Apex Triggers

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert, before
update) {
for(Account account : Trigger.New){
if(account.Match_Billing_Address__c == True){
account.ShippingPostalcode =
account.BillingPostalCode;
}
}
}
```

ClosedOpportunityTrigger.apxt

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

Apex Class and Tests

AccountManager

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

```
}
}
```

AccountManagerTest

```
@IsTest
private class AccountManagerTest{
@isTest static void testAccountManager(){
Id recordId = getTestAccountId();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
request.httpMethod = 'GET';
RestContext.request = request;
// Call the method to test
Account acc = AccountManager.getAccount();
// Verify results
System.assert(acc != null);
}
private static Id getTestAccountId(){
Account acc = new Account(Name = 'TestAcc2');
Insert acc;
Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
Insert con;
return acc.Id;
}
}
```

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;
}
}
```

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();
}
}
```

AddPrimaryContact

```
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;
}
}
}
```

AddPrimaryContactTest

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

AnimalLocator

```
public class AnimalLocator {
public class Animal {
public Integer id;public String name;
public String eats;
public String says;
}
public class AnimalResult {
public Animal animal;
}
public static String getAnimalNameById(Integer id) {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
request.setMethod('GET');
HttpResponse response = http.send(request);
AnimalResult result = (AnimalResult) JSON.deserialize(response.getBody(), AnimalResult.class);
return result.animal.name;
}
}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
global HTTPResponse respond(HTTPRequest request) {
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
response.setStatusCode(200);
return response;
}
}
```

AnimalLocatorTest

```
@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);
}
}
```

AsyncParkService

```
//Generated by wsdl2apex
public class AsyncParkService {public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
public String[] getValue() {
```

```
ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
return response.return_x;
}
}
public class AsyncParksImplPort {
public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
public Map<String,String> inputHttpHeaders_x;
public String clientCertName_x;
public Integer timeout_x;
private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation continuation,String arg0) {
ParkService.byCountry request_x = new ParkService.byCountry();
request_x.arg0 = arg0;
return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(
this,
request_x,
AsyncParkService.byCountryResponseFuture.class,
continuation,
new String[]{endpoint_x,
'',
'http://parks.services/',
'byCountry',
'http://parks.services/',
'byCountryResponse',
'ParkService.byCountryResponse'}
);
}
}
```

BotController

```
public with sharing class BotController {
class HandlerMapping {
public String handlerClassName;
public Pattern utterancePattern;
public HandlerMapping(String handlerClassName, String patternStr) {
this.handlerClassName = handlerClassName;
this.utterancePattern = Pattern.compile(patternStr);
}
}
static List<HandlerMapping> handlerMappings;
static {
List<Bot_Command__c> commands = [SELECT apex_class__c, pattern__c FROM Bot_Command__c WHEREActive__c =
True ORDER BY Name];
System.debug(commands);
List<HandlerMapping> mappings = new List<HandlerMapping>();
for (Bot_Command__c command : commands) {
mappings.add(new HandlerMapping(command.apex_class__c, command.pattern__c));
}
handlerMappings = mappings;
}
@AuraEnabled
public static BotResponse submit(String utterance, Map<String, String> session, String fileName, String fileContent) {
```

```
try {
if (session != null) {
String nextCommand = session.get('nextCommand');
if (nextCommand != null) {
Type t = Type.forName('', nextCommand);
BotHandler h = (BotHandler)t.newInstance();
return h.handle(utterance, null, session, fileName, fileContent);
}
}
for (HandlerMapping mapping : BotController.handlerMappings) {
Matcher utteranceMatcher = mapping.utterancePattern.matcher(utterance);
if (utteranceMatcher.matches()) {
Type t = Type.forName('', mapping.handlerClassName);
BotHandler h = (BotHandler)t.newInstance();
List<String> params = new List<String>();
for (Integer i=1; i<=utteranceMatcher.groupCount(); i=i+1) {
params.add(utteranceMatcher.group(i).trim());
}
return h.handle(utterance, params, session, fileName, fileContent);
}
}
return new BotResponse(new BotMessage('Bot', 'I don\'t know how to answer that'));
} catch (Exception e) {
System.debug(e);
return new BotResponse(new BotMessage('Bot', 'Oops, something went wrong invoking that command'));
}
}
}
```

BotField

```
public class BotField {
@AuraEnabled public String name { get;set; }@AuraEnabled public String value { get;set; }
@AuraEnabled public String linkURL { get;set; }
public BotField(String name, String value) {
this.name = name;
this.value = value;
}
public BotField(String name, String value, string linkURL) {
this.name = name;
this.value = value;
this.linkURL = linkURL;
}
}
```

BotHandler

```
public interface BotHandler {
BotResponse handle(String utterance, String[] params, Map<String, String> session, String fileName, String fileContent);
}
```

BotItem

```
public class BotItem {
@AuraEnabled public String name { get;set; }
```

```
@AuraEnabled public String linkURL { get;set; }
public BotItem(String name) {
this.name = name;
}
public BotItem(String name, string linkURL) {
this.name = name;
this.linkURL = linkURL;
}
}
```

BotMessage

```
public virtual class BotMessage {
@AuraEnabled public String author { get;set; }
@AuraEnabled public String messageText { get;set; }
@AuraEnabled public List<BotRecord> records { get;set; }
@AuraEnabled public List<BotItem> items { get;set; }
@AuraEnabled public List<BotMessageButton> buttons { get;set; }@AuraEnabled public String imageURL { get;set; }
public BotMessage() {
}
public BotMessage(String author, String messageText) {
this.author = author;
this.messageText = messageText;
}
public BotMessage(String author, String messageText, List<BotRecord> records) {
this.author = author;
this.messageText = messageText;
this.records = records;
}
public BotMessage(String author, String messageText, List<BotItem> items) {
this.author = author;
this.messageText = messageText;
this.items = items;
}
public BotMessage(String author, String messageText, List<BotMessageButton> buttons) {
this.author = author;
this.messageText = messageText;
this.buttons = buttons;
}
public BotMessage(String author, String messageText, String imageURL) {
this.author = author;
this.messageText = messageText;
this.imageURL = imageURL;
}
}
```

BotMessageButton

```
public class BotMessageButton {
@AuraEnabled public String label { get;set; }
@AuraEnabled public String value { get;set; }
public BotMessageButton(String label, String value) {
this.label = label;
this.value = value;
}
```

}BotRecord

```
public class BotRecord {
@AuraEnabled
public List<BotField> fields { get;set; }
public BotRecord(List<BotField> fields) {
this.fields = fields;
}
}
```

BotResponse

```
public class BotResponse {
@AuraEnabled public List<BotMessage> messages { get; set; }
@AuraEnabled public Map<String, String> session { get; set; }
public BotResponse() {
}
public BotResponse(BotMessage[] messages) {
this.messages = messages;
}
public BotResponse(List<BotMessage> messages, Map<String, String> session) {
this.messages = messages;
this.session = session;
}
/**
* Convenience constructor to create a response with a single message
*/
public BotResponse(BotMessage message) {
this.messages = new BotMessage[]{message};
}
/**
* Convenience constructor to create a response with a single message
*/
public BotResponse(BotMessage message, Map<String, String> session) {
this.messages = new BotMessage[]{message};
this.session = session;
}
}
```

BotTest

```
@isTest
public class BotTest {
static testMethod void testBotController() {
Bot_Command__c bc = new Bot_Command__c(Sample_Utterance__c='help lightning',
apex_class__c='HandlerHelpTopic', pattern__c='help (.*)');
insert bc;
BotResponse response = BotController.submit('help lightning', null, null, null);
Map<String, String> session = response.session;
response = BotController.submit('Developer', session, null, null);
System.assert(response.messages[0].items.size() > 0);
}
static testMethod void testHello() {
BotHandler handler = new HandlerHello();
```

```
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].messageText == 'Hi there!');
}
static testMethod void testAddTwoNumbers() {
BotHandler handler = new HandlerAddTwoNumbers();
BotResponse response = handler.handle('', null, null, null, null);
Map<String, String> session = response.session;
response = handler.handle('1', null, session, null, null);
session = response.session;
response = handler.handle('2', null, session, null, null);
System.assert(response.messages[0].messageText == '1 + 2 = 3');
}
static testMethod void testCostCenter() {
BotHandler handler = new HandlerCostCenter();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].messageText == 'Your cost center is 21852');
}
static testMethod void testEmployeeId() {
BotHandler handler = new HandlerEmployeeId();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].messageText == 'Your employee id is 9854');
}
static testMethod void testFindAccount() {
Account a = new Account(Name='TestAccount');
insert a;
BotHandler handler = new HandlerFindAccount();
BotResponse response = handler.handle('', new String[]{'Test'}, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testFindContact() {
Contact c = new Contact(LastName='TestContact');insert c;
BotHandler handler = new HandlerFindContact();
BotResponse response = handler.handle('', new String[]{'Test'}, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testHelp() {
Bot_Command__c bc = new Bot_Command__c(Sample_Utterance__c='Hello', apex_class__c='HelloHandler',
pattern__c='Hello');
insert bc;
BotHandler handler = new HandlerHelp();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].items.size() == 1);
}
static testMethod void testHelpTopic() {
BotHandler handler = new HandlerHelpTopic();
BotResponse response = handler.handle('', null, null, null, null);
Map<String, String> session = response.session;
handler.handle('User', null, session, null, null);
response = handler.handle('', null, null, null, null);
session = response.session;
response = handler.handle('Admin', null, session, null, null);
response = handler.handle('', null, null, null, null);
session = response.session;
response = handler.handle('Developer', null, session, null, null);
```

```
System.assert(response.messages[0].items.size() > 0);
}
static testMethod void testMyOpenCases() {
Case c = new Case(Subject='TestCase');
insert c;
BotHandler handler = new HandlerMyOpenCases();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testTopOpportunities() {
Account a = new Account(Name='TestAccount');
insert a;
Opportunity o = new Opportunity(Name='TestOpportunity', AccountId=a.id, StageName='Prospecting',
CloseDate=System.today().addMonths(1));
insert o;
BotHandler handler = new HandlerTopOpportunities();
BotResponse response = handler.handle('', new String[]{'3'}, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testTravelApproval() {
BotHandler handler = new HandlerTravelApproval();
BotResponse response = handler.handle('', null, null, null, null);Map<String, String> session = response.session;
handler.handle('Boston', null, session, null, null);
handler.handle('Customer Facing', null, session, null, null);
handler.handle('02/23/2017', null, session, null, null);
handler.handle('1000', null, session, null, null);
handler.handle('1000', null, session, null, null);
System.assert(response.messages[0].messageText.length() > 0);
}
static testMethod void testPipeline() {
BotHandler handler = new HandlerPipeline();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].imageURL != null);
}
static testMethod void testQuarter() {
BotHandler handler = new HandlerQuarter();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages[0].imageURL != null);
}
static testMethod void testNext() {
Account a = new Account(Name='TestAccount');
insert a;
Opportunity o = new Opportunity(Name='TestOpportunity', AccountId=a.id, StageName='Prospecting',
CloseDate=System.today().addMonths(1));
insert o;
Case c = new Case(Subject='TestCase', Priority='High');
insert c;
BotHandler handler = new HandlerNext();
BotResponse response = handler.handle('', null, null, null, null);
System.assert(response.messages.size() > 1);
}
static testMethod void testSOQL() {
Account a = new Account(Name='TestAccount');
insert a;
```

```
BotHandler handler = new HandlerSOQL();
BotResponse response = handler.handle('select id from account', null, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testFindPropertiesByBedrooms() {
Property__c p = new Property__c(Name='TestProperty', Beds__c=3, City__c='Boston');
insert p;
BotHandler handler = new HandlerFindPropertiesByBedrooms();
BotResponse response = handler.handle('', new String[]{'3', 'Boston'}, null, null, null);
System.assert(response.messages[0].records.size() == 1);
}
static testMethod void testFindProperties() {
Property__c p = new Property__c(Name='TestProperty', Price__c=450000, City__c='Boston');
insert p;
BotHandler handler = new HandlerFindProperties();Map<String, String> session = handler.handle('', null, null, null,
null).session;
session = handler.handle('Boston', null, session, null, null).session;
session = handler.handle('Single Family', null, session, null, null).session;
session = handler.handle('400000', null, session, null, null).session;
BotResponse response = handler.handle('500000', null, session, null, null);
System.assert(response.messages[0].records.size() == 1);
}
}
```

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
global void execute(SchedulableContext sc){
List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
List<Lead> lstOfUpdatedLead = new List<Lead>();
if(!lstOfLead.isEmpty()){
for(Lead ld : lstOfLead){
ld.LeadSource = 'Dreamforce';
lstOfUpdatedLead.add(ld);
}
UPDATE lstOfUpdatedLead;
}
}
}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
@testSetup
static void setup(){
List<Lead> lstOfLead = new List<Lead>();
for(Integer i =1;i <=200;i++){
Lead ld = new Lead(Company = 'Comp' + i, LastName = 'LN' + i, Status = 'Working - Contacted');
lstOfLead.add(ld);
}
Insert lstOfLead;
}
static testmethod void testDailyLeadProcessorScheduledJob(){
String sch = '0 5 12 * * ?';
```

```
Test.startTest();
String jobId = System.Schedule('ScheduledApexText', sch, new DailyLeadProcessor());
List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
system.assertEquals(200, lstOfLead.size());
Test.stopTest();
}}
```

Apex Trigger

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account (before insert, before update) {
For(Account accountAddress: Trigger.new){
if(accountAddress.BillingPostalCode !=null && accountAddress.Match_Billing_Address__c ==true){
accountAddress.ShippingPostalCode=accountAddress.BillingPostalCode;
}
}
}
```

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;
}
}
```

PushNotificationTrigger

```
trigger PushNotificationTrigger on Property__c (after update) {
/*
for (Property__c property : Trigger.New) {
if (property.Price__c != Trigger.oldMap.get(property.Id).Price__c) {
Messaging.PushNotification msg = new Messaging.PushNotification();
String text = property.Name + '. New Price: $' + property.Price__c.setScale(0).format();
Map<String, Object> payload = Messaging.PushNotificationPayload.apple(text, '', null, null);
msg.setPayload(payload);
Set<String> users = new Set<String>();
users.add(UserInfo.getUserId());
msg.send('DreamHouzz', users);
}
}*/
}
```

RejectDuplicateFavorite

```
trigger RejectDuplicateFavorite on Favorite__c (before insert) {
// NOTE: this trigger needs to be bulkified
Favorite__c favorite = Trigger.New[0];
List<Favorite__c> dupes = [Select Id FROM Favorite__C WHERE Property__c = :favorite.Property__c AND User__c =
:favorite.User__c];
```

```
if (!dupes.isEmpty()) {
favorite.addError('duplicate');
}
}
```

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');
}
}
}
```

APEX SPECIALIST SUPERBADGE

MaintenanceRequest

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

MaintainenceRequestHelper.cls

```
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);
}
}
}
//When an existing maintenance request of type Repair or Routine Maintenance is closed,
//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
Map<Id,Case> closedCases = 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>();
//calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.
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'));
List<Case> newCases = new List<Case>();
for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
//define the due date by applying the shortest maintenance cycle to today’s date.
//If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
//} else {
// nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
//}
newCases.add(nc);
}
insert newCases;
List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c item = clonedListItem.clone();
item.Maintenance_Request__c = nc.Id;
clonedList.add(item);
}
}
insert clonedList;
}
}
```

```
}
```

MaintainenceRequestHelperTest.cls

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }
    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }
    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }
    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }
    @isTest
```

```
    private static void testPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        Product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;        case createdCase =
createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;
        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;
        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();
        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
                        Vehicle__c,
                        Date_Due__c
                       from case
                       where status ='New'];
        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);
        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }
```

```
    @isTest    private static void testNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;
        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;
        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;
        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();
        list<case> allCase = [select id from case];
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c = :createdCase.Id];
        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }
    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();       list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();
        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;
        for(integer i = 0; i < 300; i++){
```

```
        caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
      }
      insert caseList;
      for(integer i = 0; i < 300; i++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentLi
st.get(i).id, caseList.get(i).id));
      }
      insert equipmentMaintenanceItemList;
      test.startTest();
      for(case cs : caseList){
        cs.Status = 'Closed';
        oldCaseIds.add(cs.Id);
      }
      update caseList;
      test.stopTest();
      list<case> newCase = [select id
                            from case
                            where status ='New'];       list<Equipment_Maintenance_Item__c>
workParts = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c in: oldCaseIds];
      system.assert(newCase.size() == 300);
      list<case> allCase = [select id from case];
      system.assert(allCase.size() == 600);
    }
}
```

WarehouseCalloutService.cls

```
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();
  }
```

}

WarehouseCalloutServiceMock.cls

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantit
y":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d662
26726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726
b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseSyncSchedule.cls

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

WarehouseSyncScheduleTest.cls

```
    @isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
```

```
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');       Test.stopTest();
    }
}
```

WarehouseCalloutServiceTest.cls

```
@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);
    }
}
```

WarehouseCalloutServiceMock.cls

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generat
or 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611100aaf7
42","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf743","re
placement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');       response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutService.cls

```
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();
    }
}
```