**APEX TRIGGERS**

ClosedOppurtunityTrigger.apxt

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

 list<Task> newTask= new list<Task>();

    for(Opportunity oppWon :[Select Id from Opportunity where StageName='Closed Won'

                            and Id in: Trigger.new]){

        newTask.add(new Task (Subject ='Follow Up Test Task',WhatId=oppWon.Id));

    }

    if(newTask.size()>0){

        upsert newTask;

    }

}

AccountAddressTrigger.apxt

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

 for(Account a : Trigger.New)

    {

        if(a.Match\_Billing\_Address\_\_c==true)

        {

            a.ShippingPostalCode = a.BillingPostalCode;

        }

    }

}

**APEX TESTING**

TestVerifyDate.apxc

@isTest

private class TestVerifyDate {

 @isTest

    static void testCheckDates\_1()

    {

        Date d = VerifyDate.CheckDates(date.parse('01/03/2032'), date.parse('01/07/2032'));

        System.assertEquals(date.parse('01/07/2032'), d);

    }

    @isTest

    static void testCheckDates\_2()

    {

        Date d = VerifyDate.CheckDates(date.parse('01/04/2032'), date.parse('03/09/2032'));

        System.assertEquals(date.parse('01/31/2032'),d);

    }

    @isTest

    static void testDateWithin30Days\_1()

    {

        Boolean f = VerifyDate.DateWithin30Days(date.parse('01/01/2032'), date.parse('12/30/2031'));

        System.assertEquals(false,f);

    }

    @isTest

    static void testDateWithin30Days\_2()

    {

        Boolean f = VerifyDate.DateWithin30Days(date.parse('01/01/2032'), date.parse('02/02/2031'));

        System.assertEquals(false,f);

    }

    @isTest

    static void testDateWithin30Days\_3()

    {

        Boolean f = VerifyDate.DateWithin30Days(date.parse('01/01/2032'), date.parse('01/10/2032'));

        System.assertEquals(false,f);

    }

    @isTest

    static void testSetEndOfMonthDate()

    {

        Date rd = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2032'));

        System.assertEquals(date.parse('01/01/2032'), rd);

    }

}

TestRestrictContactByName.apxc

@isTest

private class TestRestrictContactByName {

    @isTest

 static void methodTest()

    {

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

        Contact cntct1 = new Contact(FirstName='varsh', LastName='vark' , email='Test@test.com');

        Contact cntct2 = new Contact(FirstName='valkey', LastName = 'INVALIDNAME',email='Test@test.com');

        Contactlist.add(cntct1);

        Contactlist.add(cntct2);

        Test.startTest();

            try

            {

                insert Contactlist;

            }

            catch(Exception ee)

            {

            }

        Test.stopTest();

    }

}

RandomContactFactory.apxc

public class RandomContactFactory {

 public static List<Contact> generateRandomContacts(integer n,string last){

       integer num= n;

       list<contact> first=[select  FirstName from contact limit : num];

       return first;

 }

}

**Asynchronous Apex**

AccountProcessor.apxc

public without sharing class AccountProcessor {

  @future

  public static void countContacts(Set<id> setId)

  {

      List<Account> lstAccount = [select id,Number\_of\_Contacts\_\_c , (select id from contacts ) from account where id in :setId ];

      for( Account acc : lstAccount )

      {

          List<Contact> lstCont = acc.contacts ;

          acc.Number\_of\_Contacts\_\_c = lstCont.size();

      }

      update lstAccount;

  }

}

AccountProcessorTest.apxc

@isTest

private class AccountProcessorTest {

 public static testmethod void TestAccountProcessorTest()

    {

        Account a = new Account();

        a.Name = 'Test Account';

        Insert a;

        Contact cont = New Contact();

        cont.FirstName ='Bob';

        cont.LastName ='Masters';

        cont.AccountId = a.Id;

        Insert cont;

        set<Id> setAccId = new Set<ID>();

        setAccId.add(a.id);

        Test.startTest();

            AccountProcessor.countContacts(setAccId);

        Test.stopTest();

        Account ACC = [select Number\_of\_Contacts\_\_c from Account where id = :a.id LIMIT 1];

        System.assertEquals ( Integer.valueOf(ACC.Number\_of\_Contacts\_\_c) ,1);

  }

}

LeadProcessor.apxc

public without sharing class LeadProcessor implements Database.Batchable<sObject>{

    public Database.QueryLocator start(Database.BatchableContext dbc) {

        return Database.getQueryLocator([SELECT Id, Name FROM Lead]);

    }

    public void execute(Database.BatchableContext dbc, List<Lead> leads) {

        for(Lead l :leads) {

            l.LeadSource = 'Dreamforce';

        }

        update leads;

    }

    public void finish (Database.BatchableContext dbc){

        System.debug('Done');

    }

}

LeadProcessorTest.apxc

@isTest

private class LeadProcessorTest {

 @isTest

    private static void testBatchClass() {

        //Load test data

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

        for(Integer i=0;i<200;i++)

        {

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

        }

        insert leads;

        Test.startTest();

        LeadProcessor lp = new LeadProcessor() ;

        Id batchId = Database.executeBatch(lp,200);

        Test.stopTest();

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

        System.assertEquals(200,updateLeads.size(),'Error');

    }

}

AddPrimaryContact.apxc

public without sharing class AddPrimaryContact implements Queueable{

 private contact contact;

    private string state;

    public AddPrimaryContact (Contact inputContact, String inputState) {

        this.contact = inputContact;

        this.state = inputState;

    }

    public void execute(QueueableContext context) {

        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

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

        for(Account acc : accounts) {

            contact contactclone = contact.clone();

            contactClone.AccountId = acc.Id;

            contacts.add(contactClone);

        }

 insert contacts;

    }

}

AddPrimaryContactTest.apxc

@isTest

private class AddPrimaryContactTest {

 @isTest

    private static void testQueuebleClass() {

        List <Account> accounts = new List<Account>();

        for (Integer i=0;i<500;i++) {

            Account acc = new Account(Name='Test Account');

            if(i<250) {

                acc.BillingState = 'NY';

            }else{

                acc.BillingState = 'CA';

            }

            accounts.add(acc);

        }

        insert accounts;

        Contact contact = new Contact(FirstName='Simon', LastName='Connock');

        insert contact;

        Test.startTest();

        Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));

        Test.stopTest();

        List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState = 'CA'];

        System.assertEquals(200,contacts.size(),'ERROR');

    }

}

DailyLeadProcessor.apxc

global class DailyLeadProcessor implements Schedulable{

    global void execute(SchedulableContext ctx){

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

        if(leads.size() > 0){

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

            for(Lead lead : leads){

                lead.LeadSource = 'DreamForce';

                newLeads.add(lead);

            }

            update newLeads;

        }

    }

}

DailyLeadProcessorTest.apxc

@isTest

private class DailyLeadProcessorTest{

    //Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

    public static String CRON\_EXP = '0 0 1 \* \* ?';

    static testmethod void testScheduledJob(){

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

        for(Integer i = 0; i < 200; i++){

            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');

            leads.add(lead);

        }

        insert leads;

        Test.startTest();

        // Schedule the test job

        String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());

        // Stopping the test will run the job synchronously

        Test.stopTest();

    }

}

// Schedule the test job

        String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());

        // Stopping the test will run the job synchronously

        Test.stopTest();

    }

}

**Apex Integration Services**

AnimalLocator.apxc

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);

        Map<Integer,String> mapAnimal = new Map<Integer,String>();

        Integer varId;

        String varName;

        JSONParser parser1= JSON.createParser(response.getBody());

        while (parser1.nextToken() != null) {

            if ((parser1.getCurrentToken() == JSONToken.FIELD\_NAME) && (parser1.getText() == 'id')) {

                parser1.nextToken();

                varId=parser1.getIntegerValue();

                parser1.nextToken();

            }

            if ((parser1.getCurrentToken() == JSONToken.FIELD\_NAME) && (parser1.getText() == 'name')) {

                parser1.nextToken();

                varName=parser1.getText();

            }

            mapAnimal.put(varId,varName);

        }

        return mapAnimal.get(id);

    }

}

AnimalLocatorMock.apxc

@isTest

global class AnimalLocatorMock implements HttpCalloutMock{

  global HTTPResponse respond(HTTPRequest request) {

        // Create a fake response

        HttpResponse response = new HttpResponse();

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

         response.setBody('{"animal": {"id":1,"name":"tiger","eats":"small animals","says":"roar"}}');

        response.setStatusCode(200);

        return response;

     }

}

AnimalLocatorTest.apxc

@isTest

private class AnimalLocatorTest

{

    @isTest

    static void testAnimalNameById\_1()

    {

        // Set mock callout class

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

        String actualValue = AnimalLocator.getAnimalNameById(1);

        String expectedValue = 'tiger';

        System.assertEquals(expectedValue, actualValue);

    }

}

ParkLocator.apxc

public class ParkLocator {

    public static List <String> country(String country){

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

        return p.byCountry(country);

    }

}

ParkServiceMock.apxc

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> {'wrewte','g','df'};

        // end

        response.put('response\_x', response\_x);

 }

}

ParkLocatorTest.apxc

@isTest

private class ParkLocatorTest {

    @isTest static void test(){

        Test.setMock(WebServiceMock.class, new ParkServiceMock());

        String country = 'India';

        List<String> exp = new List<String>{'wrewte','g','df'};

        System.assertEquals(exp,ParkLocator.country(country));

    }

}

AccountManager.apxc

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

global 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;

    }

}

AccountManagerTest.apxc

@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 BADGE** 

MaintenanceRequestHelper.apxc

public with sharing class MaintenanceRequestHelper {

    public static void updateWorkOrders(List<Case> uwo, Map<Id,Case> nonucMap) {

        // TODO: Complete the method to update workorders

        Set<Id> validIds = new Set<Id>();

        For (Case c : uwo){

            if (nonucMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

                    validIds.add(c.Id);

                }

            }

        }

        //When an existing maintenance request of type Repair or Routine Maintenance is closed,

        //create a new maintenance request for a future routine checkup.

        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>();

            //calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

            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 multiple pieces of equipment are used in the maintenance request,

                //define the due date by applying the shortest maintenance cycle to today’s date.

                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;

        }

    }

}

MaintenanceRequest.apxt

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

    // ToDo: Call MaintenanceRequestHelper.updateWorkOrders

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

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

    }

}

MaintenanceRequestHelperTest.apxc

@istest

public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS\_NEW = 'New';

    private static final string WORKING = 'Working';

    private static final string CLOSED = 'Closed';

    private static final string REPAIR = 'Repair';

    private static final string REQUEST\_ORIGIN = 'Web';

    private static final string REQUEST\_TYPE = 'Routine Maintenance';

    private static final string REQUEST\_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle\_\_c createVehicle(){

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

        return Vehicle;

    }

    PRIVATE STATIC Product2 createEq(){

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

                                         lifespan\_months\_\_C = 10,

                                         maintenance\_cycle\_\_C = 10,

                                         replacement\_part\_\_c = true);

        return equipment;

    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){

        case cs = new case(Type=REPAIR,

                          Status=STATUS\_NEW,

                          Origin=REQUEST\_ORIGIN,

                          Subject=REQUEST\_SUBJECT,

                          Equipment\_\_c=equipmentId,

                          Vehicle\_\_c=vehicleId);

        return cs;

    }

    PRIVATE STATIC Equipment\_Maintenance\_Item\_\_c createWorkPart(id equipmentId,id requestId){

        Equipment\_Maintenance\_Item\_\_c wp = new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipmentId, Maintenance\_Request\_\_c = requestId);

        return wp;

    }

    @istest

    private static void testMaintenanceRequestPositive(){

        Vehicle\_\_c vehicle = createVehicle();

        insert vehicle;

        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();

        insert equipment;

        id equipmentId = equipment.Id;

        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);

        insert somethingToUpdate;

        Equipment\_Maintenance\_Item\_\_c workP = createWorkPart(equipmentId,somethingToUpdate.id);

        insert workP;

        test.startTest();

        somethingToUpdate.status = CLOSED;

        update somethingToUpdate;

        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment\_\_c, Date\_Reported\_\_c, Vehicle\_\_c, Date\_Due\_\_c

                      from case

                      where status =:STATUS\_NEW];

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

                                                 from Equipment\_Maintenance\_Item\_\_c

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

        system.assert(workPart != null);

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

        system.assertEquals(newReq.Type, REQUEST\_TYPE);

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

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

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

    }

    @istest

    private static void testMaintenanceRequestNegative(){

        Vehicle\_\_C vehicle = createVehicle();

        insert vehicle;

        id vehicleId = vehicle.Id;

        product2 equipment = createEq();

        insert equipment;

        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);

        insert emptyReq;

        Equipment\_Maintenance\_Item\_\_c workP = createWorkPart(equipmentId, emptyReq.Id);

        insert workP;

        test.startTest();

        emptyReq.Status = WORKING;

        update emptyReq;

        test.stopTest();

        list<case> allRequest = [select id

                                 from case];

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

                                                  from Equipment\_Maintenance\_Item\_\_c

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

        system.assert(workPart != null);

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

    }

    @istest

    private static void testMaintenanceRequestBulk(){

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

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

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

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

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

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

           vehicleList.add(createVehicle());

            equipmentList.add(createEq());

        }

        insert vehicleList;

        insert equipmentList;

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

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

        }

        insert requestList;

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

            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));

        }

        insert workPartList;

        test.startTest();

        for(case req : requestList){

            req.Status = CLOSED;

            oldRequestIds.add(req.Id);

        }

        update requestList;

        test.stopTest();

        list<case> allRequests = [select id

                                 from case

                                 where status =: STATUS\_NEW];

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

                                                        from Equipment\_Maintenance\_Item\_\_c

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

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

    }

}

WarehouseCalloutService.apxc

public with sharing class WarehouseCalloutService implements Queueable {

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

    @future(callout=true)

    public static void runWarehouseEquipmentSync(){

        Http http = new Http();

        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE\_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);

        List<Product2> prdct2List = new List<Product2>();

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

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

            for (Object record: results) {

                Map<String, Object> recordMap = (Map<String, Object>)record;

                Product2 p2 = new Product2();

                p2.Name = (String)recordMap.get('name');

                p2.ProductCode = (String)recordMap.get('\_id');

                p2.Warehouse\_SKU\_\_c = (String)recordMap.get('sku');

                p2.Current\_Inventory\_\_c = (Double)recordMap.get('quantity');

                p2.Maintenance\_Cycle\_\_c = (Double)recordMap.get('maintenanceperiod');

                p2.Replacement\_Part\_\_c = (Boolean)recordMap.get('replacement');

                p2.Lifespan\_Months\_\_c = (Double)recordMap.get('lifespan');

                p2.Cost\_\_c = (Decimal)recordMap.get('cost');

                prdct2List.add(p2);

            }

            if(prdct2List.size() > 0){

                insert prdct2List;

            }

        }

    }

    public static void execute (QueueableContext context){

        runWarehouseEquipmentSync();

    }

}

WarehouseCalloutServiceMock.apxc

@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":2,"sku":"100005"}]'

        response.setStatusCode(200);

        return response;

    }

}

WarehouseCalloutServiceTest.apxc

@IsTest

private class WarehouseCalloutServiceTest {

    // implement your mock callout test here

 @isTest

    static void testWareHouseCallout(){

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

        Test.startTest();

        WarehouseCalloutService.runWarehouseEquipmentSync();

        System.enqueueJob(new WarehouseCalloutService());

        Test.stopTest();

        System.assertEquals(1, [SELECT count() FROM Product2]);

    }

}

WarehouseSyncSchedule.apxc

global with sharing class WarehouseSyncSchedule implements Schedulable{

    // implement scheduled code here

    global void execute(SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

WarehouseSyncScheduleTest.apxc

@isTest

public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){

        String scheduleTime = '00 00 01 \* \* ?';

        Test.startTest();

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

        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSyncSchedule());

        Test.stopTest();

        CronTrigger a=[SELECT Id FROM CronTrigger WHERE NextFireTime > today];

        System.assertEquals(jobID, a.Id,'Schedule ');

    }

}