Apex Triggers:

a) 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.ShippingPostalCode =      a.BillingPostalCode;
      }

      }
}
```

b) Bulk Apex Triggers : ClosedOpportunityTrigger:

```
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;
    }
}
```

Apex Testing:

a) Get Started with Apex Unit Tests:

VerifyDate:

```
public class VerifyDate {

    public static Date CheckDates(Date date1, Date date2) {
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }

        Date date30Days = date1.addDays(30); //create a date 30 days

away from date1

        if( date2 >= date30Days ) { return false; }

        else { return true; }
    }

    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
```

```
 public class TestVerifyDate {
      @istest Static Void test1(){
           Date d =
Verifydate.Checkdates(date.parse('01/01/2022'),date.parse('01/03 /2022'));
          System.assertEquals(date.parse('01/03/2022'),d);

     }
      @istest Static Void test2(){
           Date d =
Verifydate.Checkdates(date.parse('01/01/2022'),date.parse('03/03 /2022'));

         System.assertEquals(date.parse('01/31/2022'),d);


     }
}
```

b) Test Apex Triggers:

RestrictContactByName:

```
trigger RestrictContactByName on Contact (before insert, before update) {

     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
    public static void testcontact(){
        contact ct = new contact();
        ct.LastName = 'INVALIDNAME';
        database.Saveresult res = Database.insert(ct,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not
allowed for DML',res.getErrors()[0].getMessage());


    }
}
```

c) Create Test Data for Apex Tests:

RandomContactFactory:

```
public class RandomContactFactory {
    Public static List generateRandomContacts (integer num, string lastName){
        List contactlist = new list();
        for(integer i=1;i<=num;i++){
             contact ct = new contact(FirstName = 'Test' + i,LastName=
lastName);
             contactlist.add(ct);
        }
       return contactlist;
    }
}
```

Asynchronous Apex:

a) Use Future Methods:

AccountProcessor:

```
public class AccountProcessor {
@future
public static void countContacts(List accountIds){
     List accounts = [Select Id, Name from Account Where Id IN: accountIds];
     List updatedAccounts = new List();
     for (Account account :accounts){
     account.Number_Of_Contacts__c = [Select count() from Contact Where
```

```
AccountId=: account.Id];
     System.debug('No Of Contacts = '+ account.Number_Of_Contacts__c);
     updatedAccounts.add(account);
     }
     update updatedAccounts;
     }
  }
```

AccountProcessorTest:

```
 @isTest
 public class AccountProcessorTest {
    @isTest
     public static void testNoOfContacts(){
          Account a = new Account();
          a.Name = 'Test Account';
          Insert a;

          Contact c= new Contact();
          c.FirstName = 'Bob';
          c.LastName= 'Willie';
          c.AccountId = a.Id;

          Contact c2 = new Contact();
          c2.FirstName='Tom';
          c2.LastName = 'Cruise';
          c2.AccountId = a.Id;

          List acctIds = new List();
          acctIds.add(a.Id);

          Test.startTest();
          AccountProcessor.countContacts(acctIds);
          Test.stopTest();
     }
 }
```

b) Use Batch Apex: LeadProcessor:

```
 global class LeadProcessor implements
 Database.Batchable {
       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 L_list){
         List L_list_new= new List();

          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 L_list = new List();

         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();
    }
}
```

c) Control Processes with Queueable Apex:

AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable {
public contact c;
public String state;

public AddPrimaryContact(Contact c, String state) {
this.c = c;
this.state = state;
}

public void execute(QueueableContext qc) {
system.debug('this.c = '+this.c+' this.state = '+this.state);
List accList = new List([select id, name,
BillingState from account where account.BillingState =
:this.state limit 200]);
List insertContact = new List();
for(account a: accList) {
contact c = new contact();
c = this.c.clone(false, false, false, false);
c.AccountId = a.Id;
insertContact.add(c);
}
insert insertContact;
}
}
```

AddPrimaryContactTest:

```
@isTest
public class AddPrimaryContactTest {

@testSetup
static void setup() {
List insertAccount = new List();
for(integer i=0; i<=100; i++) {
```

```
 if(i <=50) {
 insertAccount.add(new Account(Name='Acc'+i, BillingState = 'NY'));
 } else {
 insertAccount.add(new Account(Name='Acc'+i, BillingState = 'CA'));
 }
 }
 insert insertAccount;
 }

 static testMethod void testAddPrimaryContact() {
 Contact con = new Contact(LastName = 'LastName');
 AddPrimaryContact addPC = new AddPrimaryContact(con, 'CA');
Test.startTest();
system.enqueueJob(addPC);
 Test.stopTest();

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

b) Schedule Jobs Using the Apex Scheduler:

DailyLeadProcessor:

```
global class DailyLeadProcessor implements Schedulable {

   global void execute(SchedulableContext ctx) {

       List leads = [SELECT ID, LeadSource FROM Lead where LeadSource = ''
LIMIT 200];
       for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
       }
        update leads;
     }
 }
```

DailyLeadProcessorTest:

```
@isTest
```

```
 private class DailyLeadProcessorTest {

     @isTest
     public static void testDailyLeadProcessor(){

         List leads = new List();
         for (Integer x = 0; x < 200; x++) {
              leads.add(new Lead(lastname='lead number ' + x,
company='company number ' + x));
         }
         insert leads;
         Test.startTest();
         String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?',
new DailyLeadProcessor());
         Test.stopTest();
         List listResult = [SELECT ID, LeadSource FROM Lead where LeadSource
= 'Dreamforce' LIMIT 200];
         System.assertEquals(200, listResult.size());
     }
 }
```

Apex Integration Services:

a) Apex REST Callouts:

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/anim als/' + x);
     req.setMethod('GET');
     Map animal= new Map();
```

```
    HttpResponse res = http.send(req);
        if (res.getStatusCode() == 200) {
    Map results = (Map)JSON.deserializeUntyped(res.getBody());
 animal = (Map) results.get('animal');
   }
 return (String)animal.get('name');
   }
   }
```

AnimalLocatorTest:

```
    @isTest
    private class AnimalLocatorTest{
        @isTest static void AnimalLocatorMock1() {
           Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
           string result = AnimalLocator.getAnimalNameById(3);
           String expectedResult = 'chicken';
        }
    }
```

AnimalLocatorMock:

```
 @isTest
 global class AnimalLocatorMock implementsHttpCalloutMock {
    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;
     }
 }
```

b) Apex SOAP Callouts: ParkLocator:

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort parkSvc = new
ParkService.ParksImplPort(); // remove space
       return parkSvc.byCountry(theCountry);
    }
```

ParkLocatorTest:

```
 @isTest
 private class ParkLocatorTest {
     @isTest static void testCallout() {
         Test.setMock(WebServiceMock.class, new ParkServiceMock ());
         String country = 'United States';
         List result = ParkLocator.country(country);
         List parks = new List{'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{'Yellowstone', 'Mackinac National
Park', 'Yosemite'};
       // end
       response.put('response_x', response_x);
     }
 }
```

c) Apex Web Services:

AccountManager:

```
@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:

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://na1.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 TestAcc = new Account( Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
   }
}
```

Apex Specialist Superbadge:

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 newCases = new List();
           Map closedCasesM = new Map([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_Maintenanace_Item__c> clonedWPs = new
List<Equipment_Maintanence_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;

             }

         }

    }
```

MaitenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){

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

```
}
```

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(){
      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)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 = (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();
    }
}
```

3.Schedule synchronization using Apex code

WarehouseSyncShedule.apxc :-

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

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<Product__2> equipmentList = new list<Product__2();
     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 allRequests = [select id
                          from case
                          where status =: STATUS_NEW];
      list<Equipment_Maintanance_Item__c> workParts = [select
id
                                                       from
Equipment_Maintenance_Item__c
                                                       where
Maintenance_Request__c in: oldRequestIds];

       system.assert(allRequests.size() == 300);
   }
}
 MaintenanceRequestHelper.apxc :-
 public with sharing class MaintenanceRequestHelper {
     public static void updateworkOrders(List<Case>
 updWorkOrders, Map nonUpdCaseMap) {
        Set validIds = new Set();

        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;
        }
    }
 }
```

MaintenanceRequest.apxt :-

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

5.Test callout logic

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {
   private static final String WAREHOUSE_URL =
 'https://th-superbadge-apex.herokuapp.com/equipment';

    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 jsonResponse =
(List)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();
      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 {
    global static HttpResponse respond(HttpRequest request){


System.assertEquals('https://th-superbadge-apex.herokuapp.com/eq uipment',
request.getEndpoint());
       System.assertEquals('GET', request.getMethod());
       HttpResponse response = new HttpResponse();
       response.setHeader('Content-Type', 'application/json');


response.setBody('[{"_id":"55d66226726b611100aaf741","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10 0003"}]');
     response.setStatusCode(200);
     return response;
  }
}
```

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();

          CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime >
today];
           System.assertEquals(jobID, a.Id,'Schedule ');

       }
}
```