**Apex Triggers**

**AccountAddressTrigger.apxt**

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

for(Account a: Trigger.New){

if(a.Match\_Billing\_Address\_\_c == true && a.BillingPostalCode!= null){

a.ShippingPostalCode=a.BillingPostalCode;

}

}

}

**ClosedOpportunityTrigger.apxt**

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

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

for(Opportunity opp : Trigger.new) {

//Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on Create

if(Trigger.isInsert) {

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

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

}

}

//Only create Follow Up Task only once when Opp StageName changed to 'Closed Won' on Update

if(Trigger.isUpdate) {

if(Opp.StageName == 'Closed Won'

&& Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {

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

}

}

}

if(taskList.size()>0) {

insert taskList;

}

}

**RestrictContactByName.apxt**

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

//check contacts prior to insert or update for invalid data

For (Contact c : Trigger.New) {

if(c.LastName == 'INVALIDNAME') { //invalidname is invalid

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

**MaintenanceRequest.apxt**

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

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

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

}

}

**Apex Classes**

**DailyLeadProcessor.apxc**

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;

}

}

}

**AnimalLocatorMock.apxc**

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

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

response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');

response.setStatusCode(200);

return response;

}

}

**LeadProcessorTest.apxc**

@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();

}

}

**LeadProcessor.apxc**

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){

}

}

**AddPrimaryContact.apxc**

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;

}

}

**ParkService.apxc**

//Generated by wsdl2apex

public class ParkService {

public class byCountryResponse {

public String[] return\_x;

private String[] return\_x\_type\_info = new

String[]{'return','http://parks.services/',null,'0','-1','false'};

private String[] apex\_schema\_type\_info = new

String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'return\_x'};

}

public class byCountry {

public String arg0;

private String[] arg0\_type\_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};

private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'arg0'};

}

public class ParksImplPort {

public String endpoint\_x = 'https://th-apex-soap-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;

}

}

}

**AsyncParkService.apxc**

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'}

);

}

}

}

**AccountManagerTest.apxc**

@isTest

private class AccountManagerTest {

private static testMethod void getAccountTest1() {

Id recordId = createTestRecord();

// Set up a test request

RestRequest request = new RestRequest();

request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ 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;

}

}

**ParkLocator.apxc**

public class ParkLocator {

public static string[] country(string theCountry) {

ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space

return parkSvc.byCountry(theCountry);

}

}

**ParkServiceMock.apxc**

@isTest

global class ParkServiceMock implements WebServiceMock {

global void doInvoke(

Object stub,

Object request,

Map<String, Object> response,

String endpoint,

String soapAction,

String requestName,

String responseNS,

String responseName,

String responseType) {

// start - specify the response you want to send

ParkService.byCountryResponse response\_x = new

ParkService.byCountryResponse();

response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

// end

response.put('response\_x', response\_x);

}

}

**AccountManager.apxc**

@RestResource(urlMapping='/Accounts/\*/contacts')

global class AccountManager {

@HttpGet

global static Account getAccount() {

RestRequest req = RestContext.request;

String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

FROM Account WHERE Id = :accId];

return acc;

}

}

**AnimalLocatorTest.apxc**

@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 );

}

}

**DailyLeadProcessorTest.apxc**

@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());

}

}

**AnimalLocator.apxc**

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');

}

}

**AddPrimaryContactTest.apxc**

@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();

}

}

**AccountProcessorTest.apxc**

@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();

}

}

**AccountProcessor.apxc**

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;

}

}

**TestVerifyDate.apxc**

@isTest

public class TestVerifyDate

{

static testMethod void testMethod1()

{

Date d = VerifyDate.CheckDates(System.today(),System.today()+1);

Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);

}

}

**VerifyDate.apxc**

public class VerifyDate {

//method to handle potential checks against two dates

public static Date CheckDates(Date date1, Date date2) {

//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month

if(DateWithin30Days(date1,date2)) {

return date2;

} else {

return SetEndOfMonthDate(date1);

}

}

//method to check if date2 is within the next 30 days of date1

private static Boolean DateWithin30Days(Date date1, Date date2) {

//check for date2 being in the past

if( date2 < date1) { return false; }

//check that date2 is within (>=) 30 days of date1

Date date30Days = date1.addDays(30); //create a date 30 days away from date1

if( date2 >= date30Days ) { return false; }

else { return true; }

}

//method to return the end of the month of a given date

private static Date SetEndOfMonthDate(Date date1) {

Integer totalDays = Date.daysInMonth(date1.year(), date1.month());

Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);

return lastDay;

}

}

**CreateDefaultData.apxc**

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;

}

}

**CreateDefaultDataTest.apxc**

@isTest

private class CreateDefaultDataTest {

@isTest

static void createData\_test(){

Test.startTest();

CreateDefaultData.createDefaultData();

List<Vehicle\_\_c> vehicles = [SELECT Id FROM Vehicle\_\_c];

List<Product2> equipment = [SELECT Id FROM Product2];

List<Case> maintenanceRequest = [SELECT Id FROM Case];

List<Equipment\_Maintenance\_Item\_\_c> joinRecords = [SELECT Id FROM Equipment\_Maintenance\_Item\_\_c];

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

}

}

**MaintenanceRequestHelper.apxc**

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

Set<Id> validIds = new Set<Id>();

For (Case c : updWorkOrders){

if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

validIds.add(c.Id);

}

}

}

//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;

}

}

}

**MaintenanceRequestHelperTest.apxc**

@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(equipmentList.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.apxc**

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');

}

}

**WarehouseCalloutServiceMock.apxc**

@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":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

response.setStatusCode(200);

return response;

}

}

**WarehouseCalloutServiceTest.apxc**

@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);

}

}

**WarehouseSyncSchedule.apxc**

global with sharing class WarehouseSyncSchedule implements Schedulable{

global void execute(SchedulableContext ctx){

System.enqueueJob(new WarehouseCalloutService());

}

}

**WarehouseSyncScheduleTest**

@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();

}

}