

APEX SPECIALIST SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.apxt:-

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

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 {  
    public static Date CheckDates(Date date1, Date date2) {  
        if(DateWithin30Days(date1,date2)) {  
            return date2;  
        }  
        else {  
            return SetEndOfMonthDate(date1);  
        }  
    }  
  
    private static Boolean DateWithin30Days(Date date1, Date date2) {  
        if( date2 < date1) { return false; }  
  
        Date date30Days = date1.addDays(30);  
        if( date2 >= date30Days ) { return false; }  
        else { return true; }  
    }  
  
    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  
public class TestVerifyDate  
{  
    static testMethod void testMethod1()  
    {  
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);  
        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);  
    }  
}
```

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (before insert, before update) {
```

```
        for (Contact c : Trigger.New) {
            if(c.LastName == 'INVALIDNAME') {
                c.AddError("The Last Name '"+c.LastName+"' is not allowed for DML");
            }
        }
    }

    @isTest
    private class TestRestrictContactByName {

        static testMethod void metodoTest() {

            List<Contact> listContact= new List<Contact>();

            Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
            email='Test@test.com');

            Contact c2 = new Contact(FirstName='Francesco1', LastName =
            'INVALIDNAME',email='Test@test.com');

            listContact.add(c1);
            listContact.add(c2);

            Test.startTest();

            try{
                insert listContact;
            }

            catch(Exception ee){}

            Test.stopTest();
        }
    }
}
```

RandomContactFactory.apxc:

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
    FName) {
```

```
List<Contact> contactList = new List<Contact>();
for(Integer i=0;i<numContactsToGenerate;i++) {
    Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact ' + i);
    contactList.add(c);
    System.debug(c);
}
System.debug(contactList.size());
return contactList;
}
}
```

Asynchronous Apex

AccountProcessor.apxc

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

```
@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();
    }
}
```

LeadProcessor.apxc:

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

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
}
```

```
public void execute(Database.BatchableContext bc, List<Lead> leads){  
    for (Lead lead : leads) {  
        lead.LeadSource = 'Dreamforce';  
    }  
    update leads;  
}  
  
public void finish(Database.BatchableContext bc){  
}  
}
```

LeadProcessorTest.apxc

```
@isTest  
  
public class LeadProcessorTest {  
    @testSetup  
    static void setup() {  
        List<Lead> leads = new List<Lead>();  
        for(Integer counter=0 ;counter <200;counter++){  
            Lead lead = new Lead();  
            lead.FirstName ='FirstName';  
            lead.LastName ='LastName'+counter;  
            lead.Company ='demo'+counter;  
            leads.add(lead);  
        }  
        insert leads;  
    }  
  
    @isTest static void test() {  
        Test.startTest();  
  
        LeadProcessor leadProcessor = new LeadProcessor();
```

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

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable  
{  
    private Contact c;  
    private String state;  
    public AddPrimaryContact(Contact c, String state)  
    {  
        this.c = c;  
        this.state = state;  
    }  
    public void execute(QueueableContext context)  
    {  
        List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from  
contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];  
        List<Contact> lstContact = new List<Contact>();  
        for (Account acc:ListAccount)  
        {  
            Contact cont = c.clone(false,false,false,false);  
            cont.AccountId = acc.id;  
            lstContact.add( cont );  
        }  
        if(lstContact.size() >0 )  
        {
```

```
        insert lstContact;  
    }  
}  
}
```

AddPrimaryContactTest.apxc

```
@isTest  
public class AddPrimaryContactTest  
{  
    @isTest static void TestList()  
    {  
        List<Account> Teste = new List <Account>();  
        for(Integer i=0;i<50;i++)  
        {  
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));  
        }  
        for(Integer j=0;j<50;j++)  
        {  
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));  
        }  
        insert Teste;  
        Contact co = new Contact();  
        co.FirstName='demo';  
        co.LastName ='demo';  
        insert co;  
        String state = 'CA';  
        AddPrimaryContact apc = new AddPrimaryContact(co, state);  
        Test.startTest();
```



```
        System.enqueueJob(apc);  
        Test.stopTest();  
    }  
}
```

DailyLeadProcessor.apxc

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

DailyLeadProcessorTest.apxc

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

```
DailyLeadProcessor());  
    }  
}
```

Apex Integration Services

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');  
    }  
}
```

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 );  
}  
}
```

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;  
    }  
}
```

ParkLocator.apxc

```
public class ParkLocator {  
    public static string[] country(string theCountry) {  
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space  
        return parkSvc.byCountry(theCountry);  
    }  
}
```

ParkLocatorTest.apxc

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

ParkServiceMock.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) {
```

```
// start - specify the response you want to send

ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();

response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};

// end

response.put('response_x', response_x);

}

}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')

global with sharing class AccountManager {

    @HttpGet

    global static Account getAccount(){

        RestRequest request=RestContext.request;

        string accountId=request.requestURI.substringBetween('Accounts/', '/contacts');

        Account result=[SELECT Id,Name,(Select Id,Name from Contacts) from Account where
        Id=:accountId Limit 1];

        return result;

    }

}
```

AccountManagerTest.apxc

```
@IsTest

private class AccountManagerTest {

    @isTest static void testGetContactsByAccountId(){

        Id recordId=createTestRecord();

        RestRequest request=new RestRequest();

        request.requestUri='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+
        recordId+'/contacts';

    }

}
```

```
        request.httpMethod='GET';
        RestContext.request=request;

        Account thisAccount=AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record',thisAccount.Name);
    }

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

APEX SPECIALIST 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, 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);  
  
    }  
  
}
```

Challenge-2:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {  
  
    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 = (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();  
}  
}
```

Challenge-3:

WarehouseSyncSchedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {  
    global void execute(SchedulableContext ctx) {  
        WarehouseCalloutService.runWarehouseEquipmentSync();  
    }  
}
```

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 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);
    }
}
```

Challenge-5:

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 {  
  
    global static HttpResponse respond(HttpRequest request){  
  
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',  
request.getEndpoint());  
  
        System.assertEquals('GET', request.getMethod());  
  
        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;  
    }  
}
```

```
}  
  
}
```

Challenge-6:

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();  
  
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];  
  
        System.assertEquals(jobID, a.Id,'Schedule ');  
  
    }  
  
}
```

```
}  
  
}  
  
|
```