APEX SPECIALIST SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.axpt

```
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;
        }
    }
}
```

ClosedOpportunityTrigger.axpt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();

   for(opportunity opp: Trigger.New){

        if(opp.StageName == 'ClosedWon' ) {
           taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
        }

    }
    if(taskList.size()>0){
        insert tasklist;
    }
}
```

APEX TESTING

VerifyDate.apxc

```
public class VerifyDate {

	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 {
   private static Date dateToday = date.today();
   private static Integer totalDays = Date.daysInMonth(dateToday.year(), dateToday.month());

   @isTest static void testOldDate(){
      Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(-1));
      System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(), totalDays),
dateTest);
   }

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

   @isTest static void testMoreThan30Days(){
      Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(31));
      System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(), totalDays),
dateTest);
   }
}
```

<u>RestrictContactByName.apxt</u>

```
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');
		}
}
```

<u>TestRestrictContactByName.apxc</u>

```
@isTest
public class TestRestrictContactByName {
static testMethod void  Test()
   {

        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='Raam', LastName='Leela' ,
email='ramleela@test.com');
        Contact c2 = new Contact(FirstName='gatsby', LastName = 'INVALIDNAME',
email='gatsby@test.com');
        listContact.add(c1);
        listContact.add(c2);

        Test.startTest();
           try
           {
              insert listContact;
           }
           catch(Exception ee)
           {
           }

        Test.stopTest();

   }
}
```

RandomContactFactory.apxc

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer NumberofContacts,
String lName){
        List<Contact> con = new List<Contact>();
        for(Integer i=0; i<NumberofContacts; i++){
            lName = 'Test'+i;
            Contact c = new Contact(FirstName=lName, LastName=lName);
            con.add(c);
        }
        return con;
    }
}
```

ASYNCHRONOUS APEX

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);
```

```
  }

}
```

LeadProcessor.apxc

```
global class LeadProcessor implements Database.Batchable<sObject> {
  global Integer count = 0;

  global Database.QueryLocator start (Database.BatchableContext bc) {
    return Database.getQueryLocator('Select Id, LeadSource from lead');
  }

  global void execute (Database.BatchableContext bc,List<Lead> l_lst) {
    List<lead> l_lst_new = new List<lead>();
    for(lead l : l_lst) {
      l.leadsource = 'Dreamforce';
      l_lst_new.add(l);
      count+=1;
    }
    update l_lst_new;
  }

  global void finish (Database.BatchableContext bc) {
    system.debug('count = '+count);
  }
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

  @isTest
  public static void testit() {
        List<lead> l_lst = new List<lead>();
        for (Integer i = 0; i<200; i++) {
            Lead l = new lead();
            l.LastName = 'name'+i;
            l.company = 'company';
            l.Status =  'somestatus';
            l_lst.add(l);
        }
        insert l_lst;

        test.startTest();

        Leadprocessor lp = new Leadprocessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();

    }

}
```

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);
    }

}
```

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements schedulable{

    public void execute(schedulableContext sc) {
        List<lead> l_lst_new = new List<lead>();
        List<lead> l_lst = new List<lead>([select id, leadsource from lead where leadsource
= null]);
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
        }
        update l_lst_new;
    }
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {

    @isTest
    public static void testing() {

        List<lead> l_lst = new List<lead>();
        for(Integer i=0;i<200;i++) {
            lead l = new lead();
            l.lastname = 'lastname'+i;
            l.Company = 'company'+i;
            l_lst.add(l);
        }
        insert l_lst;

        Test.startTest();
        DailyLeadProcessor dlp = new DailyLeadProcessor ();
        String jobId = System.Schedule('dailyleadprocessing','0 0 0 1 12 ? 2016',dlp);
        Test.stopTest();

        List<lead> l_lst_chk = new List<lead>([select id,leadsource from lead where
leadsource != 'Dreamforce']);
        System.assertequals(0,l_lst_chk.size());
    }

}
```

APEX INTEGRATION SERVICES

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.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);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        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());
    //Httpresponse response = AnimalLocator.getAnimalNameById(1);
    String s =  AnimalLocator.getAnimalNameById(1);
    system.debug('string returned: ' + s);
  }

}
```

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);
   }
}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
         RestRequest request=RestContext.request;
         string accountId=request.requestURI.substringBetween('Accounts/','/contacts');
        Account result=[SELECT Id,Name,(Select Id,Name from Contacts) from Account where
Id=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest {
  @isTest static void testGetContactsByAccountId(){
    Id recordId=createTestRecord();
    RestRequest request=new RestRequest();
    request.requestUri='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+
    recordId+'/contacts';
    request.httpMethod='GET';
    RestContext.request=request;
    Account thisAccount=AccountManager.getAccount();
    System.assert(thisAccount != null);
    System.assertEquals('Test record',thisAccount.Name);
  }
  static Id createTestRecord(){
    Account accountTest=new Account(
    Name='Test record'
    );
    insert accountTest;
    Contact contactTest=new Contact(
    FirstName='John',LastName='Doe',AccountId=accountTest.Id);
    insert contactTest;
    return accountTest.Id;
  }
}
```

APEX SPECIALIST SUPERBADGE

CHALLENGE 1

MaintainanceRequestHelper.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));
          } else {
               nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.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;
      }
   }
}
```

MaintainanceRequest.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();
    }

}
```

CHALLENGE 3

WarehouseSyncShedule.apxc

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

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(){

        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 ');

    }
}
```