List < String > country(String country){
4           ParkService.ParksImplPort prkSvc = new
    ParkService.ParksImplPort();
5           return prkSvc.byCountry(country);
6       }
7
8   }
```

*3) ParkLocatorTest.apxc*

```
@isTest
public class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class, new
ParkServiceMock());
        String country = 'United States';
        List<String> expectedParks = new List<String>{'Yosemite',
'Sequoia', 'Crater Lake'};

System.assertEquals(expectedParks,ParkLocator.country(country));


    }

}
```

*4) ParkServiceMock.apxc*

```
@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>{'Yosemite','Sequoia','Crater Lake'};
         response.put('response_x', response_x);
        }
}
```

**Apex Web Services**

*1) AccountManager.apxc*

```
@RestResource(urlMapping = '/Accounts/*/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, Name from Contacts) from
Account where Id=:accountId Limit 1];
        return result;
    }
}
```

*2) AccountManagerTest.apxc*

```
@isTest
public class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestURI =
'https://yourInstance.my.salesforce.com/services/apexrest/Account

        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }
    static Id createTestRecord(){
        Account accountTest = new Account(
          Name ='Test record');
        insert accountTest;
        Contact contactTest = new Contact(
          FirstName='John',
          LastName = 'Doe',
          AccountId = accountTest.Id);
        insert contactTest;
```

```
22          return accountTest.Id;
23      }
24  }
```

**APEX SPECIALIST SUPERBADGE**

**2) Automate record creation**

*2.1) MaintaenanceRequesrHelper.apxc*

```
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);


                }
            }
        }

        if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            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) FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c, MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){
```

```
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
        }

            for(Case cc : closedCasesM.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 (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                } else {
                    nc.Date_Due__c = Date.today().addDays((Integer) cc.Equipment__r.maintenance_Cycle__c);
                }

                newCases.add(nc);
             }

           insert newCases;

           List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
           for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp : closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);
```

```
57
58                 }
59             }
60             insert ClonedWPs;
61         }
62     }
63 }
```

*2.2) MaintenanceRequest.apxt*

```
1  trigger MaintenanceRequest on Case (before update, after update)
   {
2
3      if(Trigger.isUpdate && Trigger.isAfter){
4
5          MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
   Trigger.OldMap);
6
7      }
8
9  }
```

**3) Synchronize Salesforce data with an external system**

*3.1) WarehouseCalloutService.apxc*

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


3
4      //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           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> warehouseEq = new List<Product2>();
17
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: replacement part
    (always true), cost, current inventory, lifespan, maintenance
    cycle, and warehouse SKU
23              //warehouse SKU will be external ID for identifying
    which equipment records to update within Salesforce
24              for (Object eq : jsonResponse){
25                  Map<String,Object> mapJson =
    (Map<String,Object>)eq;
26                  Product2 myEq = new Product2();
27                  myEq.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
28                  myEq.Name = (String) mapJson.get('name');
29                  myEq.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
30                  myEq.Lifespan_Months__c = (Integer)
    mapJson.get('lifespan');
31                  myEq.Cost__c = (Integer) mapJson.get('cost');
32                  myEq.Warehouse_SKU__c = (String)
    mapJson.get('sku');
33                  myEq.Current_Inventory__c = (Double)
    mapJson.get('quantity');
34                  myEq.ProductCode = (String) mapJson.get('_id');
35                  warehouseEq.add(myEq);
36              }
37
```

```
            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the

            }
        }
    }

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

After saving the code open execute anonymous window ( CTRl+E ) and run this method ,

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

**4) Schedule Synchronization using Apex code**

*4.1) WarehouseSyncSchedule.apxc*

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

**5) Test automation logic**

*1) MaintenanceRequestHelperTest.apxc*

*5.1) MaintenanceRequestHelperTest.apxc :-*

```
@istest
public with sharing class MaintenanceRequestHelperTest {
```

```
3
4       private static final string STATUS_NEW = 'New';
5       private static final string WORKING = 'Working';
6       private static final string CLOSED = 'Closed';
7       private static final string REPAIR = 'Repair';
8       private static final string REQUEST_ORIGIN = 'Web';
9       private static final string REQUEST_TYPE = 'Routine

10      private static final string REQUEST_SUBJECT = 'Testing

11
12      PRIVATE STATIC Vehicle__c createVehicle(){
13          Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
14          return Vehicle;
15      }
16
17      PRIVATE STATIC Product2 createEq(){
18          product2 equipment = new product2(name =
    'SuperEquipment',
19                                            lifespan_months__C = 10,
20                                            maintenance_cycle__C =
    10,
21                                            replacement_part__c =
    true);
22          return equipment;
23      }
24
25      PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id
    equipmentId){
26          case cs = new case(Type=REPAIR,
27                             Status=STATUS_NEW,
28                             Origin=REQUEST_ORIGIN,
29                             Subject=REQUEST_SUBJECT,
30                             Equipment__c=equipmentId,
31                             Vehicle__c=vehicleId);
32          return cs;
33      }
34
35      PRIVATE STATIC Equipment_Maintenance_Item__c
    createWorkPart(id equipmentId,id requestId){
```

```
36          Equipment_Maintenance_Item__c wp = new
    Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
37
    Maintenance_Request__c = requestId);
38          return wp;
39      }
40
41
42      @istest
43      private static void testMaintenanceRequestPositive(){
44          Vehicle__c vehicle = createVehicle();
45          insert vehicle;
46          id vehicleId = vehicle.Id;
47
48          Product2 equipment = createEq();
49          insert equipment;
50          id equipmentId = equipment.Id;
51
52          case somethingToUpdate =
    createMaintenanceRequest(vehicleId,equipmentId);
53          insert somethingToUpdate;
54
55          Equipment_Maintenance_Item__c workP =
    createWorkPart(equipmentId,somethingToUpdate.id);
56          insert workP;
57
58          test.startTest();
59          somethingToUpdate.status = CLOSED;
60          update somethingToUpdate;
61          test.stopTest();
62
63          Case newReq = [Select id, subject, type, Equipment__c,
    Date_Reported__c, Vehicle__c, Date_Due__c
64                         from case
65                         where status =:STATUS_NEW];
66
67          Equipment_Maintenance_Item__c workPart = [select id
68                                                    from
    Equipment_Maintenance_Item__c
69                                                    where
```

```
   Maintenance_Request__c =:newReq.Id];
70
71          system.assert(workPart != null);
72          system.assert(newReq.Subject != null);
73          system.assertEquals(newReq.Type, REQUEST_TYPE);
74          SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
75          SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
76          SYSTEM.assertEquals(newReq.Date_Reported__c,
   system.today());
77      }
78
79      @istest
80      private static void testMaintenanceRequestNegative(){
81          Vehicle__C vehicle = createVehicle();
82          insert vehicle;
83          id vehicleId = vehicle.Id;
84
85          product2 equipment = createEq();
86          insert equipment;
87          id equipmentId = equipment.Id;
88
89          case emptyReq =
   createMaintenanceRequest(vehicleId,equipmentId);
90          insert emptyReq;
91
92          Equipment_Maintenance_Item__c workP =
   createWorkPart(equipmentId, emptyReq.Id);
93          insert workP;
94
95          test.startTest();
96          emptyReq.Status = WORKING;
97          update emptyReq;
98          test.stopTest();
99
100          list<case> allRequest = [select id
101                                   from case];
102
103          Equipment_Maintenance_Item__c workPart = [select id
104                                                    from
   Equipment_Maintenance_Item__c
105                                                    where
```

```
   Maintenance_Request__c = :emptyReq.Id];
106
107         system.assert(workPart != null);
108         system.assert(allRequest.size() == 1);
109     }
110
111     @istest
112     private static void testMaintenanceRequestBulk(){
113         list<Vehicle__C> vehicleList = new list<Vehicle__C>();
114         list<Product2> equipmentList = new list<Product2>();
115         list<Equipment_Maintenance_Item__c> workPartList = new
   list<Equipment_Maintenance_Item__c>();
116         list<case> requestList = new list<case>();
117         list<id> oldRequestIds = new list<id>();
118
119         for(integer i = 0; i < 300; i++){
120            vehicleList.add(createVehicle());
121             equipmentList.add(createEq());
122         }
123         insert vehicleList;
124         insert equipmentList;
125
126         for(integer i = 0; i < 300; i++){
127
   requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
   equipmentList.get(i).id));
128         }
129         insert requestList;
130
131         for(integer i = 0; i < 300; i++){
132
   workPartList.add(createWorkPart(equipmentList.get(i).id,
   requestList.get(i).id));
133         }
134         insert workPartList;
135
136         test.startTest();
137         for(case req : requestList){
138             req.Status = CLOSED;
139             oldRequestIds.add(req.Id);
```

```
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

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

        system.assert(allRequests.size() == 300);
    }
}
```

*5.2) MaintenanceRequestHelper.apxc*

```
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 Maintenance'){
                    validIds.add(c.Id);


                }
            }
        }
```

```
15
16          if (!validIds.isEmpty()){
17              List<Case> newCases = new List<Case>();
18              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)
19
    FROM Case WHERE Id IN :validIds]);
20              Map<Id,Decimal> maintenanceCycles = new
    Map<ID,Decimal>();
21              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];
22
23          for (AggregateResult ar : results){
24              maintenanceCycles.put((Id)
    ar.get('Maintenance_Request__c'), (Decimal)
    ar.get('cycle'));
25          }
26
27              for(Case cc : closedCasesM.values()){
28                  Case nc = new Case (
29                      ParentId = cc.Id,
30                  Status = 'New',
31                      Subject = 'Routine Maintenance',
32                      Type = 'Routine Maintenance',
33                      Vehicle__c = cc.Vehicle__c,
34                      Equipment__c =cc.Equipment__c,
35                      Origin = 'Web',
36                      Date_Reported__c = Date.Today()
37
38                  );
```

```
39
40                 If (maintenanceCycles.containskey(cc.Id)){
41                     nc.Date_Due__c =
   Date.today().addDays((Integer)
   maintenanceCycles.get(cc.Id));
42                 }
43
44                 newCases.add(nc);
45             }
46
47             insert newCases;
48
49             List<Equipment_Maintenance_Item__c> clonedWPs =
   new List<Equipment_Maintenance_Item__c>();
50             for (Case nc : newCases){
51                 for (Equipment_Maintenance_Item__c wp :
   closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__

52                     Equipment_Maintenance_Item__c wpClone =
   wp.clone();
53                     wpClone.Maintenance_Request__c = nc.Id;
54                     ClonedWPs.add(wpClone);
55
56                 }
57             }
58             insert ClonedWPs;
59         }
60     }
61 }
```

*5.3) MaintenanceRequest.apxt*

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

```
    Trigger.OldMap);
4       }
5   }
```

### 6) Test Callout Logic:

*6.1) WarehouseCalloutService.apxc*

```
1   public with sharing class WarehouseCalloutService {
2
3       private static final String WAREHOUSE_URL = 'https://th-
4
5       //@future(callout=true)
6       public static void runWarehouseEquipmentSync(){
7
8           Http http = new Http();
9           HttpRequest request = new HttpRequest();
10
11          request.setEndpoint(WAREHOUSE_URL);
12          request.setMethod('GET');
13          HttpResponse response = http.send(request);
14
15
16          List<Product2> warehouseEq = new List<Product2>();
17
18          if (response.getStatusCode() == 200){
19              List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
20              System.debug(response.getBody());
21
22              for (Object eq : jsonResponse){
23                  Map<String,Object> mapJson =
    (Map<String,Object>)eq;
24                  Product2 myEq = new Product2();
25                  myEq.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
26                  myEq.Name = (String) mapJson.get('name');
27                  myEq.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
28                  myEq.Lifespan_Months__c = (Integer)
```

```
   mapJson.get('lifespan');
29                 myEq.Cost__c = (Decimal) mapJson.get('lifespan');
30                 myEq.Warehouse_SKU__c = (String)
   mapJson.get('sku');
31                 myEq.Current_Inventory__c = (Double)
   mapJson.get('quantity');
32                 warehouseEq.add(myEq);
33             }
34
35             if (warehouseEq.size() > 0){
36                 upsert warehouseEq;
37                 System.debug('Your equipment was synced with the

38                 System.debug(warehouseEq);
39             }
40
41         }
42     }
43 }
```

*6.2) WarehouseCalloutServiceMock.apxc*

```
1  @isTest
2  global class WarehouseCalloutServiceMock implements
   HttpCalloutMock {
3      // implement http mock callout
4      global static HttpResponse respond(HttpRequest request){
5
6          System.assertEquals('https://th-superbadge-
   ));
7          System.assertEquals('GET', request.getMethod());
8
9          // Create a fake response
10         HttpResponse response = new HttpResponse();
11         response.setHeader('Content-Type', 'application/json');
12
   response.setBody('[{"_id":"55d66226726b611100aaf741","replacement
```

```
        response.setStatusCode(200);
        return response;
    }
}
```

*6.3) WarehouseCalloutServiceTest.apxc*

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new
WarehouseCalloutServiceMock());

WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM
Product2]);
    }
}
```

### 7) Test scheduling logic

*7.1) WarehouseCalloutServiceMock.apxc*

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest
request){
        System.assertEquals('https://th-superbadge-
));
        System.assertEquals('GET', request.getMethod());
```

```
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","repla




        response.setStatusCode(200);
        return response;
    }
}
```

*7.2) WarehouseSyncSchedule.apxc*

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

*7.3) WarehouseSyncScheduleTest.apxc*

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule

        Test.stopTest();
        //Contains schedule information for a scheduled job.
```

```
   CronTrigger is similar to a cron job on UNIX systems.
11         // This object is available in API version 17.0 and
   later.
12         CronTrigger a=[SELECT Id FROM CronTrigger where
   NextFireTime > today];
13         System.assertEquals(jobID, a.Id,'Schedule ');
14
15
16     }
17 }
```