Salesforce Developer Catalyst

1. Apex Triggers

1.1 Get Started with Apex Triggers

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

1.2 Bulk Apex Triggers

```
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;
    }
}
```

2. Apex Testing

2.1 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
public class TestVerifyDate {

    @isTest static void test1(){
        Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('01/03/2022'));
        //Date -> MM/DD/YYYY;
        System.assertEquals(Date.parse('01/03/2022'), d);
    }

    @isTest static void test2(){
        Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('03/03/2022'));
        System.assertEquals(Date.parse('01/31/2022'), d);
    }
}
```

2.2 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
    public static void testContact(){
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        Database.SaveResult res = Database.insert(ct, false);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed
for DML',
                            res.getErrors()[0].getMessage());
    }
}
```

2.3 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;
    }
}
```

3. Asynchronous Apex

3.1 Use Future Methods

AccountProcessor

```
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accList = [Select Id, Number_Of_Contacts__c,
                                 (Select Id from Contacts) from Account where Id in :
accountIds];
            For(Account acc: accList){
                acc.Number_Of_Contacts__c = acc.Contacts.size();
            }

        update accList;
    }

}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
        Account a =new Account();
        a.Name = 'Test Account';
        insert a;

        Contact con = new Contact();
        con.FirstName = 'Rajat';
        con.LastName = 'Prasad';
        con.AccountId = a.Id;

        insert con;

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

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

        Account acc = [Select Number_Of_Contacts__c from Account where Id
=: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c), 1);
    }
}
```

3.2 Use Batch Apex

LeadProcessor

```
public class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
        return Database.getQueryLocator([Select LeadSource From Lead]);
    }

    public void execute(Database.BatchableContext bc, List<Lead> leads){
        // process each batch of records
        for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
        // execute any post-processing operations
    }
}
```

LeadProcessorTest

```
@isTest
private class LeadProcessorTest {

    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();

        for (Integer counter=0;counter<200;counter++) {
            Lead lead = new Lead();
                            lead.FirstName = 'FirstName';
            lead.LastName = 'LastName';
            lead.Company = 'demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new leadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }
}
```

3.3 Control Processes with Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
          private Contact c;
    private String state;
    public AddPrimaryContact (Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }

    public void execute(QueueableContext context)
    {
        List<Account> ListAccount = [SELECT ID, Name ,(Select
id,FirstName,LastName from contacts) FROM Account WHERE Billingstate
= :state LIMIT 200];
        List<Contact> lstContact = new List<Contact>();
        for ( Account acc: ListAccount)
        {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId = acc.Id;
            lstContact.add(cont);
        }

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

AddPrimaryContactTest

```
@isTest
```

```
public class AddPrimaryContactTest
{
   @isTest static void TestList()
   {
      List<Account> Teste = new List<Account>();
      for (Integer i=0; i<50; i++)
      {
         Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
      }
      for (Integer j=0; j<50; j++)
      {
         Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
      }
      insert Teste;

      Contact co = new Contact();
      co.FirstName='demo';
      co.LastName='demo';
      insert co;
      String state = 'CA';

      AddPrimaryContact apc = new AddPrimaryContact(co, state);
      Test.startTest();
      System.enqueueJob(apc);
      Test.stopTest();
   }
}
```

3.4 Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```
public class DailyLeadProcessor implements Schedulable {
   public void execute(SchedulableContext SC){
      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null
limit 200];
        for (Lead l: LeadObj) {
            l.LeadSource= 'Dreamforce';
            update l;
        }
    }
}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
    static testMethod void testDailyLeadProcessor() {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> Lista = new List<Lead>();
        for (Integer i = 0; i < 200; i++) {
            Lista.add(new Lead (LastName='Dreamforce'+i, Company='Test1
Inc.', Status='Open - Not Contacted'));
        }
        insert Lista;

        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
    }
}
```

4. Apex Integration Services

4.1 Apex REST Callouts

AnimalLocator

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'
+ x);
        req.setMethod('GET');
        Map<String, Object> animal = new Map<String, Object>();
        HttpResponse res = http.send(req);
        if(res.getStatusCode() == 200){
            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) results.get('animal');
        }
        return (String)animal.get('name');
    }

}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPResponse respond(HTTPRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals":
["lion","fox","bear","panda","snake","raccoon"]}');
        response.setStatusCode(200);
        return response;
    }

}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {
   @isTest static void AnimalLocatorMock1(){
      try{
         Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

         String result = AnimalLocator.getAnimalNameById(1);
         String expectedResult = 'fox';
         System.assertEquals(result, expectedResult);
      }
      catch (exception e){
         System.debug('The following exception has occured: '+
e.getMessage());
      }
   }

}
```

4.2 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 String[] country(String theCountry){
        ParkService.ParksImplPort parkSvc = new
ParkService.ParksImplPort();
        return parkSvc.byCountry(theCountry);
    }
}
```

ParkLocatorTest

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone','Mackinac National
Park','Yosemite' };
        System.assertEquals(parks, result);
    }
}
```

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 responseNam,
        String responseType
   ){
   ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
   response_X.return_x = new List<String>{'Yellowstone','Mackinac National
Park','Yosemite' };
   response.put('response_x',response_x);
   }
}
```

4.3 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 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 2 : Automate Record Creation

2.1 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;
      }
   }
}
```

2.2 MaitenanceRequest

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

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

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

   }

}
```

Challenge 3 : Synchronize Salesforce data with an external system

3.1 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();
   }

}
```

Open execute anonymous window (CTRl+E) and run this method ,

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

Challenge 4 : Schedule synchronization

4.1 WarehouseSyncSchedule

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

Challenge 5 : Test automation logic

5.1 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);
    }
}
```

5.2 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));
                }

                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;
        }
    }
}
```

5.3 MaintenanceRequest

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

Challenge 6 : Test callout logic

6.1 WarehouseCalloutService

```
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);
            }

        }
    }
}
```

6.2 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]);
    }
}
```

6.3 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":fals
e,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Challenge 7 : Test scheduling logic

7.1 WarehouseSyncSchedule

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

7.2 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 ');
    }
}
```

Process Automation Specialist SuperBadge

Steps based project and there is no code in it.

****