APEX TRIGGERS

Get Started with Apex Triggers

Trigger Name : AccountAddressTrigger

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

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

Bulk Apex Triggers

Trigger Name : ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update){
   List<Task> taskList= new List<Task>();
   for(Opportunity opp: Trigger.New){
      if(opp.StageName == 'Closed Won'){
          taskList.add(new Task(Subject='Follow Up Test Task',
                       WhatId=opp.Id));
      }
    }
   if(taskList.size() > 0){
     insert taskList;
   }
}
```

APEX TESTING

Get Started with Apex Unit Tests

Class Name : VerifyDate

```
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;
 }

}
```

Class Name : TestVerifyDate

```
@isTest
private class TestVerifyDate {
 @isTest static void date2within30Daysofdate1() {
    Date returndate1 = VerifyDate.CheckDates(date.valueOf('2022-02-14'),date.valueOf('2022-
02-24'));
    System.assertEquals(date.valueOf('2022-02-24'), returndate1);
  }
```

```
    @isTest static void date2Notwithin30Daysofdate1() {
        Date returndate2 = VerifyDate.CheckDates(date.valueOf('2022-02-14'),date.valueOf('2022-
04-24'));
        System.assertEquals(date.valueOf('2022-02-28'), returndate2);
    }
}
```

Test Apex Triggers

Trigger Name : RestrictContactByName

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

  //check contacts prior to insert or update for invalid data
  For (Contact c : Trigger.New) {
    if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid
      c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
    }

  }
}
```

Class Name : TestRestrictContactByName

```
@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 a Contact Test Factory

Class Name : RandomContactFactory

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer noofcontacts,String
lastname){
      List<Contact> contacts = new List<Contact>();
      for(Integer i=0 ; i < noofcontacts ; i++){
         Contact c=new Contact(FirstName='Test '+i,LastName=lastname);
        contacts.add(c);
      }
      System.debug(contacts);
      return contacts;
   }
}
```

ASYNCHRONOUS APEX

Use Future Methods

Class Name : AccountProcessor

```
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;
   }
}
```

Class Name : AccountProcessorTest

```
@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

Class Name : LeadProcessor

```
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){
        List<Lead> leads = new List<Lead>();
        for (Lead led : scope) {
           led.LeadSource = 'Dreamforce';
```

```
          leads.add(led);
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){

    }
}
```

Class Name : LeadProcessorTest

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new lead(LastName='Lead '+i,Company = 'mylead co'));
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor mylead = new LeadProcessor();
    Id batchId = Database.executeBatch(mylead);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce']);
    }
}
```

Control Processes with Queueable Apex

Class Name : AddPrimaryContact

```
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> primarycontact =new List<Contact>();
        for(Account acc: accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primarycontact.add(c);
        }
        if(primarycontact.size()>0){
            insert primarycontact;
        }
    }
}
```

Class Name : AddPrimaryContactTest

```
@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 '+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testaccounts.add(new Account(Name = 'Account '+j,BillingState='NY'));
        }
        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')]);
    }
}
```

Schedule Jobs Using the Apex Scheduler

Class Name : DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {
        List<Lead> updatelead = new List<Lead>();
        List<Lead> leads = [SELECT Id
            FROM Lead
            WHERE LeadSource = NULL Limit 200];
        for(Lead a : leads){
            a.LeadSource = 'Dreamforce';
            updatelead.add(a);
        }
        update updatelead;
    }
}
```

Class Name : DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
    static testmethod void testScheduledJob() {
        List<Lead> leads = new List<Lead>();

        for (Integer i=0; i<200; i++) {
            Lead l = new Lead(
                FirstName = 'First ' + i,
                LastName = 'LastName',
                Company = 'Com co'
            );
            leads.add(l);
        }
        insert leads;

        Test.startTest();
        DailyLeadProcessor ab = new DailyLeadProcessor();
        String jobId = System.Schedule('jobName','0 5 * * * ?',ab);
        Test.stopTest();
        List<Lead> checkleads = new List<Lead>();
        checkleads = [SELECT Id
            FROM Lead
            WHERE LeadSource = 'Dreamforce' and Company = 'Com co'];
        System.assertEquals(200,
            checkleads.size(),
            'Leads were not created');
    }
}
```

APEX INTEGRATION SERVICES

Apex REST Callouts

Class Name: AnimalLocator

```
public class AnimalLocator {
  public static String  getAnimalNameById(Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
         Map<String, Object> result = (Map<String, Object>)
                   JSON.deserializeUntyped(response.getBody());
         Map<String, Object> animal = (Map<String, Object>) result.get('animal');
         System.debug('name: '+string.valueOf(animal.get('name')));
       return string.valueOf(animal.get('name'));
    }
}
```

Class Name: AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {
    @isTest
    static void animalLocatorTest1(){
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
     String expected = 'moose';
        System.assertEquals(actual, expected);
      }
}
```

Class Name : AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('ContentType', 'application/json');

response.setBody('{"animal":{"id":1,"name":"moose","eat":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

Class Name: ParkService

```
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;
        }
    }
}
```

Class Name : ParkLocator

```
 public class ParkLocator {
     public static List<String> country(String country){
         ParkService.ParksImplPort prkSvc = new
 ParkService.ParksImplPort();
         return prkSvc.byCountry(country);
     }
 }
```

Class Name : ParkLocatorTest

```
@isTest
public class ParkLocatorTest {
  @isTest
    static void testCallout(){
        Test.setMock(webServiceMock.class,new ParkServiceMock());
        String country = 'United States';
        List<String> expectedParks = new List<String>{'Yosemite','Sequoia','Crater Lake'};
      System.assertEquals(expectedParks,ParkLocator.country(country));
    }
}
```

Apex Web Services

Class Name : AccountManager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result =  [SELECT Id,Name,(select Id,Name from Contacts) from Account
where Id=:accountId];
        return result;
    }
}
```

Class Name : AccountManagerTest

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =

'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/cont
acts';
        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 accountTest = new Account(
            Name='Test record');
        insert accountTest;

        Contact contactTest = new Contact(
        FirstName='John',
        LastName='Doe',
        AccountId=accountTest.Id);
        insert contactTest;
        return accountTest.Id;
    }
}
```

Apex Specialist

Class Name : CreateDefaultData

```
   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
    @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 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;

    }
}
```

Class Name : CreateDefaultDataTest

```
   @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');

    }
}
```

Class Name : MaintenanceRequestHelper

```
public class MaintenanceRequestHelper {


 public static void updateWorkOrders(Map<Id, Case> applicableCases){

     Map<Id, Integer> mapProduct = new Map<Id, Integer>();
     List<Case> newCases = new List<Case>();
     List<Product2> listProduct = [select Id, Maintenance_Cycle__c from Product2];
     for (Product2 p : listProduct) {
        if (p != null) {
           if(p.Maintenance_Cycle__c != null){
              mapProduct.put(p.Id, Integer.valueOf(p.Maintenance_Cycle__c));
           }
        }
     }
     for(Case a: applicableCases.values()){
        Case newCase = new Case();
        newCase.Vehicle__c = a.Vehicle__c;
        newCase.Equipment__c = a.Equipment__c;
        newCase.Type = 'Routine Maintenance';
        newCase.Subject = String.isBlank(a.Subject) ? 'Routine Maintenance Request' :
a.Subject;
        newCase.Date_Reported__c = Date.today();
        newCase.Status = 'New';
        newCase.Product__c = a.Product__c;
        newCase.AccountId = a.AccountId;
        newCase.ContactId = a.ContactId;
        newCase.AssetId = a.AssetId;
        newCase.Origin = a.Origin;
        newCase.Reason = a.Reason;
        newCase.Date_Due__c =  (mapProduct.get(a.Id) != null) ?
(Date.today().addDays(Integer.valueOf(mapProduct.get(a.Id)))) : (Date.today());
           newCases.add(newCase);
     }
     if(newCases.size() > 0){
            insert newCases;
        }
    }
}
```

Class Name : MaintenanceRequestHelperTest

```
  @isTest
public class MaintenanceRequestHelperTest {
    @isTest
    static void testMaintenanceRequest(){

        List<Case> caseList = new List<Case>();
        Product2 prod = new Product2();
        prod.Cost__c = 50;
        prod.Name = 'Ball Valve 10 cm';
        prod.Lifespan_Months__c = 12;
        prod.Maintenance_Cycle__c = 365;
        prod.Current_Inventory__c = 50;
        prod.Replacement_Part__c = true;
        prod.Warehouse_SKU__c = '100009';
        insert prod;
        System.assertEquals(1, [SELECT count() FROM Product2 WHERE Name = 'Ball
Valve 10 cm']);

        for(Integer i=1;i<=300;i++) {
            Case caseNew = new Case();
            caseNew.Subject = 'Maintenance';
            caseNew.Type = 'Other';
            caseNew.Status = 'New';
            caseNew.Equipment__c = prod.Id;
            caseList.add(caseNew);
        }

        Test.startTest();
        insert caseList;
        System.assertEquals(300, [SELECT count() FROM Case WHERE Type = 'Other']);

        for(Case a : caseList){
           a.Type = 'Repair';
           a.Status = 'Closed';
        }
        update caseList;
        System.assertEquals(300, [SELECT count() FROM Case WHERE Type = 'Repair']);
        Test.stopTest();
    }
}
```

Class Name : WarehouseCalloutService

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

    //class that makes a REST callout to an external warehouse system to get a list of
equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in
Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

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

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

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

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

Class Name : WarehouseCalloutServiceMock

```
 @isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   // implement http mock callout
   global static HttpResponse respond(HttpRequest request){

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

       // Create a fake response
       HttpResponse response = new HttpResponse();
       response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
       response.setStatusCode(200);
       return response;
   }
}
```

Class Name : WarehouseCalloutServiceTest

```
 @isTest

private class WarehouseCalloutServiceTest {
   @isTest
   static void testWareHouseCallout(){
       Test.startTest();
       // implement mock callout test here
       Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
       WarehouseCalloutService.runWarehouseEquipmentSync();
       Test.stopTest();
```

```
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

Class Name : WarehouseSyncSchedule

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

Class Name : WarehouseSyncScheduleTest

```
 @isTest
public class WarehouseSyncScheduleTest {

    @isTest static void testScheduler() {

        Test.SetMock(HttpCallOutMock.class, new WarehouseCalloutServiceMock());
        String CRON_EXP = '0 0 0 1 1/1 ? *';   // To be executed monthly at day one

        Integer runDate = 1;

        DateTime firstRunTime = System.now();
        DateTime nextDateTime;
        if(firstRunTime.day() < runDate) {

            nextDateTime = firstRunTime;

        } else {

            nextDateTime = firstRunTime.addMonths(1);

        }
        Datetime nextRunTime = Datetime.newInstance(nextDateTime.year(),
nextDateTime.month(), runDate);
        Test.startTest();
        WarehouseSyncSchedule warehouseSyncSchedule = new
WarehouseSyncSchedule();
        String jobId = System.schedule('Test Scheduler',
                                CRON_EXP,
                                warehouseSyncSchedule);

        Test.stopTest();
        // Get the information from the CronTrigger API object

        CronTrigger ct = [SELECT Id,  CronExpression,  TimesTriggered,  NextFireTime
FROM CronTrigger WHERE Id = :jobId];
        // Verify the expressions are the same

        System.assertEquals(CRON_EXP, ct.CronExpression);
        // Verify the job has not run

        System.assertEquals(0, ct.TimesTriggered);

        // Verify the next time the job will run

        System.assertEquals(String.valueOf(nextRunTime),
String.valueOf(ct.NextFireTime));
    }
}
```