APEX TRIGGERS

1. GET STARTED WITH APEX TRIGGERS:

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;
        }
    }
}
```

2. BULK APEX TRIGGERS:

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

1. GET STARTED WITH APEX UNIT TEST:

VerifyDate.apxc

```
public class VerifyDate {
 public static Date CheckDates(Date date1, Date date2) {
```

```
                if(DateWithin30Days(date1,date2)) {
                return date2;
        } else {
                return SetEndOfMonthDate(date1);
        }
 }
 private static Boolean DateWithin30Days(Date date1, Date date2) {
 if( date2 < date1) { return false; }
 Date date30Days = date1.addDays(30);
        if( date2 >= date30Days ) { return false; }
        else { return true; }
 }
 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;
 }
}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
   @isTest static void testDate2within30daysofDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 04, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 04, 11);
      System.assertEquals(testDate,resultDate);
   }
   @isTest static void testDate2beforeDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 02, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 02, 11);
      System.assertNotEquals(testDate, resultDate);
```

```
    }
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

2. TEST APEX TRIGGERS:

RestrictContactByName.apxt

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

3.CREATE TEST DATA FOR APEX TESTS:

RandomContactFactory.apxc

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

ASYNCRONOUS APEX

1. USE FUTURE METHODS:

AccountProcessor.apxc

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountId_lst) {
      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from
contacts) from account]);
      for(account a:account_lst_all) {
         account_cno.put(a.id,a.contacts.size()); //populate the map
      }
      List<account> account_lst = new List<account>(); // list of account that we will
upsert
      for(Id accountId : accountId_lst) {
         if(account_cno.containsKey(accountId)) {
            account acc = new account();
            acc.Id = accountId;
            acc.Number_of_Contacts__c = account_cno.get(accountId);
            account_lst.add(acc);
         }
      }
      upsert account_lst;
   }
}
```

AccountProcessorTest.apxc

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

```
@isTest
   public static void testFunc() {
      account acc = new account();
      acc.name = 'MATW INC';
      insert acc;
      contact con = new contact();
      con.lastname = 'Mann1';
      con.AccountId = acc.Id;
      insert con;
      contact con1 = new contact();
      con1.lastname = 'Mann2';
      con1.AccountId = acc.Id;
      insert con1;
      List<Id> acc_list = new List<Id>();
      acc_list.add(acc.Id);
      Test.startTest();
 AccountProcessor.countContacts(acc_list);
      Test.stopTest();
      List<account> acc1 = new List<account>([select Number_of_Contacts__c from
account where id = :acc.id]);
      system.assertEquals(2,acc1[0].Number_of_Contacts__c);
   }
}
```

2. USE BATCH APEX:

LeadProcessor.apxc

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

```
        for(Lead lead : leads){
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    global void finish(Database.BatchableContext bc){
    }
}
```

LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 10 leads
        for (Integer i=0;i<10;i++) {
            leads.add(new Lead(LastName='Lead '+i, Company='TestCompany'));
        }
        insert leads;
    }
    static testmethod void test() {
        Test.startTest();
            LeadProcessor lp = new LeadProcessor();
            Database.executeBatch(lp);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(10, [SELECT count() FROM Lead where LeadSource =
'Dreamforce']);
    }
}
```

3. CONTROL PROCESSES WITH QUEUEABLE APEX

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {
   public contact c;
   public String state;

   public AddPrimaryContact(Contact c, String state) {
      this.c = c;
      this.state = state;
   }
   public void execute(QueueableContext qc) {
      system.debug('this.c = '+this.c+' this.state = '+this.state);
      List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
      List<contact> c_lst = new List<contact>();
      for(account a: acc_lst) {
         contact c = new contact();
         c = this.c.clone(false, false, false, false);
         c.AccountId = a.Id;
         c_lst.add(c);
      }
      insert c_lst;
   }
}
```

AddPrimaryContactTest.apxc

```
@IsTest
public class AddPrimaryContactTest {
   @IsTest
   public static void testing() {
      List<account> acc_lst = new List<account>();
      for (Integer i=0; i<50;i++) {
         account a = new account(name=string.valueOf(i),billingstate='NY');
         system.debug('account a = '+a);
         acc_lst.add(a);
      }
      for (Integer i=0; i<50;i++) {
```

```
        account a = new account(name=string.valueOf(50+i),billingstate='CA');
        system.debug('account a = '+a);
        acc_lst.add(a);
      }
      insert acc_lst;
      Test.startTest();
      contact c = new contact(lastname='alex');
      AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
      system.debug('apc = '+apc);
      System.enqueueJob(apc);
      Test.stopTest();
      List<contact> c_lst = new List<contact>([select id from contact]);
      Integer size = c_lst.size();
      system.assertEquals(50, size);
   }
}
```

4. SCHEDULE JOBS USING APEX SCHEDULER:

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext ctx){
      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];
      if(leads.size() > 0){
         List<Lead> newLeads = new List<Lead>();
         for(Lead lead : leads){
            lead.LeadSource = 'DreamForce';
            newLeads.add(lead);
         }
         update newLeads;
      }
   }
}
```

DailyLeadProcessorTest.apxc

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

APEX INTEGRATION SERVICES

1. APEX REST CALLOUTS:

AnimalLocator.apxc

```
public class AnimalLocator {
 public class cls_animal {
         public Integer id;
         public String name;
         public String eats;
         public String says;
 }
public class JSONOutput{
 public cls_animal animal;
```

```
}

    public static String getAnimalNameById (Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
        //request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
            system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }
}
```

AnimalLocatorMock.apxc

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}')
        return response;
    }
}
```

AnimalLocatorTest.apxc

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
```

```
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}
```

2. APEX SOAP CALLOUTS:

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-
service.herokuapp.com/service/parks';
        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;
        }
    }
}
```

ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
   @isTest
   static void testParkLocator() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String[] arrayOfParks = ParkLocator.country('India');
      System.assertEquals('Park1', arrayOfParks[0]);
   }
}
```

3. APEX WEB SERVICES:

AccountManager.apxc

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

AccountManagerTest.apxc

```
@Istest(SeeAllData=true)
public class AccountManagerTest {
   @IsTest
   public static void testaccountmanager() {
      RestRequest request = new RestRequest();
      request.requestUri = 'https://mannharleen-dev-
ed.my.salesforce.com/services/apexrest/Accounts/00190000016cw4tAAA/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
         system.debug('test account result = '+ AccountManager.getAccount());
   }
}
```

APEX SPECIALIST SUPERBADGE

1. AUTOMATE RECORD CREATION:

MaintenanceRequest.apxt

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

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);
        }
      }
    }
    if (!validIds.isEmpty()){
      List<Case> newCases = new List<Case>();
      Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle_c,
Equipmentc, Equipmentr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc 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(Equipmentr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc
WHERE Maintenance_Requestc 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.Equipmentr.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);
            }
        }
        insert ClonedWPs;
    }
  }
}
```

2. SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  @future(callout=true)
  public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
```

```
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }
            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }
    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

3. SCHEDULE SYNCHRONIZATION USING APEX CODE:

WarehouseSyncSchedule.apxc

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

4. TEST AUTOMATION LOGIC:

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_Itemc(Equipment_c = equipmentId,
                                                Maintenance_Request__c = requestId);
    return wp;
  }


  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
```

```
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment_c, Date_Reportedc, Vehiclec,
Date_Due_c
                from case
                where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }

    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,
emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
```

```
        update emptyReq;
        test.stopTest();

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

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle_C> vehicleList = new list<Vehicle_C>();
        list<Product2> equipmentList = new list<Product2>();
        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);
    }
}
```

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);
                }
            }
```

```
    }
    if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle_c,
Equipmentc, Equipmentr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc 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(Equipmentr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc
WHERE Maintenance_Requestc 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));
            }

            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);

            }
         }
         insert ClonedWPs;
      }
   }
}
```

MaintenanceRequest.apxt

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

5. TEST CALLOUT LOGIC:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

   private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

   //@future(callout=true)
```

```
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);


        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

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

        }
    }
```

```
}
```

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]);
  }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
  // implement http mock callout
  global static HttpResponse respond(HttpRequest request){

    System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
    System.assertEquals('GET', request.getMethod());

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
```

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

6. TEST SCHEDULING LOGIC:

WarehouseSyncSchedule.apxc

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

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 to Test',
scheduleTime, new WarehouseSyncSchedule());
        Test.stopTest();
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
    }
}
```

****