APEX TRIGGERS

AccountAddressTrigger.apxt:

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

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

ClosedOpportunityTrigger.apxt:

```apex
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
        }
    }

    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

APEX TESTING

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
 @TestVisible 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
    @TestVisible 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
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = 
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.p

        system.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = 
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.p

        system.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = 
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.p

        system.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = 
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }

}
```

TestRestrictContactByname.apxc:

```
@isTest

public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

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

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is

    }
}
```

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
        }

    }



}
```

RandomContactFactory.apxc:

```
public class RandomContactFactory {

    public static List<Contact>
generateRandomContacts(Integer numcnt, string lastname){

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

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

            Contact cnt = new Contact(FirstName = 'Test


            contacts.add(cnt);

        }

        return contacts;

    }

}
```

ASYNCHRONOUS APEX

AccountProcessor.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];

        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

        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();


    }
}
```

AddPrimaryContact.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;
        }
    }

}
```

AddPrimaryContactTest.apxc:

```
@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

        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account

        }
        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')]);
    }
}
```

DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

DailyLeadProcessorTest.apxc:

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 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

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

LeadProcessor.apxc:

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID,

    }

    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

        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.apxc:

```
@isTest
private class LeadProcessorTest
{
    private static testMethod void LeadProcess()
    {
        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            lstLead.add(new Lead(LastName ='LastName'+i, Company ='demo'+i, City='New York', Country='US', LeadSource='Phone inquiry'));
        }

        insert lstLead;

        Test.startTest();

            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);

        Test.stopTest();
    }
}
```

APEX INTEGRATION SERVICES

AccountManager.apxc:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

AccountManagerTest.apxc:

```
@isTest
private class AccountManagerTest {

    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/'+re

        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
;
    }
}
```

AnimalLocator.apxc:

```
public class AnimalLocator {
 public class cls_animal {
      public Integer id;
      public String name;
      public String eats;
      public String says;
 }
public class JSONOutput{
 public cls_animal animal;

 //public JSONOutput parse(String json){
 //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
 //}
}

    public static String getAnimalNameById (Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-

        //request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        //Map<String,Object> map_results = (Map<String,Object>) JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(), jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
      system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }

}
```

AnimalLocatorMock.apxc:

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats"
        return response;
    }

}
```

AnimalLocatorTest.apxc:

```
@IsTest
```

```
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}

```

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);
    }
}
```

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','-

        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','fals

        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;
        }
    }
}
```

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);
    }
}
```

APEX SPECIALIST SUPERBADGE

MaintenanceRequest.apxt:

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

MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders() {
        List<case> newCaseList = new List<case>();
        Integer avgAmount=10000;

        List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
        List<case> caseList = [SELECT
id,Vehicle__c,Subject,ProductID,Product__c, (SELECT id from
Equipment_Maintenance_Items__r) from case where
status='closed' and Type IN ('Repair', 'Routine
        Map<id,Equipment_Maintenance_Item__c> equip = new map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c, Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle_

        for(case c: caseList){
            case newCase = new Case();
            newCase.Type = 'Routine Maintenance';
            newCase.Status = 'New';
            newCase.Vehicle__c = c.Vehicle__c;
            newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request' : c.Subject;
            newCase.Date_Reported__c = Date.today();
            newCase.ProductId = c.ProductId;
            newCase.Product__c = c.Product__c;
            newCase.parentID = c.Id;


            for(Equipment_Maintenance_Item__c emi : c.Equipment_Maintenance_Items__r ){
                avgAmount = Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipm

                newEMI.add(new Equipment_Maintenance_Item__c(
                    Equipment__c = equip.get(emi.id).Equipment__c,
                    Maintenance_Request__c = c.id,
                    Quantity__c = equip.get(emi.id).Quantity__c));
            }
            Date dueDate = date.TODAY().adddays(avgAmount);
            newCase.Date_Due__c =dueDate;
            newCaseList.add(newCase);

        }
```

```
        if(newCaseList.size()>0){
            Database.insert(newCaseList);
        }

        for(Case c2: newCaseList){
            for(Equipment_Maintenance_Item__c emi2 : newEmi){
                if(c2.parentID == emi2.Maintenance_Request__c){
                    emi2.Maintenance_Request__c = c2.id;
                }
            }
        }

        if(newEmi.size()>0){
            Database.insert(newEmi);
        }
    }
}
```

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:

    }

  }
```

```
}
```

WarehouseSyncSchedule.apxc:

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    // implement scheduled code here
    global void execute(SchedulableContext sc){
        System.enqueueJob(new WarehouseCalloutService());

    }
}
```

MaintenanceRequest.apxt:

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

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__

                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);

                }
            }
            insert ClonedWPs;
        }
    }
```

```
}
```

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;
    }

}
```

WarehouseCalloutServiceTest.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 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

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 ');


    }
}
```