Apex Triggers:

**1.Get Started with Apex Triggers:**

**AccountAddressTrigger:**

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 Triggers:**

**ClosedOpportunityTrigger**:

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;

    }

}

***Apex Testing:***

***1.***[**Get Started with Apex Unit Tests**](https://trailhead.salesforce.com/content/learn/modules/apex_testing/apex_testing_intro?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**VerifyDate:**

public class VerifyDate {

public static Date CheckDates(Date date1, Date date2) {

if(DateWithin30Days(date1,date2)) {

return date2;

} else {

return SetEndOfMonthDate(date1);

}

}

@TestVisible 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; }

}

@TestVisible private static Date SetEndOfMonthDate(Date date1) {

Integer totalDays = Date.daysInMonth(date1.year(), date1.month());

Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);

return lastDay;

}

}

**TestVerifyDate**:

@isTest

private class TestVerifyDate {

    @isTest static void Test\_CheckDates\_Case1(){

        Date D=VerifyDate.CheckDates(date.parse('01/01/2021'),date.parse('01/05/2021'));

        System.assertEquals(date.parse('01/05/2021'),D);

    }

    @isTest static void Test\_CheckDates\_Case2(){

        Date D=VerifyDate.CheckDates(date.parse('01/01/2021'),date.parse('05/05/2021'));

        System.assertEquals(date.parse('01/30/2021'),D);

    }

    @isTest static void Test\_DateWithin30Days\_Case1(){

        Boolean flag=VerifyDate.DateWithin30Days(date.parse('01/01/2021'),date.parse('12/30/2020'));

        System.assertEquals(false,flag);

    }

    @isTest static void Test\_DateWithin30Days\_Case2(){

        Boolean flag=VerifyDate.DateWithin30Days(date.parse('01/01/2021'),date.parse('02/02/2021'));

        System.assertEquals(false,flag);

    }

    @isTest static void Test\_DateWithin30Days\_Case3(){

        Boolean flag=VerifyDate.DateWithin30Days(date.parse('01/01/2021'),date.parse('01/15/2021'));

        System.assertEquals(true,flag);

    }

    @isTest static void Test\_SetEndOfMonthDate(){

        Date returndate=VerifyDate.SetEndOfMonthDate(date.parse('01/01/2021'));

    }

}

**2.**  [**Test Apex Triggers**](https://trailhead.salesforce.com/content/learn/modules/apex_testing/apex_testing_triggers?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**RestrictContactByName:**

trigger RestrictContactByName on Contact (before insert) {

For (Contact c : Trigger.New) {

if(c.LastName == 'INVALIDNAME') {

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

**TestRestrictContactByName:**

@isTest

public class TestRestrictContactByName {

    @isTest static void Test\_insertupdateContact(){

        Contact cnt=new Contact();

        cnt.LastName='INVALIDNAME';

        Test.startTest();

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

        Test.stopTest();

        System.assert(!result.isSuccess());

        System.assert(result.getErrors().size()>0);

        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',result.getErrors()[0].getMessage());

    }

}

[**3.Create Test Data for Apex Tests**](https://trailhead.salesforce.com/content/learn/modules/apex_testing/apex_testing_data?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**RandomContactFactory:**

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**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)  **:**

**1.**[**Use Future Methods**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_future_methods?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AccountProcessor:**

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;

    }

}

**AccountProcessorTest:**

@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();

    }

}

**2.**[**Use Batch Apex**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_batch?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)

**LeadProcessor:**

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);

    }

}

**LeadProcessorTest:**

@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();

    }

}

**3.**[**Control Processes with Queueable Apex**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_queueable?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AddPrimaryContact:**

public class AddPrimaryContact implements Queueable{

    private Contact con;

    private String state;

    public AddPrimaryContact(Contact con,String state){

        this.con=con;

        this.state=state;

    }

    public void execute(QueueableContext context){

        List<Account> accounts=[Select Id,Name,(Select FirstName,LastName,Id from contacts)from Account where BillingState= :state Limit 200];

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

        for(Account acc:accounts){

            Contact c=con.clone();

            c.AccountId=acc.Id;

            primaryContacts.add(c);

        }

        if(primaryContacts.size()>0){

            insert primaryContacts;

        }

    }

}

**AddPrimaryContactTest:**

@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')]);

    }

}

**4.**[**Schedule Jobs Using the Apex Scheduler**](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_scheduled?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**DailyLeadProcessor:**

global class DailyLeadProcessor implements Schedulable{

    global void execute(SchedulableContext ctx){

        List<lead> leadstoupdate=new List<lead>();

        List<Lead> leads=[Select id From Lead Where LeadSource=NULL Limit 200];

        for(Lead l:leads){

            l.LeadSource='Dreamforce';

            leadstoupdate.add(l);

        }

     update leadstoupdate;

    }

}

**DailyLeadProcessorTest:**

@isTest

private class DailyLeadProcessorTest {

    public static String CRON\_EXP='0 0 0 15 3 ? 2023';

    static testmethod void testScheduleJob(){

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

        for(Integer i=0;i<200;i++){

            Lead l=new Lead(

                FirstName='First ' +i,

                LastName='LastName',

                Company='The Inc'

            );

            leads.add(l);

        }

        insert leads;

        Test.startTest();

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

        Test.stopTest();

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

        checkleads=[Select Id From Lead Where LeadSource='Dreamforce' and Company='The Inc'];

        System.assertEquals(200,checkleads.size(),'Leads were not created');

    }

}

**Apex Integration Services**  **:**

[**Apex REST Callouts**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_rest_callouts?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AnimalLocator:**

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:**

@isTest

public 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:**

@isTest

global class AnimalLocatorMock implements HttpCalloutMock{

    global HTTPResponse respond(HTTPRequest request){

       HttpResponse response=new HttpResponse();

        response.setHeader('Content-Type','application/json');

        response.setBody('{"animals":["majestic badger","fluffy bunny","scary bear","chicken","mighty moose"]}');

        response.setStatusCode(200);

        return response;

    }

}

**2.**[**Apex SOAP Callouts**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_soap_callouts?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**ParkLocator** :

public class ParkLocator {

    public static string[] country(string theCountry) {

        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort();

        return parkSvc.byCountry(theCountry);

    }

}

**ParkLocatorTest:**

@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:**

@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);

   }

}

**3.**[**Apex Web Services**](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services/apex_integration_webservices?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)**:**

**AccountManager:**

@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:**

@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 Superbadge:***

**1. Automated Record Creation:**

**MaintenanceRequest:**

trigger MaintenanceRequest on Case (before update, after update) {

    if(Trigger.isUpdate && Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }

}

**MaintenanceRequestHelper:**

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()){

            Map<Id,Case> closedCases = 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'));

            }

            List<Case> newCases = new List<Case>();

            for(Case cc : closedCases.values()){

                Case nc = new Case (

                    ParentId = cc.Id,

                    Status = 'New',

                    Subject = 'Routine Maintenance',

                    Type = 'Routine Maintenance',

                    Vehicle\_\_c = cc.Vehicle\_\_c,

                    Equipment\_\_c =cc.Equipment\_\_c,

                    Origin = 'Web',

                    Date\_Reported\_\_c = Date.Today()

            };

                //If (maintenanceCycles.containskey(cc.Id)){

                    nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));

                //} else {

                //    nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);

                //}

                newCases.add(nc);

            }

            insert newCases;

            List<Equipment\_Maintenance\_Item\_\_c> clonedList = new List<Equipment\_Maintenance\_Item\_\_c>();

            for (Case nc : newCases){

                for (Equipment\_Maintenance\_Item\_\_c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){

                    Equipment\_Maintenance\_Item\_\_c item = clonedListItem.clone();

                    item.Maintenance\_Request\_\_c = nc.Id;

                    clonedList.add(item);

                }

            }

            insert clonedList;

        }

    }

}

**2. Synchronize Salesforce data with an external system:**

**WarehouseCalloutService.apxc :**

public with sharing class WarehouseCalloutService implements Queueable{

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

    @future(callout=true)

    public static void runWarehouseEquipmentSync(){

        System.debug('go into runWarehouseEquipmentSync');

        Http http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE\_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();

        System.debug(response.getStatusCode());

        if (response.getStatusCode() == 200){

            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());

            System.debug(response.getBody());

                for (Object jR : jsonResponse){

                Map<String,Object> mapJson = (Map<String,Object>)jR;

                Product2 product2 = new Product2();

                product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');

                product2.Cost\_\_c = (Integer) mapJson.get('cost');

                product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');

                product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');

                product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');

                product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');

                product2.ProductCode = (String) mapJson.get('\_id');

                product2List.add(product2);

            }

            if (product2List.size() > 0){

                upsert product2List;

                System.debug('Your equipment was synced with the warehouse one');

            }

        }

    }

    public static void execute (QueueableContext context){

        System.debug('start runWarehouseEquipmentSync');

        runWarehouseEquipmentSync();

        System.debug('end runWarehouseEquipmentSync');

    }

}

**3.Schedule synchronization using Apex code**

**WarehouseSyncSchedule:**

global with sharing class WarehouseSyncSchedule implements Schedulable{

    global void execute(SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

**4. Test automation logic:**

**MaintenanceRequestHelperTest :**

@isTest

public with sharing class MaintenanceRequestHelperTest {

    // createVehicle

    private static Vehicle\_\_c createVehicle(){

        Vehicle\_\_c vehicle = new Vehicle\_\_C(name = 'Testing Vehicle');

        return vehicle;

    }

    // createEquipment

    private static Product2 createEquipment(){

        product2 equipment = new product2(name = 'Testing equipment',

                                          lifespan\_months\_\_c = 10,

                                          maintenance\_cycle\_\_c = 10,

                                          replacement\_part\_\_c = true);

        return equipment;

    }

    // createMaintenanceRequest

    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){

        case cse = new case(Type='Repair',

                            Status='New',

                            Origin='Web',

                            Subject='Testing subject',

                            Equipment\_\_c=equipmentId,

                            Vehicle\_\_c=vehicleId);

        return cse;

    }

    // createEquipmentMaintenanceItem

    private static Equipment\_Maintenance\_Item\_\_c createEquipmentMaintenanceItem(id equipmentId,id requestId){

        Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = new Equipment\_Maintenance\_Item\_\_c(

            Equipment\_\_c = equipmentId,

            Maintenance\_Request\_\_c = requestId);

        return equipmentMaintenanceItem;

    }

    @isTest

    private static void testPositive(){

        Vehicle\_\_c vehicle = createVehicle();

        insert vehicle;

        id vehicleId = vehicle.Id;

        Product2 equipment = createEquipment();

        insert equipment;

        id equipmentId = equipment.Id;

        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);

        insert createdCase;

        Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = createEquipmentMaintenanceItem(equipmentId,createdCase.id);

        insert equipmentMaintenanceItem;

        test.startTest();

        createdCase.status = 'Closed';

        update createdCase;

        test.stopTest();

        Case newCase = [Select id,

                        subject,

                        type,

                        Equipment\_\_c,

                        Date\_Reported\_\_c,

                        Vehicle\_\_c,

                        Date\_Due\_\_c

                       from case

                       where status ='New'];

        Equipment\_Maintenance\_Item\_\_c workPart = [select id

                                                  from Equipment\_Maintenance\_Item\_\_c

                                                  where Maintenance\_Request\_\_c =:newCase.Id];

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

        system.assert(allCase.size() == 2);

        system.assert(newCase != null);

        system.assert(newCase.Subject != null);

        system.assertEquals(newCase.Type, 'Routine Maintenance');

        SYSTEM.assertEquals(newCase.Equipment\_\_c, equipmentId);

        SYSTEM.assertEquals(newCase.Vehicle\_\_c, vehicleId);

        SYSTEM.assertEquals(newCase.Date\_Reported\_\_c, system.today());

    }

    @isTest

    private static void testNegative(){

        Vehicle\_\_C vehicle = createVehicle();

        insert vehicle;

        id vehicleId = vehicle.Id;

        product2 equipment = createEquipment();

        insert equipment;

        id equipmentId = equipment.Id;

        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);

        insert createdCase;

        Equipment\_Maintenance\_Item\_\_c workP = createEquipmentMaintenanceItem(equipmentId, createdCase.Id);

        insert workP;

        test.startTest();

        createdCase.Status = 'Working';

        update createdCase;

        test.stopTest();

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

        Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = [select id

                                                  from Equipment\_Maintenance\_Item\_\_c

                                                  where Maintenance\_Request\_\_c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);

        system.assert(allCase.size() == 1);

    }

    @isTest

    private static void testBulk(){

        list<Vehicle\_\_C> vehicleList = new list<Vehicle\_\_C>();

        list<Product2> equipmentList = new list<Product2>();

        list<Equipment\_Maintenance\_Item\_\_c> equipmentMaintenanceItemList = new list<Equipment\_Maintenance\_Item\_\_c>();

        list<case> caseList = new list<case>();

        list<id> oldCaseIds = new list<id>();

        for(integer i = 0; i < 300; i++){

            vehicleList.add(createVehicle());

            equipmentList.add(createEquipment());

        }

        insert vehicleList;

        insert equipmentList;

        for(integer i = 0; i < 300; i++){

            caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));

        }

        insert caseList;

        for(integer i = 0; i < 300; i++){

            equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id, caseList.get(i).id));

        }

        insert equipmentMaintenanceItemList;

        test.startTest();

        for(case cs : caseList){

            cs.Status = 'Closed';

            oldCaseIds.add(cs.Id);

        }

        update caseList;

        test.stopTest();

        list<case> newCase = [select id

                                  from case

                                  where status ='New'];

        list<Equipment\_Maintenance\_Item\_\_c> workParts = [select id

                                                         from Equipment\_Maintenance\_Item\_\_c

                                                         where Maintenance\_Request\_\_c in: oldCaseIds];

        system.assert(newCase.size() == 300);

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

        system.assert(allCase.size() == 600);

    }

}

**MaintenanceRequest :**

trigger MaintenanceRequest on Case (before update, after update) {

    if(Trigger.isUpdate && Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }

}

**5. Test callout logic :**

**WarehouseCalloutService:**

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(){

        System.debug('go into runWarehouseEquipmentSync');

        Http http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE\_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();

        System.debug(response.getStatusCode());

        if (response.getStatusCode() == 200){

            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());

            System.debug(response.getBody());

                for (Object jR : jsonResponse){

                Map<String,Object> mapJson = (Map<String,Object>)jR;

                Product2 product2 = new Product2();

                product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');

                product2.Cost\_\_c = (Integer) mapJson.get('cost');

                product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');

                product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');

                product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');

                product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');

                product2.ProductCode = (String) mapJson.get('\_id');

                product2List.add(product2);

            }

            if (product2List.size() > 0){

                upsert product2List;

                System.debug('Your equipment was synced with the warehouse one');

            }

        }

    }

    public static void execute (QueueableContext context){

        System.debug('start runWarehouseEquipmentSync');

        runWarehouseEquipmentSync();

        System.debug('end runWarehouseEquipmentSync');

    }

}

**WarehouseCalloutServiceTest:**

@IsTest

private class WarehouseCalloutServiceTest {

    // implement your mock callout test here

@isTest

    static void testWarehouseCallout() {

        test.startTest();

        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        WarehouseCalloutService.execute(null);

        test.stopTest();

        List<Product2> product2List = new List<Product2>();

        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());

        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);

        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);

        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);

    }

}

**WarehouseCalloutServiceMock:**

@isTest

global class WarehouseCalloutServiceMock implements HttpCalloutMock {

    // implement http mock callout

    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type', 'application/json');

        response.setBody('[{"\_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

        response.setStatusCode(200);

        return response;

    }

}

**6. Test scheduling logic:**

**WarehouseSyncSchedule:**

global with sharing class WarehouseSyncSchedule implements Schedulable{

    global void execute(SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

**WarehouseSyncScheduleTest:**

@isTest

public with sharing class WarehouseSyncScheduleTest {

    // implement scheduled code here

    @isTest

    static void test() {

        String scheduleTime = '00 00 00 \* \* ? \*';

        Test.startTest();

        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new WarehouseSyncSchedule());

        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];

        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();

    }

}