APEX TRIGGERS

Get Started with Apex Triggers

Apex trigger: AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert, before update) {
        for(Account account : Trigger.New) {
      if (account.Match_Billing_Address__c == True){
        account.ShippingPostalCode = account.BillingPostalCode;
      }
   }
}
```

Bulk Apex Triggers

Apex trigger: ClosedOpportunityTrigger.apxt

```
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

Get Started with Apex Unit Tests

Apex class: VerifyDate.apxc

```
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;
        }

}
```

Apex class: TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
   private static Date dateToday = date.today();
   private static Integer totalDays = Date.daysInMonth(dateToday.year(),
dateToday.month());

   @isTest static void testOldDate(){
      Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(-1));
      System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(),
totalDays), dateTest);
```

```
    }

    @isTest static void testLessThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(20));
        System.assertEquals(dateToday.addDays(20), dateTest);
    }

    @isTest static void testMoreThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(31));
        System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(),
totalDays), dateTest);
    }

}
```

Test Apex Triggers

Apex trigger: RestrictContactByName.apxt

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

Apex class: TestRestrictContactByName.apxc

```
@isTest
private class TestRestrictContactByName {

    static testMethod void  metodoTest()
    {

        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
```

```
        Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
        listContact.add(c1);
        listContact.add(c2);

        Test.startTest();
          try
          {
             insert listContact;
          }
          catch(Exception ee)
          {
          }

        Test.stopTest();

    }

}
```

Create Test Data for Apex Tests
Apex class: RandomContactFactory.apxc

```
public class RandomContactFactory{
   public static List<Contact> generateRandomContacts(integer n,string lastname){
      integer n1= n;
      list<contact> c=[select  FirstName from contact limit : n1];
      return c;
   }
}
```

ASYNCHRONOUS APEX
Use Future Methods
Apex class: AccountProcessor.apxc

```
public class AccountProcessor {

    //Writting the countContacts method and marking it whit the @future label.
    @future
```

```
    public static void countContacts(Set<Id> accountIDs) {

        // Creating a list that will contain all those accounts that are referenced through the
accounIDs list.
        List<Account> accounts = [SELECT Id, Number_of_Contacts__c, (SELECT id FROM
Contacts) from Account where id in :accountIDs];

        //Assigment from the total contact number to the Number_of_Contacts__c field for
each account at accounts list.
        for( Account account : accounts ) {
          account.Number_of_Contacts__c = account.contacts.size();
        }

        //Updating all accounts in list
        update accounts;

    }

}
```

Apex test class: AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void countContactsTest(){
        //Creating an account and inserting it
        Account account = New Account(Name = 'Account Number 1');
        insert account;

        //Creating some contacts related to the account and inserting them
        List<Contact> contacts = new List<Contact>();
        contacts.add(New Contact(lastname = 'Related Contact 1', AccountId =
account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 2', AccountId =
account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 3', AccountId =
```

```
account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 4', AccountId =
account.Id));
        insert contacts;

        //Creating a List with account Ids to pass them throught the
AccountProcessor.countContacts method
        Set<Id> accountIds = new Set<Id>();
        accountIds.add(account.id);

        //Starting Test:
        Test.startTest();

        //Calling the AccountProcessor.countContacts method
        AccountProcessor.countContacts(accountIds);

        //Finishing Test:
        Test.stopTest();
        Account ACC = [SELECT Number_of_Contacts__c FROM Account WHERE id =
:account.Id LIMIT 1];

        //Setting Assert (We have to parse the account.Number_of_Contacts__c
        //to integer to avoid some comparasion error between decimal and integer)
        System.assertEquals( Integer.valueOf(ACC.Number_of_Contacts__c) , 4);
    }
}
```

Use Batch Apex

Apex class: LeadProcessor.apxc

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    // instance member to retain state across transactions
    global Integer recordsProcessed = 0;

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

```
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {

                lead.LeadSource = 'Dreamforce';
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;

        }
        update leads;
    }

    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');

    }
}
```

Apex class: LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }

    static testmethod void test() {
        Test.startTest();
```

```
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where LeadSource =
'Dreamforce']);
    }
}
```

Control Processes with Queueable Apex

Apex class: AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state
LIMIT 200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

Apex class: AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
   @testSetup
   static void setup(){
      List<Account> lstOfAcc = new List<Account>();
      for(Integer i = 1; i <= 100; i++){
         if(i <= 50)
            lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
         else
            lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
      }

      INSERT lstOfAcc;
   }

   static testmethod void testAddPrimaryContact(){
      Contact con = new Contact(LastName = 'TestCont');
      AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

      Test.startTest();
      System.enqueueJob(addPCIns);
      Test.stopTest();

      System.assertEquals(50, [select count() from Contact]);
   }
}
```

Schedule Jobs Using the Apex Scheduler

Apex class: 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;
    }
  }
}
```

Apex test class: 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 0 2 6 ? 2022';

  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();
  }
}
```

APEX INTEGRATION SERVICES

Apex REST Callouts

Apex test class: AnimalLocatorTest.apxc

```
@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);
   }
}
```

Apex class: 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);
        String strResp = '';
         system.debug('******response '+response.getStatusCode());
         system.debug('******response '+response.getBody());
      // If the request is successful, parse the JSON response.
      if (response.getStatusCode() == 200)
      {
          // Deserializes the JSON string into collections of primitive data types.
         Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
          // Cast the values in the 'animals' key as a list
         Map<string,object> animals = (map<string,object>) results.get('animal');
          System.debug('Received the following animals:' + animals );
          strResp = string.valueof(animals.get('name'));
          System.debug('strResp >>>>>>' + strResp );
```

```
        }
        return strResp ;
    }

}
```

Apex SOAP Callouts

Apex class: ParkLocator.apxc

```
publicocator class ParkL {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

Apex test class: ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

Apex Web Services

Apex class: 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;
  }
}
```

Apex test class: AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

  private static testMethod void getAccountTest1() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts' ;
    request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAccount = AccountManager.getAccount();
    // Verify results
    System.assert(thisAccount != null);
    System.assertEquals('Test record', thisAccount.Name);

  }

  // Helper method
    static Id createTestRecord() {
    // Create test record
    Account TestAcc = new Account(
      Name='Test record');
    insert TestAcc;
    Contact TestCon= new Contact(
    LastName='Test',
    AccountId = TestAcc.id);
    return TestAcc.Id;
  }
```

```
}
```

APEX SPECIALIST SUPERBADGE CODES

Step - 2: Automate Record Creation

Apex Class: 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));
            }

            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: MaintenanceRequest.apxt

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

Step - 3: Synchronize Salesforce data with an external system

Apex class: 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();
  }

}
```

In anonymous window:

```
System.enqueueJob(new WarehouseCalloutService());
```

Step - 4: Schedule Synchronization

Apex class: WarehouseSyncSchedule.apxc

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

Step - 5: Test Automation Logic

Apex Class: 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));
          }

          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: 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);
    }
}
```

Apex trigger: MaintenanceRequest.apxt

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

Step - 6: Test Callout Logic

Apex mock class: 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":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

Apex mock callout test class: WarehouseCalloutServiceTest.apxc

```
@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);
    }
}
```

Step - 7: Test scheduling logic

Apex class: WarehouseSyncSchedule.apxc

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

Apex test class: 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 ');
   }
}
```