MODULE WISE CODE USED IN THE ENTIRE LEARNING PROGRAM

Apex Triggers

🚩 Getting Started with Apex Triggers

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

ClosedOppurtunityTrigger.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

🚩 Getting Started with Unit Tests

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;
    }

}
```

TestVerifyDate.apxc

```
@IsTest
public class TestVerifyDate {
    @isTest static void date2within30daydate1() {
        Date returnDate1 = VerifyDate.CheckDates(date.valueOf('2022-

        //this should return may 24,2022 because it is within 30
 days of may 22nd
        System.assertEquals(date.valueOf('2022-06-01'),
 returnDate1);
    }
    @isTest static void date2NOTwithin30daydate1() {
        Date returnDate2 = VerifyDate.CheckDates(date.valueOf('2022-

         //this should return may 31,2022 because it is not within
 30 days of may 22 as its june 24th
         System.assertEquals(date.valueOf('2022-05-31'),
 returnDate2);
     }

}
```

Test Apex Triggers

RandomContactByName.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

        }

    }



}
```

TestRandomContactFactory.apxc

```
@IsTest
public class TestRestrictContactByName {
    @IsTest static void createBadContact(){
        Contact c = new Contact(FirstName='John', 
LastName='INVALIDNAME');

        Test.startTest();
        Database.SaveResult result=Database.insert(c,false);
        Test.stopTest();

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

}
```

Create Test Data for Apex Tests

RandomContactFactory.apxc

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer 
numOfContacts, String lastName){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numOfContacts;i++){
            Contact c = new Contact(FirstName='Test

            contacts.add(c);
        }
        system.debug(contacts);
        return contacts;

    }
}
```

Asynchronous Apex

Use Future Methods

AccountProcess.apxc

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds) {

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

    List<Account> accounts = [Select Id, Name,(Select Id from
Contacts) from Account Where Id IN :accountIds];
    // process account records to do awesome stuff
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c =contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
  }

}
```

AccountProcessorTest.apxc

```
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts() {
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;
        Contact newContact1 =new Contact(FirstName='John',
                                         LastName='Doe',
                                         AccountId=newAccount.Id);
        insert newContact1;
        Contact newContact2 =new Contact(FirstName='Jane',
                                         LastName='Doe',
                                         AccountId=newAccount.Id);
        insert newContact2;
        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

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

Use Batch Apex

LeadProcessor.apxc

```
public class LeadProcessor implements
    Database.Batchable<sObject>{

    public Database.QueryLocator start(Database.BatchableContext
bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead '
        );
    }
    public 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';
            leads.add(lead);
        }

    }
    public void finish(Database.BatchableContext bc){

    }
}
```

LeadProcessorTest.apxc

```
@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='name'+i,Company='test'));
        }
        insert leads;
    }

    @isTest static void test() {
        Test.startTest();
        LeadProcessor uca = new LeadProcessor();
        Id batchId = Database.executeBatch(uca);
        Test.stopTest();

        System.assertEquals(200, [select count() from Lead where
LeadSource = 'Dreamforce']);
    }
}
```

🚩 Control Processes with Queueable Apex

AddPrimaryConatct.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> lstAcc = [Select id , name , BillingState, (Select id , FirstName , LastName FROM Contacts)
                             FROM Account WHERE BillingState =:state LIMIT 200];
        List<Contact> lstcon = new List<Contact>();
        for(Account Acc:lstAcc){
            Contact cnt = con.clone(false);
            cnt.AccountId = Acc.Id;
            lstcon.add(cnt);
        }
        if(lstcon.size()>0){
            insert lstcon ;
        }
    }
}
```

AddPrimaryConatctTest.apxc

```
@isTest
public class AddPrimaryContactTest {
     @isTest
    Public static void testContact(){
        List<Account> Acc = New List<Account>();
        for (integer i = 1; i<=50 ; i++){
            Acc.add(new Account (BillingState = 'NY', name='Test'+i));
        }
        for (integer j=1 ; j<=50 ; j++){
            Acc.add(new Account (BillingState = 'CA', name='Test'+j));
        }
        insert Acc ;

        contact con = new contact();
        con.FirstName = 'Satyam';
        con.LastName = 'Pandey';
        insert con;
        String state = 'CA' ;

        AddPrimaryContact AC = New AddPrimaryContact(con , state);
        test.startTest();
        system.enqueueJob(AC);
        test.stopTest();
    }
}
```

Schedule Jobs Using Apex Scheduler

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements schedulable
{
    public void execute(SchedulableContext sct)
    {
        List<Lead> leadList = [Select id,lastname,company,status,leadsource from Lead where Leadsource = null LIMIT 200];
        List<Lead> insertLeadList = new List<Lead>();
        for(Lead led:leadList)
        {
            led.LeadSource = 'DreamForce';
            insertLeadList.add(led);
        }
        if(insertLeadList.size() > 0)
        {
            update insertLeadList;
        }
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest
{
    public static String CRON_EXP = '25 2 0 8 10 ?';
    @testSetup
    public static void setup()
    {
        List<Lead> leadList = new List<Lead>();
        for(Integer i=0;i<200;i++)
        {
            leadList.add(new
Lead(lastname='paul'+i,company='p.Tech'+i,state='working'));
        }
        insert leadList;
    }
    @isTest
    public static void test()
    {
        Test.startTest();
        Id jobId = System.schedule('Daily Lead

        Test.stopTest();
        System.assertEquals(200,[select count() from Lead where
Leadsource = 'DreamForce']);
    }
}
```

Apex Integration Services

Apex REST Callouts

AnimalLocator.apxc

```
public class AnimalLocator {

    public static String getAnimalNameById(Integer id){

        String animal=null;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if (response.getStatusCode() == 200) {
            // Deserializes the JSON string into collections of primitive data types.
            System.debug('Received the following response :' + response.getBody());
            Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(response.getBody());
            Map<string,object> animals = (map<string,object>) results.get('animal');
               animal = string.valueof(animals.get('name'));
            // Cast the values in the 'animals' key as a list

        }
        System.debug('Received the following animal:' + animal);
        return animal;
    }
}
```

AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock  implements HttpCalloutMock{

    // Implement this interface method
    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":"chicken","eats":"chic

        response.setStatusCode(200);
        return response;

    }
}
```

AnimalLocatorTest.apxc

```
@isTest
public class AnimalLocatorTest {

    @isTest static  void testGetCallout() {
           Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
        // Call method to test
        String result = AnimalLocator.getAnimalNameById(3);
         string expectedResult='chicken';
        System.assertEquals(result, expectedResult);

        // System.assert(String.contains(result));
    }

}
```

Apex SOAP Callouts

ParkService.apxc

```
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-

        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ParkLocator.apxc

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

ParkServiceMock.apxc

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
        // start - specify the response you want to send
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone',
'Mackinac National Park', 'Yosemite'};
        // end
        response.put('response_x', response_x);
   }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock
());
```

```
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
         System.assertEquals(parks, result);
    }

```

Apex Web Services

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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 SUPERBADGE

CreateDefaultData.apxc

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine

    //gets value from custom metadata
How_We_Roll_Settings__mdt to know if Default data was
created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest =
createMaintenanceRequest(vehicles);
```

```
        List<Equipment_Maintenance_Item__c> joinRecords =
createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }


    public static void updateCustomSetting(Boolean
isDataCreated){
        How_We_Roll_Settings__c customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
1, Model__c = 'Toy Hauler RV'));
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer

Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop

Bedrooms__c = 1, Model__c = 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
1, Model__c = 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c =
'55d66226726b611100aaf741',name = 'Generator 1000 kW',
Replacement_Part__c = true,Cost__c = 100
,Maintenance_Cycle__c = 100));
        equipments.add(new Product2(name = 'Fuse

Maintenance_Cycle__c = 30  ));
        equipments.add(new Product2(name = 'Breaker

Maintenance_Cycle__c = 15));
        equipments.add(new Product2(name = 'UPS 20

Maintenance_Cycle__c = 60));
        insert equipments;
        return equipments;

    }

    public static List<Case>
createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c =
vehicles.get(1).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c =
vehicles.get(2).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c>
createJoinRecords(List<Product2> equipment, List<Case>
maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords =
new List<Equipment_Maintenance_Item__c>();
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c =
maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c =
maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c =
maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c =
maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c =
maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c =
maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest.apxc

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM
Vehicle__c];
```

```
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There

        System.assertEquals(4, equipment.size(), 'There

        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There


    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting


        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting

    }
}
```

MaintenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update) {
    //ToDo: Call MaintenanceRequestHelper.updateWorkOrders
    if(trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders();
    }
}
```

MaintenanceRequestHelper.apxc

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There
        System.assertEquals(4, equipment.size(), 'There
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false,
CreateDefaultData.isDataCreated(), 'The custom setting


        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true,
CreateDefaultData.isDataCreated(), 'The custom setting


    }
}
```

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {
    @istest
    public static void BulkTesting(){
        product2 pt2 = new product2(Name =
'tester',Maintenance_Cycle__c = 10, Replacement_Part__c =
true);
```

```

        Database.insert(pt2);


        List<case> caseList = new List<case>();
        for(Integer i=0;i<300;i++){
            caseList.add(new case(
                Type = 'Routine Maintenance',
                Status = 'Closed',
                Subject = 'testing',
                Date_Reported__c = Date.today(),
                ProductId = pt2.id
            ));
        }
        if(caseList.size()>0){
            Database.insert(caseList);
            System.debug(pt2.id);
            System.debug(caseList.size());
        }


        List<Equipment_Maintenance_Item__c> newEMI = new List<Equipment_Maintenance_Item__c>();
        for(Integer i=0;i<5;i++){
            newEMI.add(new Equipment_Maintenance_Item__c(
                Equipment__c = pt2.id,
                Maintenance_Request__c = caseList[1].id,
                Quantity__c = 10));
        }
        if(newEmi.size()>0){
            Database.insert(newEmi);
        }

        for(case c :caseList){
            c.Subject = 'For Testing';
        }
        Database.update(caseList);
        Integer newcase = [Select count() from case where ParentId = :caseList[0].id];
        System.assertEquals(1, newcase);

    }

    @istest
    public static void positive(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
        insert pt2;

        Case cParent = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c = Date.today(),
                                        ProductId = pt2.id);
        insert cParent;
        Case cChild = new Case(Type = 'Repair',status = 'Closed',Date_Reported__c = Date.today(),
                                       ProductId = pt2.id,parentID = cParent.ParentId);
        insert cChild;

        cParent.subject = 'child refrecer record';
        update cParent;

        Integer newcase = [Select count() from case where ParentId = :cParent.id];
        System.assertEquals(1, newcase);

    }
    @istest public static void negetive(){
        product2 pt2 = new product2(Name = 'tester',Maintenance_Cycle__c = 10);
        insert pt2;

        Case c = new Case(Type = 'Repair',status =
'New',Date_Reported__c = Date.today(),
                            ProductId = pt2.id);
        insert c;

        c.Status = 'Working';
        update c;


        Integer newcase = [Select count() from case where
ParentId = :c.id];
        System.assertEquals(0, newcase);
    }




}
```

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService
implements Queueable, Database.AllowsCallouts{
    public List<product2> equip = new List<product2>();
    private static final String WAREHOUSE_URL =
'https://th-superbadge-apex.herokuapp.com/equipment';



    public void execute(QueueableContext context) {
        //System.debug('Equipments'+equip );
        Http h = new Http();
        HttpRequest httpReq = new HttpRequest();
        httpReq.setMethod('GET');
        httpReq.setHeader('Content-

        httpReq.setEndpoint(WAREHOUSE_URL);
        HttpResponse res = h.send(httpReq);
        List<Object> results = (List<Object>) JSON.deserializeUntyped(res.getBody());
        System.debug(results.size());

        for(Object fld : results){
            Map<String,Object> entry = (Map<String,Object>)fld;
            equip.add(new product2(
                Warehouse_SKU__c = String.valueOf(entry.get('_id')+''),
                Cost__c = Decimal.valueOf(entry.get('cost')+''),
                Lifespan_Months__c = Decimal.valueOf(entry.get('lifespan')+'') ,
                Maintenance_Cycle__c = Decimal.valueOf(entry.get('maintenanceperiod')+''),
                Name = String.valueOf(entry.get('name')+''),
                QuantityUnitOfMeasure = String.valueOf(entry.get('quantity')+'') ,
                Replacement_Part__c = Boolean.valueOf(entry.get('replacement') +''),
                StockKeepingUnit = String.valueOf(entry.get('sku')+'')
            ));
        }
        if(!equip.isEmpty())
        {
            upsert equip Warehouse_SKU__c;
            system.debug('list got updated. Size:

        }

    }
}
```

WarehouseCalloutServiceMock.apxc

```
@istest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock{
    // implement http mock callout
    global HttpResponse respond(HttpRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","repla




        response.setStatusCode(200);
        return response;
    }

}
```

WarhouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @isTest static void mainTest(){
        Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        Test.startTest();
        Id jobID = System.enqueueJob(new
WarehouseCalloutService());
        //System.assertEquals('Queued',aaj.status);
        Test.stopTest();
        AsyncApexJob aaj = [SELECT Id, Status,
NumberOfErrors FROM AsyncApexJob WHERE Id = :jobID];
        System.assertEquals('Completed',aaj.status);
        System.assertEquals(0, aaj.NumberOfErrors);
    }
}
```

WarehouseSyncSchedule.apxc

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    // implement scheduled code here
    global void execute(SchedulableContext sc){
        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

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 ');

```

```

    }
}
```