

# Salesforce Developer Catalyst Self-Learning & Super Badges

---

## ➤ Salesforce Developer-Self Learning

- 1) Salesforce Fundamentals & User Setup
  - 2) Relationships & ProcessAutomation
  - 3) Flows & Security
  - 4) Apex,Testing And Debugging
  - 5) Integration
- 

## ➤ Apex Specialist - Superbadge

### 1) Apex Triggers

- **Get started with apex triggers**

- ➡ **Code :**

```
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**

- ➡ **Code :**

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

```
//If an opportunity is inserted or updated with a stage of 'Closed Won'
// add a task created with the subject 'Follow Up Test Task'.
for (Opportunity opp : Trigger.new)
{
    //add a task with subject 'Follow Up Test Task'.
    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

- **Get Started With Apex Triggers**

⇒ **Code :**

```
@isTest
private class TestVerifyDate {
    static testMethod void TestVerifyDate() {
        VerifyDate.CheckDates(System.today(),System.today().addDays(10));
        VerifyDate.CheckDates(System.today(),System.today().addDays(78));
    }
}
```

- **Test Apex Triggers**

⇒ **Code :**

```
@IsTest
public class TestRestrictContactByName {
    @IsTest static void createBadContact(){

        Contact c=new Contact(Firstname='John',LastName='INVALIDNAME');

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

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

```
}
```

- **Creating Test Data For Apex Tests**

- ⇒ **Code :**

```
public class RandomContactFactory{
    public static List<Contact> generateRandomContacts(integer n,stringLastName){
        integer n1=n;
        List<contact> c1 = new list<contact>();
        list<contact> c2 =new list<contact>();
        c1 = [select FirstName from Contact Limit : n1];
        integer i=0;
        for(contact cnew : c1){
            contact cnew1 = new contact();
            cnew1.firstname = cnew.firstname + i;
            c2.add(cnew1);
            i++;
        }
        return c2;
    }
}
```

### 3) Asynchronous Apex

- **Use Future Methods**

- ⇒ **Code :**

```
//AccountProcessor class

    public class AccountProcessor {
        @future
        public static void countContacts(List<Id> accountIds){
            List<Account> accounts = [Select Id, Name from Account Where Id IN : ];
            List<Account> updatedAccounts = new List<Account>();
            for(Account account : accounts){
                account.Number_of_Contacts__c = [Select count() from Contact Where AccountId
=: account.Id];
                System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
                updatedAccounts.add(account);
            }
        }
    }
```

```

    }
    update updatedAccounts;
}

}

```

//AccountProcessorTest Class

```

@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name
= 'Test Account';
        Insert a;
        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName = 'Willie';
        c.AccountId = a.Id
;
        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id
;
        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }
}

```

- **Use Batch Apex**

⇒ **Code :**

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

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        System.debug('count = '+count);
    }
}
```

- **Control Processes With Queueable Apex**

⇒ **Code :**

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
```

```

account where account.BillingState = :this.state limit 200]);
    List<contact> c_lst = new List<contact>();
    for(account a: acc_lst) {
        contact c = new contact();
        c = this.c.clone(false, false, false, false);
        c.AccountId = a.Id;
        c_lst.add(c);
    }
    insert c_lst;
}
}

```

Add primaryContactTest :

```

    public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }
}

```

```
}
```

- Schedule Jobs Using the Apex Scheduler

⇒ Code :

Apex Class

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = "];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

Apex Test Class

@isTest

```
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';

    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();

        for(Integer i = 0; i < 200; i++){
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = "", Company = 'Test
            Company ' + i, Status = 'Open - Not Contacted');
```

```

        leads.add(lead);
    }

    insert leads;

    Test.startTest();
    // Schedule the test job
    String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());

    // Stopping the test will run the job synchronously
    Test.stopTest();
}
}

```

## 4) Apex Integration Services

- **Apex REST Callouts**

- ⇒ **Code :**

```

AnimalLocator
public class AnimalLocator {
    public class cls_animal {
        public Integer id;
        public String name;
        public String eats;
        public String says;
    }
}

public class JSONOutput{
    public cls_animal animal;

    //public JSONOutput parse(String json){
    //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
    //}
}

public static String getAnimalNameById (Integer id) {

```



```

    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
    //request.setHeader('id', String.valueOf(id)); -- cannot be used in this challenge :)
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    system.debug('response: ' + response.getBody());
    //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
    jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
    //Object results = (Object) map_results.get('animal');
        system.debug('results= ' + results.animal.name);
    return(results.animal.name);
}

```

```

}

```

AnimalLocatorMock

@IsTest

global class AnimalLocatorMock implements HttpCalloutMock {

```

    global HTTPResponse respond(HTTPRequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueOf(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        return response;
    }
}

```

```

}

```

AnimalLocatorTest.cls

```

@Test
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //HttpResponse response = AnimalLocator.getAnimalNameById(1);
        String s = AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}

```

- **Apex Soap Callouts**

⇒ **Code :**

Apex Service

//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;
    }
}
}
}

```

Apex Class

```

public class ParkLocator {
    public static String[] country(String country){

```

```

        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}

```

## Apex Test Class

```

@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}

```

## Apex Mock Test Class

```

@isTest
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;
    }
}

```

```

        response.put('response_x', response_x);
    }
}

```

- **Apex Web Services**

⇒ **Code :**

```

AccountManagerTest/////
    @isTest
    private class AccountManagerTest {
private static testMethod void getAccountTest1() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts';
    request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount != null);
    System.assertEquals('Test record', thisAccount.Name);

}

// Helper method
static Id createTestRecord() {
    // Create test record
    Account TestAcc = new Account(
        Name='Test record');
    insert TestAcc;
    Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
    return TestAcc.Id
;
}

```

```
}
}
```

AccountManager/////

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                        FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

## ★ Skills Learnt During Completion Of The Superbadge

- How to Automate record creation using Apex triggers

⇒ Code :

### MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
```

```

        validIds.add(c.Id);

    }
}

if (!validIds.isEmpty()){
    List<Case> newCases = new List<Case>();
    Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
    AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds GROUP
BY Maintenance_Request__c];

    for (AggregateResult ar : results){
        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
    }

    for(Case cc : closedCasesM.values()){
        Case nc = new Case (
            ParentId = cc.Id,
            Status = 'New',
            Subject = 'Routine Maintenance',
            Type = 'Routine Maintenance',
            Vehicle__c = cc.Vehicle__c,
            Equipment__c =cc.Equipment__c,
            Origin = 'Web',
            Date_Reported__c = Date.Today()

        );
    }
}

```

```

        If (maintenanceCycles.containsKey(cc.Id)){
            nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
        } 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.apxt**

```

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 using asynchronous REST callouts.**



⇒ **Code:**

**WarehouseCalloutService.apxc :-**

```
public with sharing class WarehouseCalloutService implements Queueable {  
    private static final String WAREHOUSE_URL = 'https://th-superbadge-  
apex.herokuapp.com/equipment';
```

//class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.

//The callout's JSON response returns the equipment records that you upsert in Salesforce.

```
@future(callout=true)  
public static void runWarehouseEquipmentSync(){  
    Http http = new Http();  
    HttpRequest request = new HttpRequest();  
  
    request.setEndpoint(WAREHOUSE_URL);  
    request.setMethod('GET');  
    HttpResponse response = http.send(request);  
  
    List<Product2> warehouseEq = new List<Product2>();  
  
    if (response.getStatusCode() == 200){  
        List<Object> jsonResponse =  
(List<Object>)JSON.deserializeUntyped(response.getBody());  
        System.debug(response.getBody());  
  
        //class maps the following fields: replacement part (always true), cost, current  
inventory, lifespan, maintenance cycle, and warehouse SKU  
        //warehouse SKU will be external ID for identifying which equipment records to  
update within Salesforce  
        for (Object eq : jsonResponse){  
            Map<String,Object> mapJson = (Map<String,Object>)eq;  
            Product2 myEq = new Product2();  
            myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');  
            myEq.Name = (String) mapJson.get('name');
```

```

        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
        myEq.Cost__c = (Integer) mapJson.get('cost');
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
        myEq.ProductCode = (String) mapJson.get('_id');
        warehouseEq.add(myEq);
    }

    if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse one');
    }
}

}

public static void execute (QueueableContext context){
    runWarehouseEquipmentSync();
}

}

execute anonymous window ( CTRL+E ) ,System.enqueueJob(new
WarehouseCalloutService());

```

- **Schedule synchronization using Apex code.**

⇒ **Code:**

**WarehouseSyncShedule.apxc :-**

```

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

```

- **Test automation logic to confirm Apex trigger side effects**

➡ **Code:**

**MaintenanceRequestHelperTest.apxc :-**

@istest

public with sharing class MaintenanceRequestHelperTest {

```
private static final string STATUS_NEW = 'New';
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';
private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';
```

```
PRIVATE STATIC Vehicle__c createVehicle(){
    Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
    return Vehicle;
}
```

```
PRIVATE STATIC Product2 createEq(){
    product2 equipment = new product2(name = 'SuperEquipment',
        lifespan_months__C = 10,
        maintenance_cycle__C = 10,
        replacement_part__c = true);
    return equipment;
}
```

```
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cs = new case(Type=REPAIR,
        Status=STATUS_NEW,
        Origin=REQUEST_ORIGIN,
        Subject=REQUEST_SUBJECT,
        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cs;
}
```

```

PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
    Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                Maintenance_Request__c = requestId);
    return wp;
}

```

```

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

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

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

    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c,
Vehicle__c, Date_Due__c
                  from case
                  where status =:STATUS_NEW];

    Equipment_Maintenance_Item__c workPart = [select id

```

```
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newReq.Id];
```

```
system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}
```

```
@istest
```

```
private static void testMaintenanceRequestNegative(){
```

```
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
```

```
    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;
```

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

```
    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,
emptyReq.Id);
    insert workP;
```

```
    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();
```

```
    list<case> allRequest = [select id
                            from case];
```

```
    Equipment_Maintenance_Item__c workPart = [select id
```

```
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :emptyReq.Id];
```

```
system.assert(workPart != null);
system.assert(allRequest.size() == 1);
}
```

```
@istest
```

```
private static void testMaintenanceRequestBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
    list<case> requestList = new list<case>();
    list<id> oldRequestIds = new list<id>();

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

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

    for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
    }
    insert workPartList;

    test.startTest();
    for(case req : requestList){
```

```

        req.Status = CLOSED;
        oldRequestIds.add(req.Id);
    }
    update requestList;
    test.stopTest();

    list<case> allRequests = [select id
                            from case
                            where status =: STATUS_NEW];

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

    system.assert(allRequests.size() == 300);
}
}

```

#### **MaintenanceRequestHelper.apxc :-**

```

public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }

        if (!validIds.isEmpty()){

```

```

List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds GROUP
BY Maintenance_Request__c];

for (AggregateResult ar : results){
    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
}

for(Case cc : closedCasesM.values()){
    Case nc = new Case (
        ParentId = cc.Id,
        Status = 'New',
        Subject = 'Routine Maintenance',
        Type = 'Routine Maintenance',
        Vehicle__c = cc.Vehicle__c,
        Equipment__c =cc.Equipment__c,
        Origin = 'Web',
        Date_Reported__c = Date.Today()

    );

    If (maintenanceCycles.containsKey(cc.Id)){
        nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
    }

    newCases.add(nc);
}

```



```

insert newCases;

List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
        Equipment_Maintenance_Item__c wpClone = wp.clone();
        wpClone.Maintenance_Request__c = nc.Id;
        ClonedWPs.add(wpClone);
    }
}
insert ClonedWPs;
}
}
}

```

#### **MaintenanceRequest.apxt :-**

```

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

```

- **Test integration logic using callout mocks**

⇒ **Code:**

#### **WarehouseCalloutService.apxc :-**

```

public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

```

```

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

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

List<Product2> warehouseEq = new List<Product2>();

if (response.getStatusCode() == 200){
    List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
    System.debug(response.getBody());

    for (Object eq : jsonResponse){
        Map<String,Object> mapJson = (Map<String,Object>)eq;
        Product2 myEq = new Product2();
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
        myEq.Cost__c = (Decimal) mapJson.get('lifespan');
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
        warehouseEq.add(myEq);
    }

    if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse one');
        System.debug(warehouseEq);
    }

}
}
}

```

### WarehouseCalloutServiceTest.apxc :-

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

### WarehouseCalloutServiceMock.apxc :-

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

        response.setBody('[{ "_id": "55d66226726b611100aaf741", "replacement": false, "quantity": 5, "name": "Generator 1000 kW", "maintenanceperiod": 365, "lifespan": 120, "cost": 5000, "sku": "100003" }]');
        response.setStatusCode(200);
        return response;
    }
}
```

- Test scheduling logic to confirm action gets queued

➡ **Code:**

**WarehouseSyncSchedule.apxc :-**

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

```
        WarehouseCalloutService.runWarehouseEquipmentSync();  
    }  
}
```

**WarehouseSyncScheduleTest.apxc :-**

```
@isTest  
public class WarehouseSyncScheduleTest {  
  
    @isTest static void WarehousescheduleTest(){  
        String scheduleTime = '00 00 01 * * ?';  
        Test.startTest();  
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());  
        String jobID=System.schedule('Warehouse Time To Schedule to Test',  
scheduleTime, new WarehouseSyncSchedule());  
        Test.stopTest();  
        //Contains schedule information for a scheduled job. CronTrigger is similar to a  
cron job on UNIX systems.  
        // This object is available in API version 17.0 and later.  
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];  
        System.assertEquals(jobID, a.Id,'Schedule ');  
  
    }  
}
```

---

## ➤ **Process Automation Specialist - SuperBadge**

- 1) Formulas And Validations
- 2) Salesforce Flow
- 3) Leads & Opportunities For Lightning Experience

### ★ **Skills Learnt During Completion Of Super badge**

- Automate lead ownership using assignment rules
- Enforce data integrity with formula fields and validation rules
- Create a custom object in a master-detail relationship to a standard object
- Define an opportunity sales process using stages, record types, and validation rules
- Automate business processes to send emails, create related records, and submit opportunities for approval
- Create a flow to display dynamic information on a Lightning record page
- Create a process to evaluate and update records