

## **1. APEX TRIGGERS :**

### **1.1 Bulk Apex Trigger :**

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

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won') {
            tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

## **2. APEX TESTING :**

### **2.1 Get Started with Apex Unit Tests :**

TestVerifyDate :

```
@isTest
private class TestVerifyDate {
    @isTest static void test1() {
        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));
        System.assertEquals(Date.parse('01/03/2020'), d);
    }
    @isTest static void test2() {
        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));
        System.assertEquals(Date.parse('01/31/2020'), d);
    }
}
```

### **2.2 Test Apex Triggers :**

### TestRestrictContactByName :

```
@isTest
public 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();
    }
}
```

### **2.3 Create Test Data For Apex Tests :**

#### RandomContactFactory :

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer num, String lastname) {
        List<Contact> contactList = new List<Contact>();
        for(Integer i = 1; i <= num ; i++) {
```

```

        Contact ct = new Contact(FirstName = 'Test '+i, LastName = lastname);
        contactList.add(ct);
    }
    return contactList;
}
}

```

### 3. ASYNCHRONOUS APEX :

#### 3.1 Use Future Methods :

AccountProcessor :

```

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

```

AccountProcessorTest :

```

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

        Contact con = new Contact();
        con.FirstName = 'Binary';
        con.LastName = 'Programming';
    }
}

```

```

con.AccountId = a.Id;

insert con;

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

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

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

```

### 3.2 Use Batch Apex :

LeadProcessor :

```

public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        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 :

@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();
    }
}

```

### **3.3 Control Processes with Queueable Apex:**

AddPrimaryContact :

```

public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)

```

```

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

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

}
}

```

#### AddPrimaryContactTest :

```

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

```

```

insert Teste;

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

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

```

### 3.4 Schedule Jobs Using The Apex Scheduler :

DailyLeadProcessor :

```

public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext ctx) {
        //System.debug('Context ' + ctx.getTriggerId()); // Returns the ID of the CronTrigger
        scheduled job

        // Get 200 Lead records and modify the LeadSource field
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null
LIMIT 200];
        for ( Lead l : leads) {
            l.LeadSource = 'Dreamforce';
        }

        // Update the modified records
        update leads;
    }
}

```

DailyLeadProcessorTest :

@isTest

private class DailyLeadProcessorTest {

private static String CRON\_EXP = '0 0 0 ? \* \* \*'; // Midnight every day

@isTest

private static void testSchedulableClass() {

    // Load test data

    List<Lead> leads = new List<Lead>();

    for (Integer i=0; i<500; i++) {

        if ( i < 250 ) {

            leads.add(new Lead(LastName='Connock', Company='Salesforce'));

        } else {

            leads.add(new Lead(LastName='Connock', Company='Salesforce',  
LeadSource='Other'));

        }

    }

    insert leads;

    // Perform the test

    Test.startTest();

    String jobId = System.schedule('Process Leads', CRON\_EXP, new  
DailyLeadProcessor());

    Test.stopTest();

    // Check the result

    List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE  
LeadSource = 'Dreamforce'];

    System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 record not  
updated correctly');

    // Check the scheduled time

    List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM  
CronTrigger WHERE Id = :jobId];



```

System.debug('Next Fire Time ' + cts[0].NextFireTime);

// Not sure this works for all timezones
//Datetime midnight = Datetime.newInstance(Date.today(),
Time.newInstance(0,0,0,0));
//System.assertEquals(midnight.addHours(24), cts[0].NextFireTime, 'ERROR: Not
scheduled for Midnight local time');
}
}

```

## 5. APEX INTEGRATION SERVICES :

### 5.1 Apex REST Callouts :

Animal Locator:

```

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

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

    public static String getAnimalNameById (Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
        //request.setHeader('id', String.valueOf(id)); -- cannot be used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
    }
}

```

```

        system.debug('response: ' + response.getBody());
        //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
        system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }
}

```

### AnimalLocatorTest:

```

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

```

### AnimalLocatorMock:

```

@Test
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}

```

```
}  
}
```

## 5.2 Apex SOAP Callouts :

### Apex Class

ParkService

//Generated by wsdl2apex

```
public class ParkService {  
    public class byCountryResponse {  
        public String[] return_x;  
        private String[] return_x_type_info = new  
String[]{'return','http://parks.services/',null,'0','-1','false'};  
        private String[] apex_schema_type_info = new  
String[]{'http://parks.services/','false','false'};  
        private String[] field_order_type_info = new String[]{'return_x'};  
    }  
    public class byCountry {  
        public String arg0;  
        private String[] arg0_type_info = new  
String[]{'arg0','http://parks.services/',null,'0','1','false'};  
        private String[] apex_schema_type_info = new  
String[]{'http://parks.services/','false','false'};  
        private String[] field_order_type_info = new String[]{'arg0'};  
    }  
    public class ParksImplPort {  
        public String endpoint_x = 'https://th-apex-soap-  
service.herokuapp.com/service/parks';  
        public Map<String,String> inputHttpHeaders_x;  
        public Map<String,String> outputHttpHeaders_x;  
        public String clientCertName_x;  
        public String clientCert_x;  
        public String clientCertPasswd_x;  
        public Integer timeout_x;
```

```

    private String[] ns_map_type_info = new String[]{"http://parks.services/",
'ParkService'};

    public String[] byCountry(String arg0) {
        ParkService.byCountry request_x = new ParkService.byCountry();
        request_x.arg0 = arg0;
        ParkService.byCountryResponse response_x;
        Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
            this,
            request_x,
            response_map_x,
            new String[]{"endpoint_x",
            "http://parks.services/",
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkService.byCountryResponse'}
        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
    }
}

```

#### Park Locator:

```

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

```

#### ParkLocatorTest:

```

@isTest
public class ParkLocatorTest {
    @isTest static void testMock(){
        test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] parksName = ParkLocator.Country('India');
        List<String> country = new List<String>();
        country.add('Inamdar National Park');
        country.add('Riza National Park');
        country.add('Shilpa National Park');
        System.assertEquals(country, parksName, 'park names are not as expected');
    }
}

```

ParkServiceMock:

```

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> country = new List<String>();
        country.add('Inamdar Shola National Park');
        country.add('Riza National Park');
        country.add('Shilpa National Park');
        response_x.return_x = country;
        response.put('response_x', response_x);
    }

}

```

### **5.3 Apex Web Services :**

### AccountManager:

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

    @HttpGet
    global static account getAccount() {

        RestRequest request = RestContext.request;

        String accountId =
request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
        request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account
where id = :accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= '+ a[0]);
        return a[0];

    }

}
```

### AccountManagerTest:

```
@istest
public class AccountManagerTest {

    @isTest static void testGetAccount() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://resourceful-badger-76636-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts'
```

```

        + recordId;
request.httpMethod = 'GET';
RestContext.request = request;
// Call the method to test
Account thisAcc = AccountManager.getAccount();
// Verify results
System.assert(thisAcc != null);
System.assertEquals('Test record', thisAcc.Name);
}
// Helper method
static Id createTestRecord() {
    // Create test record
    Account accTest = new Account(
        Name='Test record');
    insert accTest;
    return accTest.Id;
}
}

```

## **Apex Specialist**

### **CHALLENGE 1**

#### **MAINTENANCE REQUEST HELPER**

```

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;
}
}
}

```

### **MAINTENANCE REQUEST**

```

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

```

### **CHALLENGE 2**

#### **WAREHOUSECALLOUTSERVICES**

```

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);
        }

    }

}

```

### **CHALLENGE 3**

#### **WAREHOUSESYNCSCHEDULE**

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

### **CHALLENGE 4**

#### **MAINTENANCEREQUESTHELPERTEST**

```

@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
}

```

```
private static final string REQUEST_TYPE = 'Routine Maintenance';  
private static final string REQUEST_SUBJECT = 'Testing subject';
```

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

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

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

```
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
```

```
requestId){  
    Equipment_Maintenance_Item__c wp = new  
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,  
                                Maintenance_Request__c = requestId);  
    return wp;  
}
```

@istest

```
private static void testMaintenanceRequestPositive(){
```

```
    Vehicle__c vehicle = createVehicle();
```

```
    insert vehicle;
```

```
    id vehicleId = vehicle.Id;
```

```
    Product2 equipment = createEq();
```

```
    insert equipment;
```

```
    id equipmentId = equipment.Id;
```

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

```
    insert somethingToUpdate;
```

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

```
    insert workP;
```

```
    test.startTest();
```

```
    somethingToUpdate.status = CLOSED;
```

```
update somethingToUpdate;
```

```
test.stopTest();
```

```
Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c,  
Vehicle__c, Date_Due__c
```

```
from case
```

```
where status =:STATUS_NEW];
```

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

```
from Equipment_Maintenance_Item__c
```

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

```
system.assert(workPart != null);
```

```
system.assert(newReq.Subject != null);
```

```
system.assertEquals(newReq.Type, REQUEST_TYPE);
```

```
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
```

```
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
```

```
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
```

```
}
```

```
@istest
```

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

```
Vehicle__C vehicle = createVehicle();
```

```
insert vehicle;
```

```
id vehicleId = vehicle.Id;
```

```
product2 equipment = createEq();
```

```

insert equipment;
id equipmentId = equipment.Id;

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

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

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

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

Equipment_Maintenance_Item__c workPart = [select id
                                           from Equipment_Maintenance_Item__c
                                           where Maintenance_Request__c = :emptyReq.Id];

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

@istest

```



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

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

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

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

    test.startTest();
}
```

```

for(case req : requestList){
    req.Status = CLOSED;
    oldRequestIds.add(req.Id);
}

update requestList;
test.stopTest();

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

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

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

```

### **MAINTENCEREQUESTHELPER**

```

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**

```

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

```

### **CHALLENGE 5**

#### **WAREHOUSECALLOUTSERVICE**

```

public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

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

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

        request.setEndpoint(WAREHOUSE_URL);
    }
}

```

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

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

```
if (response.getStatusCode() == 200){  
    List<Object> jsonResponse =  
    (List<Object>)JSON.deserializeUntyped(response.getBody());  
    System.debug(response.getBody());  
  
    for (Object eq : jsonResponse){  
        Map<String,Object> mapJson = (Map<String,Object>)eq;  
        Product2 myEq = new Product2();  
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');  
        myEq.Name = (String) mapJson.get('name');  
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');  
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');  
        myEq.Cost__c = (Decimal) mapJson.get('lifespan');  
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');  
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');  
        warehouseEq.add(myEq);  
    }  
}
```

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

```

        System.debug(warehouseEq);
    }

}

}

}

```

### **WAREHOUSECALLOUTSERVICETEST**

@isTest

```

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

```

### **WAREHOUSECALLOUTSERVICEMOCK**

@isTest

```

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

```

```

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

        System.assertEquals('GET', request.getMethod());

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

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

```

## **CHALLENGE 6**

### **WAREHOUSESYNCSCHEDULE**

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

### **WAREHOUSESYNCSCHEDULETEST**

```

@isTest

```



```
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId=System.schedule('Warehouse Time To Schedule to Test',
scheduleTime, new WarehouseSyncSchedule());
        Test.stopTest();

        //Contains schedule information for a scheduled job. CronTrigger is similar to a
cron job on UNIX systems.

        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');

    }

}
```