

## **APEX TRIGGERS**

### **AccountAddressTrigger.apxt**

```
trigger AccountAddressTrigger on Account (before insert, before update) {  
    for(Account account:Trigger.New){  
        if(account.Match_Billing_Address__c == True){  
            account.ShippingPostalCode = account.BillingPostalCode;  
        }  
    }  
}
```

### **Bulk Apex Triggers**

#### **ClosedOpportunityTrigger.apxt**

```
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

### **VerifyDate.apxc**

```
public class VerifyDate {  
    //method to handle potential checks against two dates  
    public static Date CheckDates(Date date1, Date date2) {  
        //if date2 is within the next 30 days of date1, use date2. Otherwise use the end  
of the month  
        if(DateWithin30Days(date1,date2)) {  
            return date2;  
        } else {  
            return SetEndOfMonthDate(date1);  
        }  
    }  
  
    //method to check if date2 is within the next 30 days of date1  
    @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {  
        //check for date2 being in the past  
        if( date2 < date1) { return false; }  
  
        //check that date2 is within (>=) 30 days of date1  
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1  
        if( date2 >= date30Days ) { return false; }  
        else { return true; }  
    }  
  
    //method to return the end of the month of a given date  
    @TestVisible private static Date SetEndOfMonthDate(Date date1) {  
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());  
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);  
        return lastDay;  
    }  
}
```

## TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'),D);
    }
    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'),D);
    }
    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false,flag);
    }
    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false,flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true,flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = verifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}
```

## Test Apex Triggers

### RestrictContactByName.apxt

```
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.apxc

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

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

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
    }
}
```

## Create Test Data for Apex Test

### RandomContactFactory.apxc

```
public class RandomContactFactory {  
  
    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){  
  
        List<Contact> contacts = new List<Contact>();  
  
        for(Integer i=0;i<numcnt;i++){  
  
            Contact cnt = new Contact(FirstName = 'Test'+i, LastName =  
lastname);  
  
            contacts.add(cnt);  
  
        }  
  
        return contacts;  
  
    }  
  
}
```

## **ASYNCHRONOUS APEX**

### **AccountProcessor.apxc**

```
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountId_Lst) {

        Map<Id,Integer> account_cno = new Map<Id,Integer>();
        List<account> account_Lst_all = new List<account>([select id, (select id from
contacts) from account]);
        for(account a:account_Lst_all) {
            account_cno.put(a.id,a.contacts.size()); //populate the map

        }

        List<account> account_Lst = new List<account>(); // list of account that we will
upsert

        for(Id accountId : accountId_Lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c = account_cno.get(accountId);
                account_Lst.add(acc);
            }

        }
        upsert account_Lst;
    }
}
```

## AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;

        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
        AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from account
where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }
}
```

## Uses Batch Apex

### LeadProcessor.apxc

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

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

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

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

### LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

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



```

        l.Status = 'somestatus';
        l_lst.add(l);
    }
    insert l_lst;

    test.startTest();

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

```

## Control Processes with Queueable Apex

### AddPrimaryContact.apcx

```

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

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

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

```

## AddPrimaryContactTest.apxc

```
@IsTest
public class AddPrimaryContactTest {

    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }
}
```

## Schedule Jobs Using the Apex Scheduler

### DailyLeadProcessor.apxc

```
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 new leads;
    }
}

```

### DailyLeadProcessorTest.apxc

```

    @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();
        }
    }
}

```

## APEX INTEGRATION SERVICES

### Apex REST Callouts

#### AnimalLocator.apxc

```
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> result = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) result.get('animal');
        }
        return (String)animal.get('name');
    }
}
```

#### AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock(){
        Test.setmock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(3);
        string expectedResult = 'chicken';
        System.assertEquals(result,expectedResult);
    }
}
```

#### AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock{
```

```

global HTTPResponse respond(HttpRequest request){
    httpResponse response = new HttpResponse();
    response.setHeader('Content-type', 'application/json');
    response.setBody('{"animals":["majestic badger","fluffy bunny","scary bear","chicken"]}');
    response.setStatusCode(200);
    return response;
}
}

```

## Apex SOAP Callouts

### ParkService.apxc

//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.apxc

```

public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}

```

### ParkLocatorTest.apxc

```

@isTest
private class ParkLocatorTest {

```

```

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

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

```

### **ParkServiceMock.apxc**

```

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

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

```

### **AsyncParksServices.apxc**

//Generated by wsdl2apex

```

public class AsyncParkService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);

```

```

        return response.return_x;
    }
}

public class AsyncParksImplPort {
    public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
    public Map<String,String> inputHttpHeaders_x;
    public String clientCertName_x;
    public Integer timeout_x;
    private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
    public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
    ParkService.byCountry request_x = new ParkService.byCountry();
    request_x.arg0 = arg0;
    return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
    this,
    request_x,
    AsyncParkService.byCountryResponseFuture.class,
    continuation,
    new String[]{endpoint_x,
    "",
    'http://parks.services/',
    'byCountry',
    'http://parks.services/',
    'byCountryResponse',
    'ParkService.byCountryResponse'}
    );
}
}
}
}

```

## Apex Web Services

### AccountManager.apxc

```

@RestResource(urlMapping='/Accounts/*/contacts')
Global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        //Grab the accountId from end of URL
    }
}

```



```

        String accountId = request.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [select Id,Name,(select Id,Name from Contacts) from Account where Id =
:accountId];
        system.debug('Account and Related Contacts->>>>'+acc);
        return acc;
    }
}

```

### AccountManagerTest.apxc

```

@isTest
private class AccountManagerTest {
    //Helper method to create dummy record
    static Id createTestRecord(){
        //Create test record
        Account TestAcc = new Account(Name='Test Account', Phone='8786757657');
        insert TestAcc;
        List<Contact> conList = new List<Contact>();
        Contact TestCon = new Contact();
        for(Integer i=1;i<=3;i++){
            TestCon.LastName = 'Test Contact'+i;
            TestCon.AccountId = TestAcc.Id;
            //conList.add(TestCon);
            insert conList;//Its not best practice but I have use it for testing purposes
        }
        //insert conList;
        //insert TestAcc;
        return TestAcc.Id;
    }

    //Method to test getAccount()
    @isTest static void getAccountTest(){
        Id recordId = createTestRecord();
        //setup a test request
        RestRequest request = new RestRequest();
        //set request properties
        request.requestURI = 'https://yourInstance.salesforce.com/services/apexrest/Accounts/'
+ recordId + '/contacts';
        request.httpMethod = 'GET';
        // Finally, assign the request to RestContext if used
    }
}

```

```

RestContext.request = request;
//End test setup

//Call the method
Account thisAcc = AccountManager.getAccount();
//Verify the result
system.assert(thisAcc != null);
system.assertEquals('Test Account', thisAcc.Name);
//system.assertEquals(3, thisAcc.Contact__c.size()); how to get this
}
}

```

## **APEX SPECIALIST SUPER BADGE**

### **Challenge 2**

#### **AutomatedRecord Creation**

#### **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);
                }
            }
        }
    }
    //When an existing maintenance request of type Repair or Routine Maintenance is closed,
    //create a new maintenance request for a future routine checkup.
    if (!validIds.isEmpty()){
        Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle_c, Equipmentc,
Equipmentr.Maintenance_Cycle_c,
                                (SELECT Id,Equipment_c,Quantityc FROM
Equipment_Maintenance_Items_r)
                                FROM Case WHERE Id IN :validIds]);
    }
}

```

```

    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
    //calculate the maintenance request due dates by using the maintenance cycle defined
on the related equipment records.
    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 multiple pieces of equipment are used in the maintenance request,
        //define the due date by applying the shortest maintenance cycle to today's date.
        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.Equipmentr.maintenance_Cycle_c);
        }
        newCases.add(nc);
    }
    insert new cases;
    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
}
}
}

```

### **MaintenanceRequest.apxt**

```

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**

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

```

public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    //Write a 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(){
        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> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
    }
}

```

```

    if (response.getStatusCode() == 200){
        List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
        System.debug(response.getBody());
        //class maps the following fields:
        //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
        for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            //cost
            product2.Cost__c = (Integer) mapJson.get('cost');
            //current inventory
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');
            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }
        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}
}

```

### **Challenge 4**

**Schedule synchronization using Apex code**

### **WarehouseSyncSchedule.apxc :-**

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

### **Challenge 5**

#### **Test automation logic**

### **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);
                }
            }
        }
        //When an existing maintenance request of type Repair or Routine Maintenance is closed,
        //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle_c, Equipmentc,
Equipmenttr.Maintenance_Cycle_c,
                                (SELECT Id,Equipment_c,Quantityc FROM
Equipment_Maintenance_Items_r)
                                FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
```

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

```
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 multiple pieces of equipment are used in the maintenance request,
    //define the due date by applying the shortest maintenance cycle to today's date.
    //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.Equipmentr.maintenance_Cycle_c);
    //}
    newCases.add(nc);
}
insert new cases;
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;
}
}
}

```

### **MaintenanceRequest.apxc :-**

```

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

```

### **MaintenanceRequestHelperTest.apxc :-**

```

@isTest
public with sharing class MaintenanceRequestHelperTest {
    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle_C(name = 'Testing Vehicle');
        return vehicle;
    }
    // createEquipment
    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;
    }
    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
            Status='New',
            Origin='Web',
            Subject='Testing subject',

```



```

        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cse;
}
// createEquipmentMaintenanceItem
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++){

equipmentMaintenanceItem.add(createEquipmentMaintenanceItem(equipmentList.get(i).id, caseList.get(i).id));
}
insert equipmentMaintenanceItem;
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

**WarehouseCalloutService.apxc :-**

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    //Write a 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(){
        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> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
```

```

        product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
        //warehouse SKU
        product2.Warehouse_SKU__c = (String) mapJson.get('sku');
        product2.Name = (String) mapJson.get('name');
        product2.ProductCode = (String) mapJson.get('_id');
        product2List.add(product2);
    }
    if (product2List.size() > 0){
        upsert product2List;
        System.debug('Your equipment was synced with the warehouse one');
    }
}
}
}
public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}
}

```

### **WarehouseCalloutServiceTest.apxc :-**

```

@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @IsTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();
        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];
        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}

```

### WarehouseCalloutServiceMock.apxc :-

```
@isTest
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');

        response.setBody('{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b
611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b61110
0aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

### Challenge 7

#### Test scheduling logic

### WarehouseSyncSchedule.apxc :-

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

### WarehouseSyncScheduleTest.apxc :-

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
```

```

String scheduleTime = '00 00 00 * * ? *';
Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
String jobId = System.schedule('Warehouse Time to Schedule to 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();
}
}

```

## Lightning Web Components

### Bike Card Project

#### **BikeCard.html**

```

<template>
  <div>
    <div>Name: {name}</div>
    <div>Description: {description}</div>
    <lightning-badge label={material}></lightning-badge>
    <lightning-badge label={category}></lightning-badge>
    <div>Price: {price}</div>
    <div><img src={pictureUrl}/></div>
  </div>
</template>

```

#### **BikeCard.js**

```

import { LightningElement } from 'lwc';
export default class BikeCard extends LightningElement {
  name = 'Electra X4';
  description = 'A sweet bike built for comfort.';
  category = 'Mountain';
  material = 'Steel';
  price = '$2,700';

```

```
    pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';
  }
```

### BikeCard.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <!-- The apiVersion may need to be increased for the current release -->
  <apiVersion>52.0</apiVersion>
  <isExposed>true</isExposed>
  <masterLabel>Product Card</masterLabel>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```

### Selector.html

```
<template>
  <div class="wrapper">
    <header class="header">Available Bikes</header>
    <section class="content">
      <div class="columns">
        <main class="main" >
          <b>{name}</b>
          <c-list onproductselected={handleProductSelected}></c-list>
        </main>
        <aside class="sidebar-second">
          <c-detail product-id={selectedProductId}></c-detail>
        </aside>
      </div>
    </section>
  </div>
</template>
```

### Selector.js



```

import { LightningElement, wire, track } from 'lwc';
import {
    getRecord
} from 'lightning/uiRecordApi';
import Id from '@salesforce/user/Id';
import NAME_FIELD from '@salesforce/schema/User.Name';
import EMAIL_FIELD from '@salesforce/schema/User.Email';
export default class Selector extends LightningElement {
    @track selectedProductId;
    @track error ;
    @track email ;
    @track name;
    @wire(getRecord, {
        recordId: Id,
        fields: [NAME_FIELD, EMAIL_FIELD]
    }) wireuser({
        error,
        data
    }) {
        if (error) {
            this.error = error ;
        } else if (data) {
            this.email = data.fields.Email.value;
            this.name = data.fields.Name.value;
        }
    }
    handleProductSelected(evt) {
        this.selectedProductId = evt.detail;
    }
    userId = Id;
}

```

### Selector.js-meta.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>

```

```
<target>lightning__AppPage</target>
<target>lightning__RecordPage</target>
<target>lightning__HomePage</target>
</targets>
</LightningComponentBundle>
```

## Visualforce Basics

### DisplayImage.vfp

### Create & Edit Visualforce Pages

```
<apex:page showHeader="false">
<apex:image url="https://developer.salesforce.com/files/salesforce-developer-network-
logo.png"/>
</apex:page>
```

### Use Simple Variables and Formulas

### DisplayUserInfo.vfp

```
<apex:page >
{! $User.FirstName}
</apex:page>
```

### ContactView.vfp

### Use StandardControllers

```
<apex:page standardController="Contact">
<apex:pageBlockSection >
```

```
First Name : {! Contact.FirstName} Last Name: {! Contact.LastName} Owner Email : {!  
Contact.Owner.Email}  
</apex:pageBlockSection>  
</apex:page>
```

### OppView.vfp

#### Display Records, Fields, and Tables

```
<apex:page standardController="Opportunity">  
<apex:outputField value="{! Opportunity.Name}"/>  
<apex:outputField value="{! Opportunity.Amount}"/>  
<apex:outputField value="{! Opportunity.CloseDate}"/>  
<apex:outputField value="{! Opportunity.Account.Name}"/>  
</apex:page>
```

### CreateContact.vfp

#### Input Data Using Forms

```
public class NewCaseListController {  
  
    public List<Case> getNewCases(){  
        List<Case> filterList= [Select ID , CaseNumber from Case where status = 'New']; return filterlist;  
    }  
}
```

### AccountList.vfp

#### Use StandardList Controllers

```
<apex:page standardController="Account" recordSetVar="accounts">  
<apex:repeat var="a" value="{! accounts}">  
    <li>  
        <apex:outputLink value="/{!a.ID}">  
            <apex:outputText value="{! a.Name}"></apex:outputText>  
        </apex:outputLink>  
    </li>  
</apex:repeat>  
</apex:page>
```

## Use Static Resources

### ShowImage.vfp

```
<apex:page >
<apex:image url="{! URLFOR($Resource.vfimage, 'cats/kitten1.jpg')}" />
</apex:page>
```

### NewCaseList.vfp

#### Create & Use Custom Controllers

```
<apex:page controller="NewCaseListController">
<apex:repeat var="case" value="{!newCases}">
<apex:outputLink value="{!case.ID}">]
<apex:outputText value="{!case.CaseNumber}">
</apex:outputText>
</apex:outputLink>
</apex:repeat>
</apex:page>
```

### NewCaseListController.apxc

```
public class NewCaseListController {

    public List<Case> getNewCases(){
        SPSGP-29053-Salesforce Developer Catalyst Self-Learning & Super Badges
        return filterlist;
    }
}
```