APEX TRIGGERS

*GET STARTED WITH APEX TRIGGERS:

1.AccountAddressTrigger.apxt

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

*BULK APEX TRIGGERS:

1.ClosedOpportunityTrigger.apxt

```
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 Task',
WhatId = opp.Id));
        }
       }
    if(taskList.size()>0){
        insert taskList;
    }
}
```

APEX TESTING

*GET STARTED WITH APEX UNIT TEST:

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
  private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
  if( date2 < date1) { return false; }

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

  //method to return the end of the month of a given date
  private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
        return lastDay;
  }

}
```

2.TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {

    @isTest static void test1(){
        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020')
);
        System.assertEquals(Date.parse('01/03/2020'), d);
    }

    @isTest static void test2(){
        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020')
);
        System.assertEquals(Date.parse('01/31/2020'), d);
    }
}
```

*TEST APEX TRIGGERS:

1.RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (before insert, before update)
{

  //check contacts prior to insert or update for invalid data
  For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') {   //invalidname is invalid
              c.AddError('The Last Name "'+c.LastName+'" is not allowed

        }
  }
}
```

*CREATE TEST DATA FOR APEX TESTS:

1.RandomContactFactory.apxc

```
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:

1.AccountProcessor.apxc

```
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds) {
        List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id from Contacts) from Account where Id in : accountIds];
        For(Account acc : accList) {
            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }
        update accList;
    }

}
```

2.AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
```

```
3
4       public static testmethod void testAccountProcessor() {
5           Account a = new Account();
6           a.Name = 'Test Account';
7           insert a;
8
9           Contact con = new Contact();
10          con.FirstName = 'Binary';
11          con.LastName = 'Programming';
12          con.AccountId = a.Id;
13
14          insert con;
15
16          List<Id> accListId = new List<Id>();
17          accListId.add(a.Id);
18
19          Test.startTest();
20          AccountProcessor.countContacts(accListId);
21          Test.stopTest();
22
23          Account acc = [Select Number_Of_Contacts__c from Account where
    Id = : a.Id];
24          System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),
    1);
25
26      }
27  }
```

***USE BATCH APEX:**

1.LeadProcessor.apxc

```
1   public class LeadProcessor implements Database.Batchable<sObject> {
2
3       public Database.QueryLocator start(Database.BatchableContext bc) {
4           // collect the batches of records or objects to be passed to
    execute
5             return Database.getQueryLocator([Select LeadSource From Lead
    ]);
6       }
7       public void execute(Database.BatchableContext bc, List<Lead> leads){
8           // process each batch of records
9               for (Lead Lead : leads) {
```

```
                lead.LeadSource = 'Dreamforce';
            }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
      }
}
```

2.LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

        @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company ='demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }

    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }

}
```

CONTROL PROCESSES WITH QUEUEABLE APEX:

1.AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
```

```
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
                 Contact cont = c.clone(false,false,false,false);
                 cont.AccountId =  acc.id;
                 lstContact.add( cont );
         }

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

    }

}
```

2.AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
     @isTest static void TestList()
     {
         List<Account> Teste = new List <Account>();
         for(Integer i=0;i<50;i++)
         {
             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 APEX SCHEDULER:**

1.DailyLeadProcessor.apxc

```
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;
        }
    }
}
```

2.DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
  static testMethod void testDailyLeadProcessor() {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<Lead>();
    for (Integer i = 0; i < 200; i++) {
                lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open - Not Contacted'));
        }
```

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

APEX INTEGRATION SERVICES

*APEX REST CALLOUTS:

1.AnimalLocator.apxc

```
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,
    Object>)JSON.deserializeUntyped(res.getBody());
11       animal = (Map<String, Object>) results.get('animal');
12         }
13 return (String)animal.get('name');
14     }
15 }
```

2.AnimalLocatorMock.apxc

```
1  @isTest
2  global class AnimalLocatorMock implements HttpCalloutMock {
3       // Implement this interface method
4      global HTTPResponse respond(HTTPRequest request) {
5          // Create a fake response
6          HttpResponse response = new HttpResponse();
7          response.setHeader('Content-Type', 'application/json');
8          response.setBody('{"animals": ["majestic badger", "fluffy
```

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

3.AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}
```

*APEX SOAP CALLOUTS:

1.ParkService.apxc

```
//
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    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.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>{'Yellowstone', 'Mackinac

        // end
        response.put('response_x', response_x);
    }
}
```

3.ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac

         System.assertEquals(parks, result);
    }
}
```

*APEX WEB SERVICES:

1.AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/',
'/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

2.AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }


    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE

***AUTOMATE RECORD CREATION*:**

1)MaintenanceRequest.apxt

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

2)MaintenanceRequestHelper.apxc

```
public class MaintenanceRequestHelper {

    public static void updateWorkOrders(List<Case> updatedWOs, Map<Id,Case> oldCaseMap){
        Set<Id> validWOIds = new Set<Id>(); //set of valid work order IDs

        //going thru updatedWOs and checking if there is closed one AND of type repair/routine maintenance -> throw it into our Set then
        for (Case c: updatedWOs) {
            if (oldCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed') {
                if (c.Type == 'Repair' || c.Type == 'Routine

                    validWOIds.add(c.Id);
                }
            }
        }

        //If our Set is NOT empty,we calculate times
        if (!validWOIds.isEmpty()) {
            List<Case> newCases = new List<Case>();

            Map<Id, Case>closedCaseMap = new Map<Id, Case>(updatedWOs);

            Map<Id, Decimal> maintCycleMap = new Map<Id, Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c, MIN(Equipment__r.Maintenance_Cycle__c)cycle
                                         FROM
```

```
    Equipment_Maintenance_Item__c
24                                          WHERE Maintenance_Request__c IN
    :validWOIds
25                                          GROUP BY
    Maintenance_Request__c];
26
27              List<Equipment_Maintenance_Item__c> itemList = [SELECT Id,
    Maintenance_Request__c
28                                          FROM
    Equipment_Maintenance_Item__c
29                                          WHERE Maintenance_Request__c IN
    :validWOIds];
30
31              //put every result into map with Id as key and Decimal as
    its value
32              for (AggregateResult ar : results) {
33                  maintCycleMap.put((Id) ar.get('Maintenance_Request__c'),
    (Decimal) ar.get('cycle') );
34              }
35
36              //Creating new cases here
37              for (Id caseId: validWOIds){
38                  Case cc = closedCaseMap.get(caseId);
39                  Case nc = new Case (ParentId = cc.Id,
40                                      Status = 'New',
41                                      Subject = 'Routine Maintenance',
42                                      Type = 'Routine Maintenance',
43                                      Vehicle__c = cc.Vehicle__c,
44                                      Equipment__c = cc.Equipment__c,
45                                      Origin = 'Web',
46                                      Date_Reported__c = Date.today());
47
48                  //calculating the maintenance request due dates
49                  nc.Date_Due__c = Date.today().addDays((Integer)
    maintCycleMap.get(cc.Id));
50                  newCases.add(nc);
51              }
52
53              insert newCases;
54
55
56              List<Equipment_Maintenance_Item__c> copiedWorkParts = new
    List<Equipment_Maintenance_Item__c>();
57              //cloning work parts here
```

```
            for (Case nc: newCases) {
                //going thru closedCaseMap and giving workparts right case ID
                for (Equipment_Maintenance_Item__c workparts: itemList) {
                    if (workparts.Maintenance_Request__c == nc.ParentId){
                       workparts.Maintenance_Request__c = nc.Id;
                    copiedWorkParts.add(workparts);
                    }
                }
            }
            update copiedWorkParts;
        }
    }
}
```

SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

1)WarehouseCalloutService.apxc

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

@future(callout=true)
public static void runWarehouseEquipmentSync() {
//ToDo: complete this method to make the callout (using @future) to the
//      REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of products from Http callout response
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your org based on the external ID SKU
}
}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse response)
{
List<Object> externalProducts = (List<Object>) JSON.deserializeUntyped(response.getBody()); //desrialize the json
```

```
    response
19  List<Product2> newProducts = new List<Product2>();
20  for(Object p : externalProducts)
21  {
22  Map<String, Object> productMap = (Map<String, Object>) p;
23  Product2 pr = new Product2();
24  //Map the fields in the response to the appropriate fields in the
    Equipment object
25  pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
26  pr.Cost__c = (Integer)productMap.get('cost');
27  pr.Current_Inventory__c = (Integer)productMap.get('quantity');
28  pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
29  pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
30  pr.Warehouse_SKU__c = (String)productMap.get('sku');
31  pr.ProductCode = (String)productMap.get('_id');
32  pr.Name = (String)productMap.get('name');
33  newProducts.add(pr);
34  }
35  return newProducts;
36  }
37  // Send Http GET request and receive Http response
38  public static HttpResponse getResponse() {
39  Http http = new Http();
40  HttpRequest request = new HttpRequest();
41  request.setEndpoint(WAREHOUSE_URL);
42  request.setMethod('GET');
43  HttpResponse response = http.send(request);
44  return response;
45  }
46  }
```

SCHEDULE SYNCHRONIZATION USING APEX CODE:

1)WarehouseSyncSchedule.apxc

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

TEST AUTOMATION LOGIC:

1)MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id
equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,

```

```
    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>();
```

```
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
           vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
   requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
   equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id,
   requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            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);
    }
}
```

2)MaintenanceRequestHelper.apxc

```
public class MaintenanceRequestHelper {

    public static void updateWorkOrders(List<Case> updatedWOs, Map<Id,Case> oldCaseMap){
        Set<Id> validWOIds = new Set<Id>(); //set of valid work order IDs

        //going thru updatedWOs and checking if there is closed one AND of type repair/routine maintenance -> throw it into our Set then
        for (Case c: updatedWOs) {
            if (oldCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed') {
                if (c.Type == 'Repair' || c.Type == 'Routine

                    validWOIds.add(c.Id);
                }
            }
        }

        //If our Set is NOT empty,we calculate times
        if (!validWOIds.isEmpty()) {
            List<Case> newCases = new List<Case>();

            Map<Id, Case>closedCaseMap = new Map<Id, Case>(updatedWOs);

            Map<Id, Decimal> maintCycleMap = 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 :validWOIds
                                         GROUP BY Maintenance_Request__c];

            List<Equipment_Maintenance_Item__c> itemList = [SELECT Id, Maintenance_Request__c
                                         FROM Equipment_Maintenance_Item__c
                                         WHERE Maintenance_Request__c IN :validWOIds];

            //put every result into map with Id as key and Decimal as
```

```
    its value
            for (AggregateResult ar : results) {
                maintCycleMap.put((Id) ar.get('Maintenance_Request__c'),
    (Decimal) ar.get('cycle') );
            }

            //Creating new cases here
            for (Id caseId: validWOIds){
                Case cc = closedCaseMap.get(caseId);
                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());

                //calculating the maintenance request due dates
                nc.Date_Due__c = Date.today().addDays((Integer)
    maintCycleMap.get(cc.Id));
                newCases.add(nc);
            }

            insert newCases;


            List<Equipment_Maintenance_Item__c> copiedWorkParts = new
    List<Equipment_Maintenance_Item__c>();
            //cloning work parts here
            for (Case nc: newCases) {
                //going thru closedCaseMap and giving workparts right
    case ID
                for (Equipment_Maintenance_Item__c workparts: itemList)
    {
                    if (workparts.Maintenance_Request__c ==
    nc.ParentId){
                       workparts.Maintenance_Request__c = nc.Id;
                    copiedWorkParts.add(workparts);
                    }
                }
            }
            update copiedWorkParts;
        }
```

```
    }
}
```

3)MaintenanceRequest.apxt

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

*TEST CALLOUT LOGIC:

1)WarehouseCalloutService.apxc

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

@future(callout=true)
public static void runWarehouseEquipmentSync() {
//ToDo: complete this method to make the callout (using @future) to the
//      REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of
products from Http callout response
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your org based
on the external ID SKU
}
}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse response)
{
List<Object> externalProducts = (List<Object>)
JSON.deserializeUntyped(response.getBody()); //desrialize the json
response
List<Product2> newProducts = new List<Product2>();
```

```
for(Object p : externalProducts)
{
Map<String, Object> productMap = (Map<String, Object>) p;
Product2 pr = new Product2();
//Map the fields in the response to the appropriate fields in the
Equipment object
pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
pr.Cost__c = (Integer)productMap.get('cost');
pr.Current_Inventory__c = (Integer)productMap.get('quantity');
pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
pr.Warehouse_SKU__c = (String)productMap.get('sku');
pr.ProductCode = (String)productMap.get('_id');
pr.Name = (String)productMap.get('name');
newProducts.add(pr);
}
return newProducts;
}
// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
return response;
}
}
```

2)WarehouseCalloutServiceTest.apxc

```
@isTest
private class WarehouseCalloutServiceTest {

    @isTest
    static void testRunWarehouseEquipmentSync(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        Test.startTest();
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();

        System.assertEquals(3, [select count() from Product2]);
    }

```

```
15
16 }
```

3)WarehouseCalloutServiceMock.apxc

```
1  public class WarehouseCalloutServiceMock implements HttpCalloutMock {
2      private String responseJson = '[' +
3
   '{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000

4
   '{"_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling

5
   '{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse

6                              ']';
7
8      // Implement this interface method
9      public HTTPResponse respond(HTTPRequest request) {
10         // Create a fake response
11         HttpResponse response = new HttpResponse();
12         response.setHeader('Content-Type', 'application/json');
13         response.setBody(responseJson);
14         response.setStatusCode(200);
15         return response;
16     }
17 }
```

*TEST SCHEDULING LOGIC:

1)WarehouseSyncSchedule.apxc

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

2)WarehouseSyncScheduleTest.apxc

```
1  @isTest
2  public class WarehouseSyncScheduleTest {
3
4      @isTest static void WarehousescheduleTest(){
5          String scheduleTime = '00 00 01 * * ?';
```

```
6           Test.startTest();
7           Test.setMock(HttpCalloutMock.class, new
    WarehouseCalloutServiceMock());
8           String jobID=System.schedule('Warehouse Time To Schedule to

9           Test.stopTest();
10          //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  }
```