APEX TRIGGERS

ACCOUNT ADDRESS TRIGGER:

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

    for(Account a:Trigger.New){

            if(a.Match_Billing_Address__c==true)

            {

                a.ShippingPostalCode=a.BillingPostalCode;

            }

}

}
```

CLOSED OPPORTUNITY TRIGGER:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {

    List<Task> tList = new List<Task>();

    for(Opportunity o:Trigger.new) {

        if(Trigger.isInsert) {

            if(o.StageName == 'Closed Won') {

                tList.add(new Task(Subject = 'Follow Up Test Task', WhatId = o.Id));

            }

        }

        if(Trigger.isUpdate) {

            if(o.StageName == 'Closed Won' && o.StageName != Trigger.oldMap.get(o.Id).StageName)
{

                tList.add(new Task(Subject='Follow Up Test Task',WhatId =o.Id));

            }

        }

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

APEX TESTING

VERIFY DATE:

```
public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2.   Otherwise use the end of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
    private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
            if( date2 < date1) { return false; }

            //check that date2 is within (>=) 30 days of date1
            Date date30Days = date1.addDays(30); //create a date 30 days away from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }


    //method to return the end of the month of a given date
    private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
        return lastDay;
    }


}
```

TEST VERIFY DATE:

```
@isTest
private class TestVerifyDate
    {
      @isTest static void testCheckDatesOne ()
      {
        Date test = VerifyDate.CheckDates (Date.newInstance(2018, 7, 19), Date.newInstance(2018, 7,
20));
        System.assertEquals(Date.newInstance(2018, 7, 20), test);
      }


    @isTest static void testCheckDatesTwo ()
    {
        Date test = VerifyDate.CheckDates (Date.newInstance(2018, 7, 19), Date.newInstance(2018, 8,
20));
        System.assertEquals(Date.newInstance(2018, 8, 20), test);
        }


        @isTest static void testDateWithin30DaysOne ()

        {

                boolean test = VerifyDate.DateWithin30Days (Date.newInstance(2018, 7, 19),
Date.newInstance(2018, 7, 18));

                System.assertEquals(false, test);

        }


        @isTest static void testDateWithin30DaysTwo ()

        {

                boolean test = VerifyDate.DateWithin30Days (Date.newInstance(2018, 7, 19),
Date.newInstance(2019, 1, 1));

                System.assertEquals(false, test);

        }


        @isTest static void testDateWithin30DaysThree ()

        {

                boolean test = VerifyDate.DateWithin30Days (Date.newInstance(2018, 7, 19),
Date.newInstance(2018, 7, 19));

                System.assertEquals(true, test);

        }


        @isTest static void testSetEndOfMonthDate ()

        {

                Date test = VerifyDate.SetEndOfMonthDate (Date.newInstance(2018, 7, 19));
                System.assertEquals(Date.newInstance(2018, 7, 31), test);

        }


  }
```

TEST APEX TRIGGERS

RESTRICT CONTACT BY NAME:

```
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');

        }

    }

}
```

TEST RESTRICT CONTACT BY NAME:

```
@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();
          }
    }
```

CONTACT TEST FACTORY

RANDOM CONTACT FACTORY:

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


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

ASYNCHRONOUS APEX:

ACCOUNT PROCESSOR:

```
public class AccountProcessor {

    @future

    public static void countContacts(List<Id> accountIds){

        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];

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

        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;

    }

}
```

ACCOUNT PROCESSOR TEST:

```
@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<Id> acctIds = new List<Id>();

            acctIds.add(a.Id);

                        Test.startTest();

            AccountProcessor.countContacts(acctIds);

            Test.stopTest();

        }

}
```

LEAD PROCESSOR:

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

      public Database.QueryLocator start(Database.BatchableContext bc) {

            // collect the batches of records or objects to be passed to execute

               return Database.getQueryLocator([Select LeadSource From Lead ]);

        }

        public void execute(Database.BatchableContext bc, List<Lead> leads){

              // process each batch of records

                  for (Lead Lead : leads) {

                        lead.LeadSource = 'Dreamforce';

                  }

            update leads;

        }
        public void finish(Database.BatchableContext bc){

           }

}
```

LEAD PROCESSOR TEST:

```
@isTest

public class LeadProcessorTest {

                @testSetup

        static void setup() {

                List<Lead> leads = new List<Lead>();

                for(Integer counter=0 ;counter <200;counter++){

                        Lead lead = new Lead();

                        lead.FirstName ='FirstName';

                        lead.LastName ='LastName'+counter;

                        lead.Company ='demo'+counter;

                        leads.add(lead);

                }

                insert leads;

        }

        @isTest static void test() {

                Test.startTest();

                LeadProcessor leadProcessor = new LeadProcessor();

                Id batchId = Database.executeBatch(leadProcessor);

                Test.stopTest();

        }

}
```

ADD PRIMARY CONTACT:

```
public class AddPrimaryContact implements Queueable

{

        private Contact c;

        private String state;

        public    AddPrimaryContact(Contact c, String state)

        {

                this.c = c;

                this.state = state;

        }

        public void execute(QueueableContext context)

        {

                  List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts )
FROM ACCOUNT WHERE BillingState = :state LIMIT 200];

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

                  for (Account acc:ListAccount)

                  {

                                  Contact cont = c.clone(false,false,false,false);

                                  cont.AccountId =    acc.id;

                                  lstContact.add( cont );

                  }

                  if(lstContact.size() >0 )

                  {

                          insert lstContact;

                  }

        }
}
```

ADD PRIMARY CONTACT TEST:

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';
          AddPrimaryContact apc = new AddPrimaryContact(co, state);
          Test.startTest();
              System.enqueueJob(apc);
```

```
                Test.stopTest();

            }

  }
```

DAILY LEAD PROCESSOR:

```
public class DailyLeadProcessor implements Schedulable    {

        Public void execute(SchedulableContext SC){

              List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];

                for(Lead l:LeadObj){

                        l.LeadSource='Dreamforce';

                        update l;

                }

        }

}
```

DAILY LEAD PROCESSOR TEST:

```
@isTest

private class DailyLeadProcessorTest {

             static testMethod void testDailyLeadProcessor() {

                          String CRON_EXP = '0 0 1 * * ?';

                          List<Lead> lList = new List<Lead>();

                     for (Integer i = 0; i < 200; i++) {

                                       lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));

                          }

                          insert lList;

                          Test.startTest();

                          String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());

            }

}
```

APEX INTEGRATION SERVICES

ANIMAL LOCATOR:

```
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/animals/' + x);

                req.setMethod('GET');

                Map<String, Object> animal= new Map<String, Object>();

                HttpResponse res = http.send(req);

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

                Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());

            animal = (Map<String, Object>) results.get('animal');

                }

return (String)animal.get('name');

        }

}
```

ANIMAL LOCATOR TEST:

```
@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 );

        }

}
```

PARK SERVICE:

```
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-service.herokuapp.com/service/parks';

            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;

        }

    }
```

```
}
```

PARK LOCATOR:

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

PARK LOCATOR TEST:

```
@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);
    }
}
```

ACCOUNT MANAGER:

```
@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;

    }

}
```

ACCOUNT MANAGER TEST:

```
@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 SPPECIALIST

CREATE DEFAULT DATA:

```
public with sharing class CreateDefaultData{
        Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
        //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default data was
created
  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
        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 RV', Air_Conditioner__c = true,
Bathrooms__c = 2, Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));

        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper', Air_Conditioner__c = true,
Bathrooms__c = 1, 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 20B',Replacement_Part__c = true,Cost__c =
1000, Maintenance_Cycle__c = 30    ));

            equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part__c = true,Cost__c =
100    , Maintenance_Cycle__c = 15));

            equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part__c = true,Cost__c =
200    , 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;

    }

}
```

CREATE DEFAULT DATA TEST:

```
@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 should have been 4 vehicles created');

        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');

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

        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance
items created');
```

```
        }


        @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
How_We_Roll_Settings__c.Is_Data_Created__c should be false');


                customSetting.Is_Data_Created__c = true;
                upsert customSetting;


                System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be true');


        }
}
```

MAINTENANCE REQUEST HELPER:

```
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;
        }
    }
}
```

MAINTENANCE REQUEST HELPER TEST:

```
@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);

    }

}
```

WAREHOUSE CALLOUT SERVICE:

```
public with sharing class WarehouseCalloutService implements Queueable {

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


        //Write a 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(){

            System.debug('go into runWarehouseEquipmentSync');

            Http http = new Http();

            HttpRequest request = new HttpRequest();


            request.setEndpoint(WAREHOUSE_URL);

            request.setMethod('GET');

            HttpResponse response = http.send(request);


            List<Product2> product2List = new List<Product2>();

            System.debug(response.getStatusCode());

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

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

                System.debug(response.getBody());


                //class maps the following fields:

                //warehouse SKU will be external ID for identifying which equipment records to update
    within Salesforce

                for (Object jR : jsonResponse){

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

                    Product2 product2 = new Product2();

                    //replacement part (always true),

                    product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

                    //cost

                    product2.Cost__c = (Integer) mapJson.get('cost');

                    //current inventory
```

```
}
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }

        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}
```

```
}

WAREHOUSE CALLOUT SERVICE MOCK:

global class WarehouseCalloutServiceMock implements HttpCalloutMock {

        // implement http mock callout

        //Mock responce created    to test the call out

        global HttpResponse respond(HttpRequest request){

                System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());

                System.assertEquals('GET', request.getMethod());

                HttpResponse response = new HttpResponse();

                response.setHeader('Content-Type', 'application/json');


response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Ge
nerator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');

                response.setStatusCode(200);

                return response;

        }

}

WAREHOUSE CALLOUT SERVICE TEST:

@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);
        }
}
```

WAREHOUSE SYNC SCHEDULE:

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


        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WAREHOUSE SYNC SCHEDULE TEST:

```
@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 ');




    }

}
```

MAINTENANCE REQUEST:

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

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

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

    }

}
```