Get Started with Apex Triggers

AccountAddressTrigger.apxt

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

}
```

Bulk Apex Triggers

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;
    }
}
```

Get Started with Apex Unit Tests

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;
        }

}
```

TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
   @isTest static void testOldDate(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));
      System.assertEquals(date.newInstance(2016, 4, 30), dateTest);
   }

   @isTest static void testLessThan30Days(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(20));
      System.assertEquals(date.today().addDays(20), dateTest);
   }

   @isTest static void testMoreThan30Days(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(31));
      System.assertEquals(date.newInstance(2016, 4, 30), dateTest);
```

```
    }

}
```

Test Apex Triggers

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 for
DML');
                }

        }
}
```

TestRestrictContactByName.apxc

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        //try inserting a Contact with INVALIDNAME
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;

        // Perform test
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        // Verify
        // In this case the creation should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                             result.getErrors()[0].getMessage());
```

```
    }
}
```

Create Test Data for Apex Tests

RandomContactFactory.apxc

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }
}
```

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);
    }

}
```

Use Batch Apex

LeadProcessor.apxc

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    // instance member to retain state across transactions
    global Integer recordsProcessed = 0;

    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {

                lead.LeadSource = 'Dreamforce';
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;

        }
        update leads;
    }

    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');

    }
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
 @testSetup
   static void setup() {
      List<Lead> leads = new List<Lead>();
      // insert 200 leads
      for (Integer i=0;i<200;i++) {
         leads.add(new Lead(LastName='Lead '+i,
            Company='Lead', Status='Open - Not Contacted'));
      }
      insert leads;
   }

   static testmethod void test() {
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp, 200);
      Test.stopTest();

      // after the testing stops, assert records were updated properly
      System.assertEquals(200, [select count() from lead where LeadSource =
'Dreamforce']);
   }
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{
   Contact con;
   String state;

   public AddPrimaryContact(Contact con, String state){
      this.con = con;
      this.state = state;
   }
   public void execute(QueueableContext qc){
      List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT
200];
```

```
        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }

        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();

        System.assertEquals(50, [select count() from Contact]);
    }
}
```

Schedule Jobs Using the 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{
   //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
   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();
      // Schedule the test job
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
```

```
new DailyLeadProcessor());

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

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 JSONOutput parse(String json){
        //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
        //}
}

    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());
        //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
                system.debug('results= ' + results.animal.name);
```

```
        return(results.animal.name);
    }

}
```

AnimalLocatorTest.apxc

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

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]);
    }
}
```

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) {
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

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

Apex Web Services

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
   }
}
```

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

      // Call the method to test
      Account  acc = AccountManager.getAccount();

      // Verify results
      System.assert(acc != null);
   }

   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;

      Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
      Insert con;
```

```
        return acc.Id;
    }
}
```

Apex Specialist

Challenge 1:

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,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
        }

            for(Case cc : closedCasesM.values()){
                Case nc = new Case (
```

```
            ParentId = cc.Id,
            Status = 'New',
            Subject = 'Routine Maintenance',
            Type = 'Routine Maintenance',
            Vehicle__c = cc.Vehicle__c,
            Equipment__c =cc.Equipment__c,
            Origin = 'Web',
            Date_Reported__c = Date.Today()

        );

        If (maintenanceCycles.containskey(cc.Id)){
            nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
        }

        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);
  }
}
```

Challenge 2:

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() {
    //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;

    }
    }
```

Challenge 3:

WarehouseSyncSchedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {
WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

Challenge 4:

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);
        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_Reported__c, Vehicle__c,
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,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                        FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
        AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

    for (AggregateResult ar : results){
        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }

        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
            Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }

            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);
   }
}
```

Challenge 5:

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() {
      //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;

    }
    }
```

WarehouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @isTest static void mainTest(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        Test.startTest();
        Id jobID = System.enqueueJob(new WarehouseCalloutService());
        //System.assertEquals('Queued',aaj.status);
        Test.stopTest();
        AsyncApexJob aaj = [SELECT Id, Status, NumberOfErrors FROM AsyncApexJob WHERE Id =
:jobID];
        System.assertEquals('Completed',aaj.status);
        System.assertEquals(0, aaj.NumberOfErrors);
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@istest
global class WarehouseCalloutServiceMock implements HttpCalloutMock{
    // implement http mock callout
    global HttpResponse respond(HttpRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":true,"quantity":5,"name":"
Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"220000"}]');
        response.setStatusCode(200);
        return response;
    }

}
```

Challenge 6

WarehouseSyncSchedule.apxc

```
global  class WarehouseSyncSchedule implements Schedulable{
// implement scheduled code here
global  void execute (SchedulableContext sc){
WarehouseCalloutService.runWarehouseEquipmentSync();
//optional this can be done by debug mode
String sch = '00 00 01 * * ?';//on 1 pm
System.schedule('WarehouseSyncScheduleTest', sch, new WarehouseSyncSchedule());
}
}
```

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();
      //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on
UNIX systems.
      // This object is available in API version 17.0 and later.
      CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
      System.assertEquals(jobID, a.Id,'Schedule ');


   }
}
```