APEX MODULES CODES

MODULE :APEX TRIGGERS

1.Get Started with apex triggers:(AccountAddressTrigger)

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

}
```

2.Bulk Apex Trigger:(ClosedOpportunityTrigger)

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
    for (Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName =
'Closed Won' AND Id IN :Trigger.new])
   {
    taskList.add(new Task(Subject = 'Follow Up Test Task',
    WhatId = opp.Id));
    }
   if(taskList.size()>0){
        insert taskList;
        }
    }
```

MODULE:APEX TESTING

1.Get Started with Apex Unit tests:(VerifyDate)

```
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
        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
        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.apxc:(VerifyDate)

```
@isTest
private class TestVerifyDate {
  static testMethod void TestVerifyDate() {
    VerifyDate.CheckDates(System.today(),System.today().addDays(10));
     VerifyDate.CheckDates(System.today(),System.today().addDays(78));
  }
}
```

2.Test apex Triggers:(Create an Apex trigger RestrictContactByName on the Contact object)

```
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');
                    }
}
}
```

Create separate test class TestRestrictContactByName

```
@istest
private class TestRestrictContactByName {
   @istest static void testname(){
      contact c = new contact(firstname='Satya',lastname='INVALIDNAME');
      test.startTest();
      database.SaveResult result = database.insert(c,false);
      test.stopTest();
      system.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

3.Create Test Data For Apex Triggers:(RandomContactFactory)

```
//@isTest
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer numContactsToGenerate,
String FName) {
      List<Contact> contactList = new List<Contact>();

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

}
```

MODULE: ASYNCHRONOUS APEX

1.Use Future Methods (Account Processor.apxc)

```
public class AccountProcessor {

   @future
   public static void countContacts(List<Id> accountId_lst) {

      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from contacts) from
account]);
      for(account a:account_lst_all) {
         account_cno.put(a.id,a.contacts.size()); //populate the map

      }

      List<account> account_lst = new List<account>(); // list of account that we will upsert

      for(Id accountId : accountId_lst) {
         if(account_cno.containsKey(accountId)) {
            account acc = new account();
            acc.Id = accountId;
            acc.Number_of_Contacts__c = account_cno.get(accountId);
            account_lst.add(acc);
         }

      }
      upsert account_lst;
   }

}
```

Test.apxc (Account Processor)

```
@isTest
public class AccountProcessorTest {

   @isTest
   public static void testFunc() {
      account acc = new account();
```

```
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;


        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
            AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from account
where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }

}
```

2.Use Batch Apex: (Lead Processor.apxc)

```
global class LeadProcessor implements Database.Batchable<sObject>, Database.Stateful {
    global Integer leadsProcessed = 0;
    global Database.QueryLocator start(Database.BatchableContext bc){


        return Database.getQueryLocator('select id, lastname ,status, company from Lead');

    }
    global void execute(Database.BatchableContext bc, List<Lead> allLeads){
        List<Lead> leads = new List<Lead>();
        for(Lead l: allLeads){
            l.LeadSource='Dreamforce';
        }
        update leads;

    }
    global void finish(Database.BatchableContext bc){
        System.debug(leadsProcessed + ' leads processed. Nigga!');
        AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
            JobItemsProcessed,
            TotalJobItems, CreatedBy.Email
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()];

        EmailManager.sendMail('jgatsby1996@gmail.com','Total Leads Porcessed are ','
'+leadsProcessed);

    }
}
```

Test.apxc(Lead Processor)

```
@isTest
public class LeadProcessorTest {

    @testSetup
    static void setup(){
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(Lastname='Last '+i,
                        status='Open - Not Contacted'
             , company='LeadCompany'+i));
        }
        insert leads;
    }
    static testmethod void test() {
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

3.Control Processes with Queuable Apex:(Add primary contact.apxc)

```
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;
        }
    }

}
```

Test.apxc:(Add primary contact)

```
@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 Apex Scheduler.:(DailyLeadProcessor.apxc)

```
public class DailyLeadProcessor implements schedulable{

    public void execute(schedulableContext sc) {
        List<lead> l_lst_new = new List<lead>();
        List<lead> l_lst = new List<lead>([select id, leadsource from lead where leadsource = null]);
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
        }
        update l_lst_new;
    }

}
```

Test.apxc:(DailyLeadProcessor)

```
@isTest
```

```
public class DailyLeadProcessorTest {
   @testSetup
   static void setup(){
       List<Lead> lstOfLead = new List<Lead>();
       for(Integer i = 1; i <= 200; i++){
           Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working -
Contacted');
           lstOfLead.add(ld);
       }
       Insert lstOfLead;
   }
   static testmethod void testDailyLeadProcessorScheduledJob(){
       String sch = '0 5 12 * * ?';
       Test.startTest();
       String jobId = System.schedule('ScheduledApexTest', sch, new DailyLeadProcessor());

       List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
       System.assertEquals(200, lstOfLead.size());

       Test.stopTest();
   }
}
```

APEX SPECALIST SUPERBADGE CODES

CHALLENGE-1:AUTOMATED RECORD CREATION:

(MaintenanceRequestHelper.apxc)

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
       Set<Id> validIds = new Set<Id>();


       For (Case c : updWorkOrders){
           if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
               if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                   validIds.add(c.Id);
```

```
                }
            }
        }
 if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                         FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
        }

            for(Case cc : closedCasesM.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()

                );
 If (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                } 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> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
           for (Case nc : newCases){
               for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                   Equipment_Maintenance_Item__c wpClone = wp.clone();
                   wpClone.Maintenance_Request__c = nc.Id;
                   ClonedWPs.add(wpClone);

               }
           }
           insert ClonedWPs;
       }
    }
}
```

MaintenanceRequest.apxt:

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

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

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

    }

}
```

CHALLENGE-2:SYNCHROIZE SALESFORCE DATE 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';

    //class that makes a REST callout to an external warehouse system to get a list of
equipment that needs to be updated.
```

```
    //The callout’s JSON response returns the equipment records that you upsert in Salesforce.

    @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());

            //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

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

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

CHALLENGE-3:SCHEDULABLE SYNCHORINIZATION USING APEX CODE:
(WarehouseSyncShedule.apxc :-)

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

CHALLENGE-4:TEST AUTOMATION LOGIC:
(MaintenanceRequestHelperTest.apxc :-)

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

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

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

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

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                    Maintenance_Request__c = requestId);
        return wp;
    }
  @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

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

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

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                     from case
                     where status =:STATUS_NEW];

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

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }
@istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

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

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

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();
        list<case> allRequest = [select id
                                 from case];

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

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }
@istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

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

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

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

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

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

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

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

MaintenanceRequestHelper.apxc :-

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

CHALLENGE-5:TEST CALLOUT LOGIC:

(WarehouseCalloutService.apxc :-)

```
public with sharing class WarehouseCalloutService {



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



    //@future(callout=true)

    public static void runWarehouseEquipmentSync(){


        Http http = new Http();

        HttpRequest request = new HttpRequest();
        request.setEndpoint(WAREHOUSE_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);



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


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

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

            System.debug(response.getBody());


            for (Object eq : jsonResponse){

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

                Product2 myEq = new Product2();

                myEq.Replacement_Part__c = (Boolean)
mapJson.get('replacement');

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

                myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');

                myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');

                myEq.Cost__c = (Decimal) mapJson.get('lifespan');

                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');

                warehouseEq.add(myEq);

            }



            if (warehouseEq.size() > 0){

                upsert warehouseEq;

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

                System.debug(warehouseEq);

 }

       }
    }
}
```

WarehouseCalloutServiceTest.apxc :-

```
@isTest



private class WarehouseCalloutServiceTest {

    @isTest

    static void testWareHouseCallout(){

        Test.startTest();

        // implement mock callout test here

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();

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

    }

}
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest

global class WarehouseCalloutServiceMock implements
HttpCalloutMock {

    // 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","replace
ment":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100
003"}]');

        response.setStatusCode(200);
        return response;
    }
}
```

CHALLENGE-6:TEST SCHEDULING   LOGIC:

(WarehouseSyncSchedule.apxc :-)

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



WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public class WarehouseSyncScheduleTest {


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

        Test.stopTest();

        //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 ');





    }

}
```