

Apex Triggers

1. Get Started with Apex Program

Trigger name - **AccountAddressTrigger**

-----Code -----

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

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

2. Bulk Apex Trigger

Trigger name - **ClosedOpportunityTrigger**

-----Code-----

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

    List<Task> taskList = new List<Task>();

    for(Opportunity opp : Trigger.new) {

        //Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on
        Create
        if(Trigger.isInsert) { if(Opp.StageName == 'Closed Won') { taskList.add(new
            Task(Subject = 'Follow Up Test Task', WhatId =
            opp.Id));
        }
        }

        //Only create Follow Up Task only once when Opp StageName changed to 'Closed
        Won' on Update
        if(Trigger.isUpdate) { if(Opp.StageName ==
            'Closed Won'
            && Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) { taskList.add(new
            Task(Subject = 'Follow Up Test Task', WhatId =
            opp.Id));
        }
        }
    }

    if(taskList.size(>0) {
        insert taskList;
    }
}
```

```
}  
}
```

APEX TESTING

1. Get Started with Apex Unit Test

Class name- **VerifyDate**

-----code-----

```
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;  
    }  
  
}
```

Test class name - **TestVerifyDate**

-----code-----

@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('01/031/2020'),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('01/015/2020'));
        system.assertEquals(true , flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}

```

2. Test Apex Triggers

Trigger name- **RestrictContactByName**

-----code-----

```

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');
    }
}

```

```
}
```

```
}
```

Test Class name - **TestRestrictContactByName**

-----code-----

@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 "INVALID" is not allowed for DML',  
result.getErrors()[0].getMessage());  
    }  
}
```

3.Create Test Data for Apex Tests

Trigger name- **RandomContactFactory**

-----code-----

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

-----ASYNCHRONOUS APEX-----

2 . Use Future method

Apex class name - **AccountProcessor**

-----code-----

```
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_Contacts__c = contactList.size();
```

```
        accountsToUpdate.add(acc);
```

```
    }
```

```
    update accountsToUpdate;
```

```
}
```

```
}
```

Apex test class name - **AccountProcessorTest**

-----code-----

```
@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();
```

```
    }
```

```
}
```

3. Use Batch Apex

Apex class name - **LeadProcessor**

-----code-----

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

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

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

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

Apex test class name - **LeadProcessorTest**

-----code-----

--

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

```

4. Control Processes with Queueable Apex

Apex class name - **AddPrimaryContact**

-----code-----

```

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;
        }
    }
}

```

Apex test class name - **AddPrimaryContactTest**

-----code-----

@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')]);

    }
}

```

5. Schedule Job Using the Apex Scheduler

Apex class name - **DailyLeadProcessor**

-----code-----

```

global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext sc){
        List<Lead> lstofLead = [SELECT Id FROM Lead WHERE Leadsource = null LIMIT 200];    List<Lead>
        lstofupdatedLead = new List<lead>();
        if(!lstoflead.isEmpty()){
            for(Lead Id : lstofLead){
                Id.Leadsource = 'Dreamforce';
                lstofupdatedLead.add(Id);
            }
            UPDATE lstofupdatedLead;
        }
    }
}

```


Apex test class name - **DailyLeadProcessorTest**

-----code-----

@isTest

private class DailyLeadProcessorTest {

 @testsetup static void

 setup(){

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

 for (Integer i = 1; i <= 200; i++){

 Lead Id=new Lead(Company='Comp'+i, LastName='LN'+i, Status='Working-Contacted');

 lstofLead.add(Id);

 }

 Insert lstofLead;

 }

 static testmethod void testDailyLeadProcessorScheduledJob(){

 String sch = '0 5 12 * * ?';

 Test.startTest();

 String jobId = System.Schedule('ScheduledApexText', sch, new DailyLeadProcessor());

 List<Lead> lstofLead = [SELECT Id FROM Lead WHERE Leadsources = null LIMIT 200];

 system.assertEquals(200, lstofLead.size());

 Test.stopTest();

 }

}

-----APEX INTEGRATION SERVICES-----

2. Apex REST Callout

Apex class name- **AnimalLocator**

-----code-----

public class AnimalLocator { 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.setMethod('GET'); HttpResponse response = http.send(request); String
 strResp = '';

 system.debug('****response '+response.getStatusCode());

 system.debug('****response '+response.getBody()); if

 (response.getStatusCode() == 200){

 Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped

```

(response.getBody());
                Map<string, object> animals = (map<string, object>) results.get('animal');
strResp = string.valueOf (animals.get('name'));
                System.debug('Received the following animals:' + animals
);
                System.debug('strResp >>>>>' + strResp );
        }
        return strResp ;
    } }

```

Apex mock class name- **AnimalLocatorMock**

-----code-----

```

@isTest 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;

    }
}

```

Apex test class name- **AnimalLocatorTest**

-----code-----

```

@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);
    }
}

```

3. Apex SOAP Callouts

Apex class name- **ParkService**

-----code-----

//Generated by wsdl2apex

```

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','1','false'};
    }
}

```

```

        private String[] apex_schema_type_info = new String[]{'http://parks.services/', 'false', 'false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }

    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{'arg0', 'http://parks.services/', null, '0', '1', 'false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/', 'false', 'false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }

    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};      public
String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
                this,
                request_x,
                response_map_x,
                new String[]{endpoint_x,
                    "",
                    'http://parks.services/',
                    'byCountry',
                    'http://parks.services/',
                    'byCountryResponse',
                    'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}

```

Apex class name - **ParkLocator**

-----code-----

```

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

```

Apex Mock class name - **ParkServiceMock**

-----code-----

@isTest

```

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

    }
}

```

Apex Test class name- **ParkLocatorTest**

-----code-----

@isTest

```

public class ParkLocatorTest {
    @isTest static void testCallout (){
        Test.setMock(WebServiceMock.class , new ParkServiceMock());
        String x = 'Yellowstone';
        List<String> result = ParkLocator.country (x);
        string resultstring = string.join(result, ',');
        System.assertEquals ('USA' , resultstring);
    }
}

```

4. Apex Web Services

Apex class name - **AccountManager**

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

Apex test class - **AccountManagerTest**

```
-----code-----  
  
@IsTest  
private class AccountManagerTest{  
    @isTest static void testAccountManager(){  
        Id recordId = getTestAccountId();  
        // Set up a test request  
        RestRequest request = new RestRequest();  
        request.requestUri =  
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId + '/contacts';  
        request.httpMethod = 'GET';  
        RestContext.request = request;  
  
        // Call the method to test  
        Account acc = AccountManager.getAccount();  
  
        // Verify results  
        System.assert(acc != null);  
    }  
    private static Id getTestAccountId(){  
        Account acc = new Account(Name = 'TestAcc2');  
        Insert acc;  
  
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);  
        Insert con;
```

```
return acc.Id;
```

```
}
```

```
}
```

-----APEX SPECIALIST-----

-

Challenge - Automate record creation

Apex class name - MaintenanceRequestHelper

-----code-----

```
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;
    }
}

```

```
}
```

Apex test class name - **MaintenanceRequestHelperTest**

-----code-----

@istest

public with sharing class c {

```
    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);
}
}

```

Apex trigger class - **MaintenanceRequest**

```

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

```

Challenge - Synchronize salesforce data with an external system

Apex class name - **WarehouseCalloutService**

-----code-----

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.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 - Schedule synchronization

Apex class name - **WarehouseSyncSchedule**

```

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

```

Challenge - Test automation logic

Apex test class name - **MaintenanceRequestHelperTest**

```

-----code-----
@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);
}
}

```


Apex class name - **MaintenanceRequestHelper**

-----code-----

```
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;
}
}
}
}
}

```

Apex trigger name - **MaintenanceRequest**

-----code-----

```

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

```

Challenge - Test callout logic

Apex class name- **WarehouseCalloutService**

-----code-----

```

public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.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);
        }
    }
}
}

```

Apex mock class name - **WarehouseCalloutServiceMock**

-----code-----

@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;
}
}

```

Apex test class - **WarehouseCalloutServiceTest**

```

-----code-----
@Test

private class WarehouseCalloutServiceTest {
    @Test
    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]);
    }
}

```

Challenge - Test scheduling logic

Apex class name - **WarehouseSyncSchedule**

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

Apex test class name - **WarehouseSyncScheduleTest**

-----code-----

@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 ');

 }

}