APEX TRIGGERS

GET STARTED WITH APEX TRIGGERS

AccountAddressTrigger

```
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

```
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 Tests

```
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:
@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

RestrictContactByName:
```
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 :
```
@isTest
public class TestRestrictContactByName
{
   @isTest static void Test_insertupdateContact()
   {
      Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';


      Test.startTest();
      Database.SaveResult result=Database.insert(cnt, false);
      Test.stopTest();

      System.assert(!result.isSuccess());
```

```
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
    }

}
```

Create test data for Apex tests

RandomContactFactory :

```
public class RandomContactFactory {
    public  static List<Contact> generateRandomContacts(Integer num,String lastName)
    {
        List<Contact> ContactList =new List<contact>();
        for(Integer i = 1;I<=num;i++){
            Contact ct=new Contact(FirstName = 'Test '+i, LastName=lastName);
            contactList.add(ct);
        }
        return contactList;
}

}
```

Asynchronous Apex

Use future methods

AccountProcessor :

```
public class AccountProcessor
{
  @future
    public static void countContacts(List<Id> accIds)
    {
        List<Account> acc = [select id,
                          Number_Of_Contacts__c, (select id from contacts)
                          from account where id in: accIds];
        for(Account a:acc)
        {
            List<Contact> updContact = a.contacts;
            a.Number_Of_Contacts__c = updContact.size();
        }
        update acc;
    }
}
```
AccountProcessorTest:
```
@IsTest
public class AccountProcessorTest {

    public static testmethod void TestAccountProcessorTest(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact cont   = New Contact();
        cont.FirstName = 'Bob';
        cont.LastName  = 'Masters';
        cont.AccountId = a.Id;
        Insert cont;

        List<Id> accIds = new List<Id>();
        accIds.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(accIds);
        Test.stopTest();

        Account Acc = [select Number_Of_Contacts__c from Account where id = :a.Id LIMIT
1];
        System.assertEquals(Integer.valueOf(Acc.Number_Of_Contacts__c), 1);
    }

}
```
Use Batch Apex

LeadProcessor:
```
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> scope)
    {
        List<Lead> leads=new List<Lead>();
        for(lead l : scope)
        {
            l.LeadSource='Dreamforce';
            leads.add(l);
        }
        update leads;
    }

    global void finish(Database.BatchableContext bc)
    {
      AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
            JobItemsProcessed,
            TotalJobItems, CreatedBy.Email
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()];
        system.debug(job);
    }

}
```

LeadProcessorTest:

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();

        // insert 10 accounts
        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 uca = new LeadProcessor();
        Id batchId = Database.executeBatch(uca);
        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
```
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
```
@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:
```
global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
      List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

      if(!lList.isEmpty()) {
  for(Lead l: lList) {
   l.LeadSource = 'Dreamforce';
  }
  update lList;
 }
   }
}
DailyLeadProcessorTest:
@isTest
public 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 INTEGRATION SERVICES

Apex REST callouts

AnimalLocator:

```
public class AnimalLocator {
   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.setMethod('GET');
        HttpResponse response = http.send(request);
          String strResp = '';
           system.debug('****response '+response.getStatusCode());
           system.debug('****response '+response.getBody());

        if (response.getStatusCode() == 200)
        {

           Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

           Map<string,object> animals = (map<string,object>) results.get('animal');
            System.debug('Received the following animals:' + animals );
            strResp = string.valueof(animals.get('name'));
            System.debug('strResp >>>>>>' + strResp );
        }
        return strResp ;
    }

}
AnimalLocatorTest:
@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);
    }
}
AnimalLocatorMock
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
         HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

ParkService:

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

}

}
 ParkLocatorTest:
@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:

```
@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:

```
@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:
```
@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;
    }
}
```
SUPERBADGE 1

APEX Specialist superbadge

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,
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);
    }
}
```

challenge 2:
WarehouseCalloutService .apxc-

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    //class that makes a REST callout to an external warehouse system to get a list of
equipment that needs to be updated.
    //The callout’s JSON response returns the equipment records that you upsert in
Salesforce.
    @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());
            //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
            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();
    }
}
```

method used to run it:

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

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_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.aptxt-

```
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(){

        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;
    }
}
```

Challenge 6:

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();
        //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 ');


    }
}
```