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

VerifyDate

```
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
	@TestVisible 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
	@TestVisible 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
private class TestVerifyDate {

  @isTest static void Test_CheckDates_case1(){
    Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
    System.assertEquals(Date.parse('01/05/2020'), D);
  }

  @isTest static void Test_CheckDates_case2(){
    Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
    System.assertEquals(Date.parse('01/31/2020'), D);
```

```
   }

   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
      System.assertEquals(false,flag);
   }

   @isTest static void Test_DateWithin30Days_case2(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
      System.assertEquals(false,flag);
   }

   @isTest static void Test_DateWithin30Days_case3(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
      System.assertEquals(true,flag);
   }

   @isTest static void Test_SetEndOfMonthDate(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
   }

}
```

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 without sharing class AccountProcessor {

   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accounts = [SELECT Id, (SELECT Id FROM Contacts) FROM Account WHERE
Id IN :accountIds];

      for(Account acc: accounts){
         acc.Number_Of_Contacts__c = acc.Contacts.size();
      }
      update accounts;
   }
}
```

AccountProcessorTest

```
@isTest
private class AccountProcessorTest {

   @isTest
   private static void countContactsTest() {
      //Load test data
      List<Account> accounts = new List<Account>();
      for (Integer i=0; i<300; i++) {
         accounts.add(new Account(Name='Test Account' + i));
      }
      insert accounts;

      List<Contact> contacts = new List<Contact>();
      List<Id> accountIds = new List<Id>();
      for (Account acc: accounts) {
         contacts.add(new Contact(FirstName=acc.Name, LastName='TestContact',
AccountId=acc.Id));
         accountIds.add(acc.Id);
      }
      insert contacts;

      //do the test
```

```
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        //Check result
        List<Account> accs = [SELECT Id, Number_of_Contacts__c FROM Account];
        for (Account acc : accs) {
            System.assertEquals(1, acc.Number_Of_Contacts__c,'ERROR: At least 1 Account record
with incorrect');
        }
    }
}
```

USE BATCH APEX

LeadProcessor

```
public without sharing class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext dbc) {
        return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
    }

     public void execute(Database.BatchableContext dbc, List<Lead> leads) {
        for(Lead l : leads) {
            l.leadsource = 'Dreamforce';
        }
        update leads;
    }

    public void finish (Database.BatchableContext dbc){
        System.debug('Done');
    }
}
```

LeadProcessorTest

```
@isTest
private class LeadProcessorTest {

    @isTest
```

```
   private static void testBatchClass(){

      //Load test data
      List<Lead> leads =new List<Lead>();
      for (Integer i=0; i<200; i++){
         leads.add(new Lead(LastName='Connock', Company='Salesforce'));
         }
      insert leads;

      //perfrom the test
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp, 200);
      Test.stopTest();
      //check the result
      List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE LeadSource = 'Dreamforce'];
      System.assertEquals(200,updatedLeads.size(), 'ERROR: At least 1 Lead record not updated
correctly');
   }
}
```

CONTROL PROCESSES WITH QUEUEABLE APEX

AddPrimaryContact

```
public without sharing class AddPrimaryContact implements Queueable {

   private Contact contact;
   private String state;

         // Constructor - pass in Contact sObject and State abbreviation as arguments
   public AddPrimaryContact(Contact inputContact, String inputState) {

                  // Store in class instance variables
      this.contact = inputContact;
      this.state = inputState;
   }

   public void execute(QueueableContext context) {
      //System.debug('Job Id ' + context.getJobId());
```

```
                    // Retrieve 200 Account records
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state LIMIT
200];

                    // Create empty list of Contact records
        List<Contact> contacts = new List<Contact>();

                    // Iterate through the Account records
        for ( Account acc : accounts) {

                                // Clone (copy) the Contact record, make the clone a child of the specific
Account record
                                // and add to the list of Contacts
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }

                    // Add the new Contact records to the database
        insert contacts;
    }
}
```

AddPrimaryContactTest

```
@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass() {

        // Load test data
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<500; i++) {
            Account acc = new Account(Name='Test Account');
            if ( i<250 ) {
                acc.BillingState = 'NY';
            } else {
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact = new Contact(FirstName='Simon', LastName='Connock');
        insert contact;

        // Perform the test
        Test.startTest();
        Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));
        Test.stopTest();

        // Check the result
        List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState =
'CA'];
        System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of Contact records
found');
    }
}
```

SCHEDULE JOBS USING THE APEX SCHEDULER

DailyLeadProcessor

```
public without sharing class DailyLeadProcessor implements Schedulable {

   public void execute(SchedulableContext ctx) {
      //System.debug('Context ' + ctx.getTriggerId()); // Returns the ID of the CronTrigger
scheduled job

                  // Get 200 Lead records and modify the LeadSource field
      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT
200];
      for ( Lead l : leads) {
         l.LeadSource = 'Dreamforce';
      }

                  // Update the modified records
      update leads;
   }
}
```

DailyLeadProcessorTest

```
@isTest c
private class DailyLeadProcessorTest {

  private static String CRON_EXP = '0 0 0 ? * * *'; // Midnight every day

  @isTest
  private static void testSchedulableClass() {

    // Load test data
    List<Lead> leads = new List<Lead>();
    for (Integer i=0; i<500; i++) {
      if ( i < 250 ) {
        leads.add(new Lead(LastName='Connock', Company='Salesforce'));
      } else {
        leads.add(new Lead(LastName='Connock', Company='Salesforce',
LeadSource='Other'));
      }
    }
    insert leads;

    // Perform the test
    Test.startTest();
    String jobId = System.schedule('Process Leads', CRON_EXP, new DailyLeadProcessor());
    Test.stopTest();

    // Check the result
    List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =
'Dreamforce'];
    System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 record not updated
correctly');

    // Check the scheduled time
    List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger
WHERE Id = :jobId];
    System.debug('Next Fire Time ' + cts[0].NextFireTime);

    // Not sure this works for all timezones
          //Datetime midnight = Datetime.newInstance(Date.today(),
Time.newInstance(0,0,0,0));
        //System.assertEquals(midnight.addHours(24), cts[0].NextFireTime, 'ERROR: Not scheduled
for Midnight local time');
    }
}
```

APEX INTEGRATION SERVICES : APEX REST CALLOUTS

AnimalLocator

```
public class AnimalLocator {

    public static String getAnimalNameById (Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

            Map<String, Object> result = (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String, Object>)result.get('animal');
            System.debug('name: '+string.valueOf(animal.get('name')));
            return string.valueOf(animal.get('name'));
    }
}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {

    @isTest
    static void animalLocatorTest1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual,expected);
    }
}
```

AnimalLocatorMock

```
@isTest
```

```
global class AnimalLocatorMock implements HttpCalloutMock {

   global HttpResponse respond(HttpRequest request) {
      HttpResponse response = new HttpResponse();
      response.setHeader('ContentType', 'application/json');
      response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"}}');
      response.setStatusCode(200);
      return response;
   }
}
```

APEX SOAP CALLOUTS

ParkService

```
//Generated by wsdl2apex

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 List< String > country(String country) {
        ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
        return prkSvc.byCountry(country);
    }

}
```

ParkLocatorTest

```
@isTest
```

```
private class ParkLocatorTest {

    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country ='United States';
        List<String> expectedParks = new List<String>{'Yosemite','Sequoia','Crater Lake'};

        System.assertEquals(expectedParks,ParkLocator.country(country));

    }

}
```

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) {
                // start - specify the response you want to send
                parkService.byCountryResponse response_x = new parkService.byCountryResponse();
                response_x.return_x = new List<String>{'Yosemite','Sequoia','Crater Lake'};
                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 request = RestContext.request;
      String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
      Account result = [SELECT ID,Name,(SELECT ID, FirstName, LastName FROM Contacts)
                  FROM Account
                  WHERE Id = :accountID];
      return result;
   }

}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest {


    @isTest
   static void testGetAccount() {
      Account a = new Account(Name='TestAccount');
      insert a;
      Contact c= new Contact(AccountId=a.Id,FirstName='Test', LastName='Test');
      insert c;

      RestRequest request = new RestRequest();
      request.requestUri
='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/contacts';
      request.httpMethod ='GET';
      RestContext.request = request;

      Account myAcct = AccountManager.getAccount();
      //verify results
      System.assert(myAcct != null);
      System.assertEquals('TestAccount', myAcct.Name);

   }

}
```

SUPERBADGE : APEX SPECIALIST
AUTOMATE RECORD CREATION
MaintenanceRequestHelper

```
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;
       }
    }
}
```

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after update) {

   if(Trigger.isUpdate && Trigger.isAfter){

      MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

   }

}
```

SYNCHRONIZE SALESFORCE DATA WITH AN EXTERNAL SYSTEM

WarehouseCalloutService

```
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();
    }

}
```

After saving the code open execute anonymous window (CTRl+E) and run this method in it ,

System.enqueueJob(new WarehouseCalloutService());

SCHEDULE SYNCHRONIZATION

WarehouseSyncSchedule

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

TEST AUTOMATION LOGIC

MaintenanceRequestHelperTest

```
@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);
    }
}
```

MaintenanceRequestHelper1(changed orginal code present in previous challenge)

```
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;
      }
   }
}
```

MaintenanceRequest1

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

TEST CALLOUT LOGIC

WarehouseCalloutService1(changed orginal code present in previous challenge)

```
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

```
@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

```
@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;
    }
}
```

TEST SCHEDULING LOGIC

WarehouseSyncSchedule1(changed orginal code present in previous challenge)

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest

```
@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 ');


    }
}
```

SUPERBADGE : PROCESS AUTOMATION

AUTOMATE LEADS

Rule name : Anything
Error Condition Formula : OR(AND(LEN(State) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", State))), NOT(OR(Country ="US",Country ="USA",Country ="United States",
ISBLANK(Country))))

AUTOMATE ACCOUNTS

Validation Rule : US_Address
Error Condition Formula : OR(AND(LEN(BillingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", BillingState))
),AND(LEN(ShippingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", ShippingState))
),NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States",
ISBLANK(BillingCountry))),
NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States",
ISBLANK(ShippingCountry))))

Validation Rule : Name Change
ISCHANGED(Name) && (OR(ISPICKVAL(Type ,'Customer - Direct') ,ISPICKVAL(Type ,'Customer - Channel')))

AUTOMATE SETUPS

Day of the Week
Formula :
Case (WEEKDAY(Date__c),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
5,"Thursday",
6,"Friday",
7,"Saturday",
Text(WEEKDay(Date__c)))

Closed deal --> action(set robo)
date field fromula -->CASE(MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7), 0, [Opportunity].CloseDate + 181, 6, [Opportunity].CloseDate + 182, [Opportunity].CloseDate + 180)