SALESFORCE DEVELOPER CATALYST SELF LEARNING & SUPERBADGES

APEX TRIGGERS -->

GET STARTED WITH APEX TRIGGERS

//AccountAddressTrigger//

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account a:Trigger.New){
        if(a.Match_Billing_Address__c == True){
            a.ShippingPostalCode = a.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
    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
private class TestVerifyDate {
   @istest static void case1(){
      Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('01/06/2022'));
      System.assertEquals(Date.parse('01/06/2022'), d);
   }

   @istest static void case2(){
      Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('06/06/2022'));
      System.assertEquals(Date.parse('01/31/2022'), d);
   }

}
```

TEST APEX TRIGGERS

```
//RestrictContactByName// --> Code Provided on Github
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
private class TestRestrictContactByName {
   @isTest static void Trcase1(){
      Contact c = new Contact();
      c.LastName = 'INVALIDNAME';

      Test.startTest();
      Database.SaveResult result = Database.insert(c, 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 numct,
string lastname){
      List<Contact> conts = new List<Contact>();
      for(integer i=0;i<numct;i++){
         Contact cnt = new Contact(FirstName = 'Test '+i, LastName =
lastname);
         conts.add(cnt);
```

```
        }
        return conts;
    }
}
```

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 countcontsCase1(){
        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;

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

        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 Contact');
        }
    }
}
```

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
public class LeadProcessorTest {
    @isTest
    private static void batchtest(){
        List<Lead> leads = new List<Lead>();
        for(integer i=0; i<200; i++){
            leads.add(new Lead(LastName='Carter', Company='Salesforce'));
        }
        insert leads;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp,200);
        Test.stopTest();

        List<lead> updateLeads = [SELECT Id FROM Lead WHERE
LeadSource='Dreamforce'];
        System.assertEquals(200, updateLeads.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;

   public AddPrimaryContact (Contact inputContact, String inputState){
      this.contact = inputContact;
      this.state = inputState;
   }

   public void execute(QueueableContext context){
      List<Account> accounts = [SELECT Id FROM Account Where
BillingState = :state LIMIT 200];
      List<Contact> contacts = new List<Contact>();
      for(Account acc: accounts){
         contact contactclone = contact.clone();
         contactClone.AccountId = acc.id;
         contacts.add(contactClone);
      }
      insert contacts;
   }
}
```

```
//AddPrimaryContactTest//
@isTest
private class AddPrimaryContactTest {
   @isTest
   private static void Queueabletest(){
      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 = 'Steve', LastName =
'Rogers');
          insert contact;

            Test.startTest();
            Id jobid = System.enqueueJob(new AddPrimaryContact(contact,
'CA'));
            Test.stopTest();

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

SCHEDULE JOBS USING THE APEX SCHEDULER

```
//DailyLeadProcessor//
public without sharing class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = null LIMIT 200];
        for(Lead l: leads){
```

```
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}
```

```
//DailyLeadProcessorTest//
@isTest
public class DailyLeadProcessorTest {
    private static String CRON_EXP = '0 0 0 ? * * *';
    @isTest
    private static void LeadScheduleableTest(){
        List<Lead> leads = new List<Lead>();
        for(integer i=0; i<500; i++){
            if (i<250){
                leads.add(new Lead(LastName='Rogers', Company='Salesforce'));
            } else{
                leads.add(new Lead(LastName='Rogers', Company='Salesforce',
LeadSource='Other'));
            }
        }
        insert leads;

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

        List<Lead> updateLeads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = 'Dreamforce'];
        System.assertEquals(200, updateLeads.size(), 'ERROR: Atleast 1
record not updated correctly');
```

```
        List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime
FROM CronTrigger WHERE Id = :jobID];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);
    }
}
```

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 animalLocatorT1(){
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        Test.setMock(HttpCalloutMock.class, mock);
        //Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //HttpResponse result = AnimalsLocator.getAnimalNameById();
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }
}
```

APEX SOAP CALLOUTS

```
//ParkService// --> Apex Class generated using WSDL parsing
//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 soaprktestco(){
    Test.setMock(WebServiceMock.class, new ParkServiceMock());
    String country = 'United States';
    List<String> expectedPrks = new
List<String>{'Yosemite','Sequoia','Carter Lake'};
    List<String> actualPrks = ParkLocator.country(country);
    System.assertEquals(expectedPrks, actualPrks);
  }
}
```

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, Name from Contacts)
FROM Account WHERE Id = :accountId Limit 1];
      return result;
   }
}
```

```
//AccountManagerTest//
@isTest
private class AccountManagerTest {
   @isTest
   static void testGetAccount(){
      Account a = new Account(Name='TestAcc');
      insert a;
      Contact c = new Contact(AccountId=a.Id, FirstName='Test',
LastName='Test');
      insert c;

      RestRequest request = new RestRequest();
      request.requesturi =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+a.i
d+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
```

```
        Account myAcc = AccountManager.getAccount();
        System.assert(myAcc != NULL);
        System.assertEquals('TestAccount', myAcc.Name);
    }
}
```

APEX SPECIALIST

Challenge 2: AUTOMATE RECORD CREATION

//MaintenanceRequest//

```
trigger MaintenanceRequest on Case (before update, after update) {
   if(trigger.isUpdate && trigger.isAfter){
      MaintenanceRequestHelper.updateWorkOrders(trigger.new,
trigger.OldMap);
   }
   // ToDo: Call MaintenanceRequestHelper.updateWorkOrders
}
```

//MaintenanceRequestHelper//

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCase) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCase.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }

      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = 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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Challenge 3: SYNCHRONIZE SALESFORCE DATA WITH AN EXTERNAL SYSTEM

```
//WarehouseCalloutService//
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {

  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  //@future(callout = True)
  public static void runWarehouseEquipmentSync(){
    //System.debug('Go into runWarehouseEquipmentSync');
```

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

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

        List<Product2> listProduct2 = new List<Product2>();
        System.debug(response.getStatusCode());
        if(response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)
JSON.deserializeUntyped(response.getBody());
            system.debug('~~ '+jsonResponse);
            //System.debug(response.getBody());

            for (Object jsonR : jsonResponse){
                Map<String, Object> mapJson = (Map<String, Object>)jsonR;
                Product2 product2 = new product2();
                product2.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
                product2.Cost__c = (Integer) mapJson.get('cost');
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');
                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                listProduct2.add(product2);
            }
            if(listProduct2.size() > 0){
                upsert listProduct2;
```

```
            //System.debug('Your equipment was synced with the warehouse
one');
          }
       }
    }

    public static void execute(QueueableContext context){
       //System.debug('Start');
       runWarehouseEquipmentSync();
       //System.debug('End');
   }
}
```

Challenge 4: SCHEDULE SYNCHRONIZATION

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

Challenge 5: TEST AUTOMATION LOGIC

```
//MaintenanceRequestHelper//
/-->Updated Step 2 Code to exclude if/else condition and ran if condition
only <--/
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCase) {
       Set<Id> validIds = new Set<Id>();
       For (Case c : updWorkOrders){
          if (nonUpdCase.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
```

```
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                validIds.add(c.Id);
            }
        }
    }

    if (!validIds.isEmpty()){
        Map<Id,Case> closedCases = 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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
```

```
    }
}
```

```
//MaintenanceRequestHelperTest//
@isTest
public with sharing class MaintenanceRequestHelperTest {
    private static Vehicle__c createVehicle(){
        Vehicle__c vechicle = new Vehicle__c(name = 'Testing Vechicle');
        return vechicle;
    }

    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing Equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = TRUE);
        return equipment;
    }

    private static Case createMaintenanceRequest(id vechicleId, id
equipmentId){
        case cse = new case(Type='Repair', Status='New', Origin='Web',
                            Subject='Testing subject', Equipment__c = equipmentId,
                            Vehicle__c = vechicleId);
        return cse;
    }

    private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
```

```
        Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
  }

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

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

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

    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
    update createdCase;
    test.stopTest();

    Case newCase = [Select id,
                    subject,
                    type,
                    Equipment__c,
                    Date_Reported__c,
```

```
                Vehicle__c,
                Date_Due__c
               from case
               where status ='New'];

    Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newCase.Id];
    list<case> allCase = [select id from case];
    system.assert(allCase.size() == 2);

    system.assert(newCase != null);
    system.assert(newCase.Subject != null);
    system.assertEquals(newCase.Type, 'Routine Maintenance');
    SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
  }

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

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

    case createdCase =
createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;
```

```
        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;

        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
[select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =
:createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c>
equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();
```

```
        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert caseList;

        for(integer i = 0; i < 300; i++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(eq
uipmentList.get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                              from case
                              where status ='New'];
```

```
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in:
oldCaseIds];

        system.assert(newCase.size() == 300);

        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

Challenge 6: TEST CALLOUT LOGIC

```
//WarehouseCalloutServiceMock//
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json;charset=UTF-8');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":fals
e,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id
":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"C
ooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d
66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
```

```
    }
}
```

```
//WarehouseCalloutServiceTest//
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @IsTest
    private static void testWarehouseCallout(){
        test.startTest();
        test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        System.enqueueJob(new WarehouseCalloutService());
        test.stopTest();

        List<Product2> listProduct2 = new List<Product2>();
        listProduct2 = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, listProduct2.size());
        System.assertEquals('55d66226726b611100aaf741',
listProduct2.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
listProduct2.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
listProduct2.get(2).ProductCode);

    }
}
```

Challenge 7: TEST SCHEDULING LOGIC

```
//WarehouseSyncScheduleTest//
@isTest
public with sharing class WarehouseSyncScheduleTest {
   // implement scheduled code here
   @isTest
   static void Scheduletest(){
      String scheduleTime = '0 0 0 ? * * *';
      test.startTest();
      test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
      String jobId = System.schedule('Warehouse Time to Schedule Test',
scheduleTime, new WarehouseSyncSchedule());
      CronTrigger c = [SELECT State FROM CronTrigger WHERE Id = :jobId];
      System.assertEquals('WAITING', string.valueOf(c.State), 'JobId does
not match');
      test.stopTest();
   }
}
```

> Understanding in Apex Modules and Superbadge

1. Apex Trigger: Trigger is used to perform operation when specific condition and event is true. All trigger are Bulk Trigger which can be used to perform single operation or multiple record update and insertion.
2. Apex Testing: Apex class that are implemented has to be tested for bugs and error it can be done using 'Execute Anonymous Window' or using @isTest annotations.
3. Asynchronous Apex: Asynchronous Apex are Apex class that work in background. It can scheduled, executed in batch, Queueable or be implemented in future.
4. Apex Integration Services: In apex there are Web Integration services which allows us to implement REST, Http or SOAP callout using apex class and add new remote site setting to required site.
5. Apex Specialist: In this Superbadge, I worked on Automating Record creation of Maintenance request based upon Repair or Routine Maintenance using Apex Trigger and Apex Class and also work on Syncronizing salesforce data with external data for Warehouse Service and scheduling of Warehouse Synchronize Service and Implement Apex Testing and Mock Class to verify if Apex class works as desired.