APEX TRIGGER

1. GET STARTED WITH APEX TRIGGERS

➤ AccountAddressTrigger.apxt

```
1  trigger AccountAddressTrigger on Account (before insert, before update)
   {
2      for (Account a : Trigger.New) {
3          if (a.Match_Billing_Address__c == true && a.BillingPostalCode !=
   null) {
4              a.ShippingPostalCode = a.BillingPostalCode;
5          }
6      }
7  }
```

2. BULK APEX TRIGGERS:

➤ClosedOpportunityTrigger.apxt

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

APEX TESTING

1. GET STARTED WITH APEX UNIT TEST:

➤ • VerifyDate.apxc

```
1  public class VerifyDate {
2      public static Date CheckDates(Date date1, Date date2) {
3          if(DateWithin30Days(date1,date2)) {
4              return date2;
5          } else {
6              return SetEndOfMonthDate(date1);
7          }
8      }
9      private static Boolean DateWithin30Days(Date date1, Date date2) {
10     if( date2 < date1) { return false; }
11     Date date30Days = date1.addDays(30);
12         if( date2 >= date30Days ) { return false; }
13         else { return true; }
14     }
15     private static Date SetEndOfMonthDate(Date date1) {
16         Integer totalDays = Date.daysInMonth(date1.year(),
   date1.month());
17         Date lastDay = Date.newInstance(date1.year(), date1.month(),
   totalDays);
18         return lastDay;
19     }
20 }
```

➤ •TestVerifyDate.apxc

```
1  @isTest
2  private class TestVerifyDate {
3      @isTest static void testDate2within30daysofDate1() {
4          Date date1 = date.newInstance(2018, 03, 20);
5          Date date2 = date.newInstance(2018, 04, 11);
6          Date resultDate = VerifyDate.CheckDates(date1,date2);
7          Date testDate = Date.newInstance(2018, 04, 11);
8          System.assertEquals(testDate,resultDate);
9      }
10     @isTest static void testDate2beforeDate1() {
11         Date date1 = date.newInstance(2018, 03, 20);
12         Date date2 = date.newInstance(2018, 02, 11);
```

```
13          Date resultDate = VerifyDate.CheckDates(date1,date2);
14          Date testDate = Date.newInstance(2018, 02, 11);
15          System.assertNotEquals(testDate, resultDate);
16      }
17      @isTest static void testDate2outside30daysofDate1() {
18          Date date1 = date.newInstance(2018, 03, 20);
19          Date date2 = date.newInstance(2018, 04, 25);
20          Date resultDate = VerifyDate.CheckDates(date1,date2);
21          Date testDate = Date.newInstance(2018, 03, 31);
22          System.assertEquals(testDate,resultDate);
23      }
24 }
```

2. TEST APEX TRIGGERS

➤● RestrictContactByName.apxt

```
1  trigger RestrictContactByName on Contact (before insert, before update)
   {
2      For (Contact c : Trigger.New) {
3          if(c.LastName == 'INVALIDNAME') {    //invalidname is invalid
4              c.AddError('The Last Name "'+c.LastName+'" is not allowed


5          }
6      }
7  }
```

3.CREATE TEST DATA FOR APEX TESTS:

● RandomContactFactory.apxc

```
1  public class RandomContactFactory {
2      public static list<contact> generateRandomContacts(integer n, string
   m) {
3          list<contact> con = new list<contact>();
4          for(integer i=1; i<n+1; i++) {
5              contact c = new contact(firstname='test'+i,lastname=m);
6              con.add(c);
```

```
7          }
8           return con;
9       }
10 }
```

ASYNCRONOUS APEX

1. USE FUTURE METHODS

➤ ● AccountProcessor.apxc

```
1  public class AccountProcessor {
2      @future
3      public static void countContacts(List<Id> accountId_lst) {
4          Map<Id,Integer> account_cno = new Map<Id,Integer>();
5          List<account> account_lst_all = new List<account>([select id,
   (select id from contacts) from account]);
6          for(account a:account_lst_all) {
7              account_cno.put(a.id,a.contacts.size()); //populate the map
8          }
9          List<account> account_lst = new List<account>(); // list of
   account that we will upsert
10         for(Id accountId : accountId_lst) {
11             if(account_cno.containsKey(accountId)) {
12                 account acc = new account();
13                 acc.Id = accountId;
14                 acc.Number_of_Contacts__c = account_cno.get(accountId);
15                 account_lst.add(acc);
16             }
17         }
18         upsert account_lst;
19     }
20 }
```

➤ ● AccountProcessorTest.apxc

```
1  @isTest
2  public class AccountProcessorTest {
3  @isTest
4      public static void testFunc() {
5          account acc = new account();
6          acc.name = 'MATW INC';
7          insert acc;
8          contact con = new contact();
9          con.lastname = 'Mann1';
10         con.AccountId = acc.Id;
11         insert con;
12         contact con1 = new contact();
13         con1.lastname = 'Mann2';
14         con1.AccountId = acc.Id;
15         insert con1;
16         List<Id> acc_list = new List<Id>();
17         acc_list.add(acc.Id);
18         Test.startTest();
19      AccountProcessor.countContacts(acc_list);
20         Test.stopTest();
21         List<account> acc1 = new List<account>([select
   Number_of_Contacts__c from account where id = :acc.id]);
22         system.assertEquals(2,acc1[0].Number_of_Contacts__c);
23     }
24 }
```

2. USE BATCH APEX:

➤ ● **LeadProcessor.apxc**

```
1  global class LeadProcessor implements Database.Batchable<sObject> {
2      global Database.QueryLocator start(Database.BatchableContext bc) {
3          return Database.getQueryLocator(
4              'SELECT Id, LeadSource FROM Lead'
5          );
6      }
7      global void execute(Database.BatchableContext bc, List<Lead> leads)
   {
8          System.debug(leads.size());
```

```
 9              for(Lead lead : leads){
10                  lead.LeadSource = 'Dreamforce';
11              }
12          update leads;
13      }
14      global void finish(Database.BatchableContext bc){
15      }
16 }
```

➤ ● LeadProcessorTest.apxc

```
 1  @isTest
 2  private class LeadProcessorTest {
 3      @testSetup
 4      static void setup() {
 5          List<Lead> leads = new List<Lead>();
 6          // insert 10 leads
 7          for (Integer i=0;i<10;i++) {
 8              leads.add(new Lead(LastName='Lead '+i,
    Company='TestCompany'));
 9          }
10          insert leads;
11      }
12      static testmethod void test() {
13          Test.startTest();
14              LeadProcessor lp = new LeadProcessor();
15              Database.executeBatch(lp);
16          Test.stopTest();
17          // after the testing stops, assert records were updated properly
18          System.assertEquals(10, [SELECT count() FROM Lead where
    LeadSource = 'Dreamforce']);
19      }
20 }
```

3. CONTROL PROCESSES WITH QUEUEABLE APEX
➤ AddPrimaryContact.apxc

```
1  public class AddPrimaryContact implements Queueable {
2      public contact c;
3      public String state;
4
5      public AddPrimaryContact(Contact c, String state) {
6          this.c = c;
7          this.state = state;
8      }
9      public void execute(QueueableContext qc) {
10         system.debug('this.c = '+this.c+' this.state = '+this.state);
11         List<Account> acc_lst = new List<account>([select id, name,
   BillingState from account where account.BillingState = :this.state limit
   200]);
12         List<contact> c_lst = new List<contact>();
13         for(account a: acc_lst) {
14             contact c = new contact();
15             c = this.c.clone(false, false, false, false);
16             c.AccountId = a.Id;
17             c_lst.add(c);
18         }
19         insert c_lst;
20     }
21 }
```

➤ ● AddPrimaryContactTest.apxc

```
1  @IsTest
2  public class AddPrimaryContactTest {
3      @IsTest
4      public static void testing() {
5          List<account> acc_lst = new List<account>();
6          for (Integer i=0; i<50;i++) {
7              account a = new
   account(name=string.valueOf(i),billingstate='NY');
8              system.debug('account a = '+a);
9              acc_lst.add(a);
10         }
11         for (Integer i=0; i<50;i++) {
12             account a = new
   account(name=string.valueOf(50+i),billingstate='CA');
```

```
13              system.debug('account a = '+a);
14              acc_lst.add(a);
15          }
16          insert acc_lst;
17          Test.startTest();
18          contact c = new contact(lastname='alex');
19          AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
20          system.debug('apc = '+apc);
21          System.enqueueJob(apc);
22          Test.stopTest();
23          List<contact> c_lst = new List<contact>([select id from
   contact]);
24          Integer size = c_lst.size();
25          system.assertEquals(50, size);
26      }
27 }
```

4. SCHEDULE JOBS USING APEX SCHEDULER:

➤ ● DailyLeadProcessor.apxc

```
1  global class DailyLeadProcessor implements Schedulable{
2      global void execute(SchedulableContext ctx){
3          List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
   LeadSource = ''];
4          if(leads.size() > 0){
5              List<Lead> newLeads = new List<Lead>();
6              for(Lead lead : leads){
7                  lead.LeadSource = 'DreamForce';
8                  newLeads.add(lead);
9              }
10             update newLeads;
11         }
12     }
13 }
```

```
1  @isTest
2  private class DailyLeadProcessorTest{
3      public static String CRON_EXP = '0 0 0 2 6 ? 2022';
4      static testmethod void testScheduledJob(){
5          List<Lead> leads = new List<Lead>();
6          for(Integer i = 0; i < 200; i++){
7              Lead lead = new Lead(LastName = 'Test ' + i, LeadSource =
   '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');
8              leads.add(lead);
9          }
10         insert leads;
11         Test.startTest();
12         String jobId = System.schedule('Update LeadSource to

13       new DailyLeadProcessor());
14         Test.stopTest();
15     }
16 }
```

APEX INTEGRATION SERVICES

1. APEX REST CALLOUTS:

● AnimalLocator.apxc

```
1  public class AnimalLocator {
2      public class cls_animal {
3          public Integer id;
4          public String name;
5          public String eats;
6          public String says;
7      }
8  public class JSONOutput{
9      public cls_animal animal;
10 }
11
```

```
12      public static String getAnimalNameById (Integer id) {
13          Http http = new Http();
14          HttpRequest request = new HttpRequest();
15          request.setEndpoint('https://th-apex-http-

16          //request.setHeader('id', String.valueof(id)); -- cannot be used
   in this challenge :)
17          request.setMethod('GET');
18          HttpResponse response = http.send(request);
19          system.debug('response: ' + response.getBody());
20          jsonOutput results = (jsonOutput)
   JSON.deserialize(response.getBody(), jsonOutput.class);
21              system.debug('results= ' + results.animal.name);
22          return(results.animal.name);
23      }
24 }
```

➤ ● AnimalLocatorMock.apxc

```
1  @IsTest
2  global class AnimalLocatorMock implements HttpCalloutMock {
3      global HTTPresponse respond(HTTPrequest request) {
4          Httpresponse response = new Httpresponse();
5          response.setStatusCode(200);
6
   response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken

7          return response;
8      }
9  }
```

● AnimalLocatorTest.apxc

```
1  @IsTest
2  public class AnimalLocatorTest {
3      @isTest
4      public static void testAnimalLocator() {
5          Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
```

```
6            String s =  AnimalLocator.getAnimalNameById(1);
7            system.debug('string returned: ' + s);
8        }
9 }
```

2. APEX SOAP CALLOUTS:

➤   ● ParkService.apxc

```
1  public class ParkService {
2      public class byCountryResponse {
3          public String[] return_x;
4          private String[] return_x_type_info = new
   String[]{'return','http://parks.services/',null,'0','-1','false'};
5          private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
6          private String[] field_order_type_info = new
   String[]{'return_x'};
7      }
8      public class byCountry {
9          public String arg0;
10         private String[] arg0_type_info = new
   String[]{'arg0','http://parks.services/',null,'0','1','false'};
11         private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
12         private String[] field_order_type_info = new String[]{'arg0'};
13     }
14     public class ParksImplPort {
15         public String endpoint_x = 'https://th-apex-soap-

16         public Map<String,String> inputHttpHeaders_x;
17         public Map<String,String> outputHttpHeaders_x;
18         public String clientCertName_x;
19         public String clientCert_x;
20         public String clientCertPasswd_x;
21         public Integer timeout_x;
22         private String[] ns_map_type_info = new
   String[]{'http://parks.services/', 'ParkService'};
23         public String[] byCountry(String arg0) {
24             ParkService.byCountry request_x = new
   ParkService.byCountry();
```

```
25              request_x.arg0 = arg0;
26              ParkService.byCountryResponse response_x;
27              Map<String, ParkService.byCountryResponse> response_map_x =
   new Map<String, ParkService.byCountryResponse>();
28              response_map_x.put('response_x', response_x);
29              WebServiceCallout.invoke(
30                this,
31                request_x,
32                response_map_x,
33                new String[]{endpoint_x,
34                '',
35                'http://parks.services/',
36                'byCountry',
37                'http://parks.services/',
38                'byCountryResponse',
39                'ParkService.byCountryResponse'}
40              );
41              response_x = response_map_x.get('response_x');
42              return response_x.return_x;
43          }
44      }
45 }
```

ParkLocator.apxc

```
1  public class ParkLocator {
2      public static String[] country(String country){
3          ParkService.ParksImplPort parks = new
   ParkService.ParksImplPort();
4          String[] parksname = parks.byCountry(country);
5          return parksname;
6      }
7  }
```

➤ ● ParkLocatorTest.apxc

```
1  @isTest
2  private class ParkLocatorTest{
```

```
3       @isTest
4       static void testParkLocator() {
5           Test.setMock(WebServiceMock.class, new ParkServiceMock());
6           String[] arrayOfParks = ParkLocator.country('India');
7           System.assertEquals('Park1', arrayOfParks[0]);
8       }
9   }
```

3. APEX WEB SERVICES:

➤   • AccountManager.apxc

```
1   @RestResource(urlMapping='/Accounts/*/contacts')
2   global with sharing class AccountManager {
3       @HttpGet
4       global static account getAccount() {
5           RestRequest request = RestContext.request;
6           String accountId =
    request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
7               request.requestURI.lastIndexOf('/'));
8           List<Account> a = [select id, name, (select id, name from
    contacts) from account where id = :accountId];
9           List<contact> co = [select id, name from contact where
    account.id = :accountId];
10          system.debug('** a[0]= '+ a[0]);
11          return a[0];
12      }
13  }
```

➤   • AccountManagerTest.apxc

```
1   @Istest(SeeAllData=true)
2   public class AccountManagerTest {
3       @IsTest
```

```
 4      public static void testaccountmanager() {
 5          RestRequest request = new RestRequest();
 6          request.requestUri = 'https://mannharleen-dev-
    00016cw4tAAA/conta

 7          request.httpMethod = 'GET';
 8          RestContext.request = request;
 9              system.debug('test account result = '+
    AccountManager.getAccount());
10      }
11 }
```

APEX SPECIALIST SUPERBADGE

1. AUTOMATE RECORD CREATION:

● 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 }
```

● MaintenanceRequestHelper.apxc

```
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
```

```apex
7                        validIds.add(c.Id);
8                    }
9                }
10           }
11       if (!validIds.isEmpty()){
12            List<Case> newCases = new List<Case>();
13            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id,
  Vehicle__c, Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
  Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
14                                            FROM Case WHERE
  Id IN :validIds]);
15            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
16            AggregateResult[] results = [SELECT Maintenance_Request__c,
  MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
  Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds
  GROUP BY Maintenance_Request__c];
17
18       for (AggregateResult ar : results){
19            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'),
  (Decimal) ar.get('cycle'));
20       }
21
22           for(Case cc : closedCasesM.values()){
23               Case nc = new Case (
24                   ParentId = cc.Id,
25               Status = 'New',
26                   Subject = 'Routine Maintenance',
27                   Type = 'Routine Maintenance',
28                   Vehicle__c = cc.Vehicle__c,
29                   Equipment__c =cc.Equipment__c,
30                   Origin = 'Web',
31                   Date_Reported__c = Date.Today()
32               );
33
34               If (maintenanceCycles.containskey(cc.Id)){
35                   nc.Date_Due__c = Date.today().addDays((Integer)
  maintenanceCycles.get(cc.Id));
36               } else {
37                   nc.Date_Due__c = Date.today().addDays((Integer)
  cc.Equipment__r.maintenance_Cycle__c);
38               }
39               newCases.add(nc);
```

```
40              }
41          insert newCases;
42          List<Equipment_Maintenance_Item__c> clonedWPs = new
   List<Equipment_Maintenance_Item__c>();
43              for (Case nc : newCases){
44                  for (Equipment_Maintenance_Item__c wp :
   closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
45                      Equipment_Maintenance_Item__c wpClone = wp.clone();
46                      wpClone.Maintenance_Request__c = nc.Id;
47                      ClonedWPs.add(wpClone);
48                  }
49              }
50          insert ClonedWPs;
51      }
52  }
53 }
```

2. SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

● WarehouseCalloutService.apxc

```
1  public with sharing class WarehouseCalloutService implements Queueable {
2      private static final String WAREHOUSE_URL = 'https://th-superbadge-
3      @future(callout=true)
4      public static void runWarehouseEquipmentSync(){
5          Http http = new Http();
6          HttpRequest request = new HttpRequest();
7
8          request.setEndpoint(WAREHOUSE_URL);
9          request.setMethod('GET');
10         HttpResponse response = http.send(request);
11
12         List<Product2> warehouseEq = new List<Product2>();
13
14         if (response.getStatusCode() == 200){
15             List<Object> jsonResponse =
   (List<Object>)JSON.deserializeUntyped(response.getBody());
16             System.debug(response.getBody());
17
```

```
18              for (Object eq : jsonResponse){
19                  Map<String,Object> mapJson = (Map<String,Object>)eq;
20                  Product2 myEq = new Product2();
21                  myEq.Replacement_Part__c = (Boolean)
   mapJson.get('replacement');
22                  myEq.Name = (String) mapJson.get('name');
23                  myEq.Maintenance_Cycle__c = (Integer)
   mapJson.get('maintenanceperiod');
24                  myEq.Lifespan_Months__c = (Integer)
   mapJson.get('lifespan');
25                  myEq.Cost__c = (Integer) mapJson.get('cost');
26                  myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
27                  myEq.Current_Inventory__c = (Double)
   mapJson.get('quantity');
28                  myEq.ProductCode = (String) mapJson.get('_id');
29                  warehouseEq.add(myEq);
30              }
31          if (warehouseEq.size() > 0){
32              upsert warehouseEq;
33              System.debug('Your equipment was synced with the

34          }
35      }
36  }
37  public static void execute (QueueableContext context){
38      runWarehouseEquipmentSync();
39  }
40
41 }
```

3. SCHEDULE SYNCHRONIZATION USING APEX CODE:

- WarehouseSyncSchedule.ap

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

4. TEST AUTOMATION LOGIC:

- MaintenanceRequestHelperTest.apxc

```
1   @istest
2   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 Maintenance';
10      private static final string REQUEST_SUBJECT = 'Testing subject';
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);
140            }
141            update requestList;
142            test.stopTest();
143
144            list<case> allRequests = [select id
145                                      from case
146                                      where status =: STATUS_NEW];
147
```

```
148          list<Equipment_Maintenance_Item__c> workParts = [select id
149                                                      from
    Equipment_Maintenance_Item__c
150                                                      where
    Maintenance_Request__c in: oldRequestIds];
151
152          system.assert(allRequests.size() == 300);
153      }
154  }
```

● MaintenanceRequestHelper.apxc

```
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
5
6           For (Case c : updWorkOrders){
7               if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status
    == 'Closed'){
8                   if (c.Type == 'Repair' || c.Type == 'Routine

9                       validIds.add(c.Id);
10                  }
11              }
12          }
13          if (!validIds.isEmpty()){
14              List<Case> newCases = new List<Case>();
15              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)
16                                                      FROM Case WHERE
    Id IN :validIds]);
17              Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
18              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];
19
```

```
20          for (AggregateResult ar : results){
21              maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'),
   (Decimal) ar.get('cycle'));
22          }
23
24              for(Case cc : closedCasesM.values()){
25                  Case nc = new Case (
26                      ParentId = cc.Id,
27                  Status = 'New',
28                      Subject = 'Routine Maintenance',
29                      Type = 'Routine Maintenance',
30                      Vehicle__c = cc.Vehicle__c,
31                      Equipment__c =cc.Equipment__c,
32                      Origin = 'Web',
33                      Date_Reported__c = Date.Today()
34
35                  );
36
37                  If (maintenanceCycles.containskey(cc.Id)){
38                      nc.Date_Due__c = Date.today().addDays((Integer)
   maintenanceCycles.get(cc.Id));
39                  }
40
41                  newCases.add(nc);
42              }
43
44          insert newCases;
45
46          List<Equipment_Maintenance_Item__c> clonedWPs = new
   List<Equipment_Maintenance_Item__c>();
47          for (Case nc : newCases){
48              for (Equipment_Maintenance_Item__c wp :
   closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
49                  Equipment_Maintenance_Item__c wpClone = wp.clone();
50                  wpClone.Maintenance_Request__c = nc.Id;
51                  ClonedWPs.add(wpClone);
52
53              }
54          }
55          insert ClonedWPs;
56      }
57  }
```

```
58 }
```

● 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  }
```

5. TEST CALLOUT LOGIC:

● WarehouseCalloutService.apxc

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

● WarehouseCalloutServiceTest.apxc

```
1  @isTest
2
3  private class WarehouseCalloutServiceTest {
4      @isTest
5      static void testWareHouseCallout(){
6          Test.startTest();
7          // implement mock callout test here
8          Test.setMock(HTTPCalloutMock.class, new
    WarehouseCalloutServiceMock());
9          WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
10          Test.stopTest();
11          System.assertEquals(1, [SELECT count() FROM Product2]);
12      }
13 }
```

● 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":false


   );
13         response.setStatusCode(200);
14         return response;
15     }
16 }
```

6. TEST SCHEDULING LOGIC

: ● WarehouseSyncSchedule.apxc

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

```
6  }
```

● WarehouseSyncScheduleTest.apxc

```
1  @isTest
2  public class WarehouseSyncScheduleTest {
3      @isTest static void WarehousescheduleTest(){
4          String scheduleTime = '00 00 01 * * ?';
5          Test.startTest();
6          Test.setMock(HttpCalloutMock.class, new
   WarehouseCalloutServiceMock());
7          String jobID=System.schedule('Warehouse Time To Schedule to

8          Test.stopTest();
9          CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime >
   today];
10         System.assertEquals(jobID, a.Id,'Schedule ');
11      }
12 }
```

****