

APEX SPECIALIST SUPERBADGE

AccountAddressTrigger.apxt

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

    for(Account account: Trigger.new){

        if(account.Match_Billing_Addressc == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }

}
```

Explanation: AccountAddressTrigger is a apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (afterinsert, 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;
    }

}
```

Explanation:ClosedOpportunityTrigger is a apex trigger which fire trigger after inserting or updating an opportunity.

APEX TESTING

APEX SPECIALIST SUPERBADGE

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;
    }
}
```

APEX SPECIALIST SUPERBADGE

```
}
```

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'));
    }

}
```

Explanation: TestVerifyDate is a apex class to test if a date is within a proper range, and if not,

APEX SPECIALIST SUPERBADGE

returns a date that occurs at the end of the month within the range.

RestrictContactByName.apxc

```
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());
    }

}
```

Explanation:TestRestrictContactByName is a Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'.

APEX SPECIALIST SUPERBADGE

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;
    }

}
```

Explanation:RandomContactFactory is an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name.

Asynchronous Apex

AccountProcessor.apxc

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new List<Account>();
        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account
Where Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contactsc = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

APEX SPECIALIST SUPERBADGE

```
}
```

AccountProcessorTest.apxc

```
@IsTest
private class AccountProcessorTest
{ @IsTest
  Private static void testCountContacts(){
    Account newAccount = new Account(Name = 'Test Account');
    insert newAccount;

    Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
    insert newContact1;

    Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
    insert newContact2;

    List<Id> accountIds = new List<Id>();
    accountIds.add(newAccount.Id);

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

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');
```

APEX SPECIALIST SUPERBADGE

```
}

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);
}

}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0;i<200;i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random
            Status'; L_list.add(L);
        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new
```

APEX SPECIALIST SUPERBADGE

```
        LeadProcessor(); Id batchId =  
        Database.executeBatch(lp);  
        Test.stopTest();  
    }  
  
}
```

AddPrimaryContact.apxc

public class AddPrimaryContact implements

```
    Queueable{ private Contact con;  
  
    private String state;  
  
    public AddPrimaryContact(Contact con, String  
        State){ this.con = con;  
        this.state = state;  
    }  
    public void execute(QueueableContext context){  
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from  
contacts) from Account where BillingState = :state Limit 200];  
        List<Contact> primaryContacts = new List<Contact>();  
  
        for(Account acc:accounts){  
            Contact c = con.clone();  
            c.AccountId = acc.Id;  
            primaryContacts.add(c);  
        }  
  
        if(primaryContacts.size() > 0){  
            insert primaryContacts;  
        }  
    }  
}
```

AddPrimaryContactTest.apxc

APEX SPECIALIST SUPERBADGE

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

        Contact testContact = new Contact(FirstName = 'John', LastName =
        'Doe'); insert testContact;
        AddPrimaryContact addit = new addPrimaryContact(testContact,

        'CA'); Test.startTest();

        System.enqueueJob(addit);
        Test.stopTest();

        System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from
        Account where BillingState='CA')]);
    }
}
```

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx) {
        List<Lead> leadstoupdate = new List<Lead>();
        List<Lead> leads = [Select id From Lead Where LeadSource = NULL Limit 200];
        for(Lead l: leads) {
```

APEX SPECIALIST SUPERBADGE

```
        l.LeadSource = 'Dreamforce';
        leadstoupdate.add(l);
    }
    update leadstoupdate;
}
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 3 ?
2024'; static testmethod void testScheduledJob() {
    List<Lead> leads = new List<Lead>();
    for(Integer i = 0; i < 200; i++) {
        Lead l = new Lead(
            FirstName = 'First' + i,
            LastName =
            'LastName', Company =
            'The Inc'
        );
        leads.add(l);
    }
    insert leads;
    Test.startTest();
    String jobId = System.schedule('ScheduledApexTest',CRON_EXP,new
        DailyLeadProcessor()); Test.stopTest();
    List<Lead> checkleads = new List<Lead>();
    checkleads = [Select Id From Lead Where LeadSource = 'Dreamforce' and Company = 'The
Inc'];
    System.assertEquals(200,checkleads.size(),'Leads were not created');
    }
}
```

APEX SPECIALIST SUPERBADGE

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

AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    / Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        / Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
            'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
            "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

APEX SPECIALIST SUPERBADGE

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}
```

ParkLocator.apxc

```
public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new
        parkService.parksImplPort(); return park.byCountry(country);
    }
}
```

ParkLocatorMock.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)
    {
```

APEX SPECIALIST SUPERBADGE

```
parkService.byCountryResponse response_x = new parkService.byCountryResponse();
response_x.return_x = new List<String>{'Hamburg Wadden Sea National Park', 'Hainich
National Park', 'Bavarian Forest National Park'};
response.put('response_x', response_x);
}
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'Germany';
        String[] result = ParkLocator.Country(country);

        System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park', 'Hainich
National Park', 'Bavarian Forest National Park'}, result);
    }
}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {

    @HttpGet
    global static account getAccount() {

        RestRequest request =

        RestContext.request;

        String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
            request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account where id
            =
            :accountId];
    }
}
```

APEX SPECIALIST SUPERBADGE

```
List<contact> co = [select id, name from contact where account.id = :accountId];
system.debug('** a[0]= '+ a[0]);
return a[0];

}

}
```

AccountManagerTest.apxc

```
@istest
public class AccountManagerTest {
    @istest static void testGetContactsByAccountId()
    { Id recordId = createTestRecord();
      / Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
      'https:// yourInstance.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);
    }

    / Helper method
    static Id createTestRecord() {

        / Create test record
        Account accountTest = new Account(
        Name='Test record');
        insert accountTest;
        Contact contactTest = new Contact(
        FirstName='John',
        LastName='Doe',
        AccountId=accountTest.Id
        );
        return accountTest.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE

```
}  
}
```

Apex Specialist super badge

Challenge-1

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, Vehiclec,  
Equipmentc, Equipmentr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc  
FROM Equipment_Maintenance_Items_r)  
FROM Case WHERE Id IN :validIds]);  
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();  
        AggregateResult[] results = [SELECT Maintenance_Requestc,  
MIN(Equipmentr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc  
WHERE Maintenance_Requestc IN :ValidIds GROUP BY Maintenance_Requestc];  
  
        for (AggregateResult ar : results){
```

APEX SPECIALIST SUPERBADGE

```
        maintenanceCycles.put((Id) ar.get('Maintenance_Requestc'), (Decimal) ar.get('cycle'));
    }

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

        );

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

        newCases.add(nc);
    }

    insert newCases;

    List<Equipment_Maintenance_Itemc> clonedWPs = new

List<Equipment_Maintenance_Itemc>();
    for (Case nc : newCases){
        for (Equipment_Maintenance_Itemc wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Itemsr){
            Equipment_Maintenance_Itemc wpClone = wp.clone();
            wpClone.Maintenance_Requestc = nc.Id;
            ClonedWPs.add(wpClone);

        }
    }
    insert ClonedWPs;
}
```


APEX SPECIALIST SUPERBADGE

```
}
```

MaintenanceRequest.apxt

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

Challenge-2

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());
```

APEX SPECIALIST SUPERBADGE

```
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 = (Boolean) mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Month = (Integer) mapJson.get('lifespan');
        myEq.Cost = (Integer) mapJson.get('cost');
        myEq.Warehouse_SKU = (String) mapJson.get('sku');
        myEq.Current_Inventory = (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();
}
}
```

Challenge-3

WarehouseSyncSchedule.apxc

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

APEX SPECIALIST SUPERBADGE

```
}  
}
```

Challenge-4

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 VehicleC createVehicle(){  
        VehicleC Vehicle = new VehicleC(name = 'SuperTruck');  
        return Vehicle;  
    }
```

```
    PRIVATE STATIC Product2 createEq(){  
        product2 equipment = new product2(name = 'SuperEquipment',  
                                            lifespan_monthsC = 10,  
                                            maintenance_cycleC = 10,  
                                            replacement_partc = 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,  
            Equipmentc=equipmentId,  
            Vehiclec=vehicleId);
```

APEX SPECIALIST SUPERBADGE

```
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Itemc createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Itemc wp = new
Equipment_Maintenance_Itemc(Equipmenttc = equipmentId,
                            Maintenance_Requestc = requestId);

        return wp;
    }

    @istest
    private static void
    testMaintenanceRequestPositive(){ Vehiclec
    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_Itemc workP =
    createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

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

    Case newReq = [Select id, subject, type, Equipmenttc, Date_Reportedc, Vehiclec,
Date_Duec
        from case
        where status =:STATUS_NEW];
```

APEX SPECIALIST SUPERBADGE

[illegible]

APEX SPECIALIST SUPERBADGE

```
where Maintenance_Requestc = :emptyReq.Id];
```

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

```
@istest
```

```
private static void testMaintenanceRequestBulk(){  
    list<VehicleC> vehicleList = new list<VehicleC>();  
    list<Product2> equipmentList = new list<Product2>();  
    list<Equipment_Maintenance_Itemc> workPartList = new  
list<Equipment_Maintenance_Itemc>();  
    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
```

APEX SPECIALIST SUPERBADGE

```
requestList;
test.stopTest();

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

list<Equipment_Maintenance_Itemc> workParts = [select id
                                                from Equipment_Maintenance_Itemc
                                                where Maintenance_Requestc 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, Vehiclec,
Equipmentc, Equipmenttr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc
```

APEX SPECIALIST SUPERBADGE

```
FROM Equipment_Maintenance_Items_r)
        FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
    AggregateResult[] results = [SELECT Maintenance_Requestc,
MIN(Equipmenttr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc
WHERE Maintenance_Requestc IN :ValidIds GROUP BY Maintenance_Requestc];

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

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

        );

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

        newCases.add(nc);
    }

    insert newCases;

    List<Equipment_Maintenance_Itemc> clonedWPs = new
    List<Equipment_Maintenance_Itemc>();
    for (Case nc : newCases){
        for (Equipment_Maintenance_Itemc wp :
        closedCasesM.get(nc.ParentId).Equipment_Maintenance_Itemsr){
            Equipment_Maintenance_Itemc wpClone = wp.clone();
            wpClone.Maintenance_Requestc = nc.Id;
```


APEX SPECIALIST SUPERBADGE

```
        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);
    }
}
```

Challenge-5

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');
```

APEX SPECIALIST SUPERBADGE

```
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 = (Boolean) mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Month = (Integer) mapJson.get('lifespan');
        myEq.Cost = (Integer) mapJson.get('cost');
        myEq.Warehouse_SKU = (String) mapJson.get('sku');
        myEq.Current_Inventory = (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();
}

}
```

APEX SPECIALIST SUPERBADGE

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){

        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');
```

APEX SPECIALIST SUPERBADGE

```
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;
}
}
```

Challenge-6

WarehouseSyncSchedule.apxc

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

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 ');
    }
}
```

APEX SPECIALIST SUPERBADGE

}

}