

APEX TESTING:

RandomContactFactory.apxc

```
public class RandomContactFactory {  
  
    public static List<Contact> generateRandomContacts(Integer num, String lastName){  
        List<Contact> contactList = new List<Contact>();  
        for(Integer i = 1; i <= num; i++){  
            Contact ct = new Contact(FirstName = 'Test '+i, LastName = lastname);  
            contactList.add(ct);  
        }  
        return contactList;  
    }  
}
```

RestrictContactByName.aptx

```
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');  
        }  
    }  
}
```

TestRestrictContactByName.apxc

```
@isTest  
public class TestRestrictContactByName {  
    @isTest static void testContact(){  
        Contact ct = new Contact();  
        ct.LastName = 'INVALIDNAME';  
        Database.SaveResult res = Database.insert(ct, false);  
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',  
res.getErrors()[0].getMessage());  
    }  
}
```

TestVerifyDate.apxc

@isTest

```
public class TestVerifyDate {  
    @isTest static void Test_CheckDates_case1(){  
        Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('01/03/2020'));  
        System.assertEquals(Date.parse('01/03/2020'), d);  
    }  
    @isTest static void Test_CheckDates_case2(){  
        Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('03/03/2020'));  
        System.assertEquals(Date.parse('01/31/2020'), d);  
    }  
}
```

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;  
    }  
}
```

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 op: Trigger.New){
        if(op.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = op.Id));
        }
    }
    if(tasklist.size() > 0){
        insert tasklist;
    }
}
```

ASYNCHRONOUS APEX:

AccountProcessor.apxc

```
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id from Contacts) from Account where Id
in :accountIds];

        for(Account acc: accList){
            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }

        update accList;
    }
}
```

AccountProcessorTest.apxc

@isTest

```
public class AccountProcessorTest {
```

```
    public static testmethod void testAccountProcessor(){
```

```
        Account a = new Account();
```

```
        a.Name = 'Test Account';
```

```
        insert a;
```

```
        Contact con = new Contact();
```

```
        con.FirstName = 'Omkar';
```

```
        con.LastName = 'Raghatwan';
```

```
        con.AccountId = a.Id;
```

```
        insert con;
```

```
        List<Id> accListId = new List<Id>();
```

```
        accListId.add(a.Id);
```

```
        Test.startTest();
```

```
        AccountProcessor.countContacts(accListId);
```

```
        Test.stopTest();
```

```
        Account acc = [Select Number_Of_Contacts__c from Account where Id =: a.Id];
```

```
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c), 1);
```

```
    }
```

```
}
```

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {
```

```
    public Contact c;
```

```
    public String state;
```

```
    public AddPrimaryContact(Contact c, String state) {
```

```
        this.c = c;
```

```
        this.state = state;
```

```
    }
```

```
    public void execute(QueueableContext qc) {
```

```
        system.debug('this.c = '+this.c+' this.state = '+this.state);
```

```
        List<Account> acc_lst = new List<account>([select id, name, BillingState from account where  
account.BillingState = :this.state limit 200]);
```

```
        List<contact> c_lst = new List<contact>();
```

```
        for(account a: acc_lst) {
```

```
            contact c = new contact();
```

```

        c = this.c.clone(false, false, false, false);
        c.AccountId = a.Id;
        c_lst.add(c);
    }
    insert c_lst;
}

}

```

AddPrimaryContactTest.apxc

@IsTest

public class AddPrimaryContactTest {

@IsTest

```

public static void testing() {
    List<account> acc_lst = new List<account>();
    for (Integer i=0; i<50;i++) {
        account a = new account(name=string.valueOf(i),billingstate='NY');
        system.debug('account a = '+a);
        acc_lst.add(a);
    }
    for (Integer i=0; i<50;i++) {
        account a = new account(name=string.valueOf(50+i),billingstate='CA');
        system.debug('account a = '+a);
        acc_lst.add(a);
    }
    insert acc_lst;
    Test.startTest();
    contact c = new contact(lastname='alex');
    AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
    system.debug('apc = '+apc);
    System.enqueueJob(apc);
    Test.stopTest();
    List<contact> c_lst = new List<contact>([select id from contact]);
    Integer size = c_lst.size();
    system.assertEquals(50, size);
}

}

```

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {
```

```
    global void execute(SchedulableContext ctx) {
```

```
        //Retrieving the 200 first leads where lead source is in blank.
```

```
        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = " LIMIT 200];
```

```
        //Setting the LeadSource field the 'Dreamforce' value.
```

```
        for (Lead lead : leads) {
```

```
            lead.LeadSource = 'Dreamforce';
```

```
        }
```

```
        //Updating all elements in the list.
```

```
        update leads;
```

```
    }
```

```
}
```

DailyLeadProcessorTest.apxc

```
@isTest
```

```
private class DailyLeadProcessorTest {
```

```
    @isTest
```

```
    public static void testDailyLeadProcessor(){
```

```
        //Creating new 200 Leads and inserting them.
```

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

```
        for (Integer x = 0; x < 200; x++) {
```

```
            leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
```

```
        }
```

```
        insert leads;
```

```
        //Starting test. Putting in the schedule and running the DailyLeadProcessor execute method.
```

```
        Test.startTest();
```

```
        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());
```

```
        Test.stopTest();
```

```
        //Once the job has finished, retrieve all modified leads.
```

```
        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT 200];
```

```
        //Checking if the modified leads are the same size number that we created in the start of this method.
```

```
        System.assertEquals(200, listResult.size());
```

```
    }
```

```
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0; i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

LeadProcessor.apxc

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

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

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

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }

    global void finish(Database.BatchableContext bc){
        System.debug('count = '+count);
    }
}
```

APEX INTEGRATION SERVICE:

AccountManager.apxc

```
@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.apxc

```
@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;
    }
}
```

AnimalLocator.apxc

```
public class AnimalLocator
```



```

{

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);
    String strResp = "";
    system.debug('*****response '+response.getStatusCode());
    system.debug('*****response '+response.getBody());
    // If the request is successful, parse the JSON response.
    if (response.getStatusCode() == 200)
    {
        // Deserializes the JSON string into collections of primitive data types.
        Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
        // Cast the values in the 'animals' key as a list
        Map<string,object> animals = (map<string,object>) results.get('animal');
        System.debug('Received the following animals:' + animals );
        strResp = string.valueOf(animals.get('name'));
        System.debug('strResp >>>>>' + strResp );
    }
    return strResp ;
}

}

```

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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.getStatusCode(200);
        return response;
    }
}

```

AnimalLocatorTest.apxc

@isTest

```

public class AnimalLocatorTest {
    @isTest public static void AnimalLocatorMock() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(1);
        system.debug(result);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}

```

ParkLocator.apxc

```

public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}

```

ParkLocatorTest.apxc

```

@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}

```

ParkService.apxc

```

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;
    }
}

```

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();

List<String> myStrings = new List<String> {'Park1','Park2','Park3'};

response_x.return_x = myStrings;
// end
response.put('response_x', response_x);
}
}

```

ParksServices.apxc

```

public class parksServices {
    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/', 'parksServices'};
        public String[] byCountry(String arg0) {
            parksServices.byCountry request_x = new parksServices.byCountry();
            request_x.arg0 = arg0;
            parksServices.byCountryResponse response_x;
            Map<String, parksServices.byCountryResponse> response_map_x = new Map<String,
parksServices.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',
    'parksServices.byCountryResponse'}
);
response_x = response_map_x.get('response_x');
return response_x.return_x;
}
}
}

```

APEX SPECIALIST SUPERBADGE:

2

MaintenanceRequest.aptx

```

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

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

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

```

MaintenanceRequestHelper.apxc

```

public with sharing class MaintenanceRequestHelper {

    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

```

```
validIds.add(c.Id);
```

```
    }  
  }  
}
```

```
if (!validIds.isEmpty()){  
    List<Case> newCases = new List<Case>();  
    Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,  
Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM  
Equipment_Maintenance_Items__r)  
FROM Case WHERE Id IN :validIds]);  
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();  
    AggregateResult[] results = [SELECT Maintenance_Request__c,  
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c WHERE  
Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];  

```

```
for (AggregateResult ar : results){  
    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));  
}
```

```
for(Case cc : closedCasesM.values()){  
    Case nc = new Case (  
        ParentId = cc.Id,  
        Status = 'New',  
        Subject = 'Routine Maintenance',  
        Type = 'Routine Maintenance',  
        Vehicle__c = cc.Vehicle__c,  
        Equipment__c =cc.Equipment__c,  
        Origin = 'Web',  
        Date_Reported__c = Date.Today()  
  
    );
```

```
    If (maintenanceCycles.containsKey(cc.Id)){  
        nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));  
    } 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> clonedWPs = new List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);
            }
        }
        insert ClonedWPs;
    }
}
}

```

3

WarehouseCalloutService.apxc

```

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();
}

}

```

4

WarehouseSyncSchedule.apxc

```

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

```

5

MaintenanceRequest.aptx

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

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {  
    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {  
        Set<Id> validIds = new Set<Id>();  
  
        For (Case c : updWorkOrders){  
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){  
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){  
                    validIds.add(c.Id);  
                }  
            }  
        }  
  
        if (!validIds.isEmpty()){  
            List<Case> newCases = new List<Case>();  
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,  
Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM  
Equipment_Maintenance_Items__r)  
FROM Case WHERE Id IN :validIds]);  
            Map<Id,Decimal> maintenanceCycles = new Map<Id,Decimal>();  
            AggregateResult[] results = [SELECT Maintenance_Request__c,  
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c WHERE  
Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];  
  
            for (AggregateResult ar : results){  
                maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));  
            }  
  
            for (Case cc : closedCasesM.values()){  
                Case nc = new Case (  
                    ParentId = cc.Id,  
                    Status = 'New',  
                    Subject = 'Routine Maintenance',  
                    Type = 'Routine Maintenance',  
                    Vehicle__c = cc.Vehicle__c,  
                    Equipment__c =cc.Equipment__c,
```

```

        Origin = 'Web',
        Date_Reported__c = Date.Today()

    );

    If (maintenanceCycles.containskey(cc.Id)){
        nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
    }

    newCases.add(nc);
}

insert newCases;

List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
        Equipment_Maintenance_Item__c wpClone = wp.clone();
        wpClone.Maintenance_Request__c = nc.Id;
        ClonedWPs.add(wpClone);

    }
}
insert ClonedWPs;
}
}
}
}

```

MaintenanceRequestHelperTest.apxc

@istest

public with sharing class MaintenanceRequestHelperTest {

```

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

```

```

PRIVATE STATIC Vehicle__c createVehicle(){
    Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
    return Vehicle;
}

```

```

PRIVATE STATIC Product2 createEq(){
    product2 equipment = new product2(name = 'SuperEquipment',
        lifespan_months__C = 10,
        maintenance_cycle__C = 10,
        replacement_part__c = true);
    return equipment;
}

PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cs = new case(Type=REPAIR,
        Status=STATUS_NEW,
        Origin=REQUEST_ORIGIN,
        Subject=REQUEST_SUBJECT,
        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cs;
}

PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
    Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
        Maintenance_Request__c = requestId);
    return wp;
}

@istest
private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c, Date_Due__c

```

```
from case
where status =:STATUS_NEW];
```

```
Equipment_Maintenance_Item__c workPart = [select id
                                             from Equipment_Maintenance_Item__c
                                             where Maintenance_Request__c =:newReq.Id];
```

```
system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
```

```
}
```

```
@istest
```

```
private static void testMaintenanceRequestNegative(){
```

```
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
```

```
    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;
```

```
    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;
```

```
    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;
```

```
    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();
```

```
list<case> allRequest = [select id
                          from case];
```

```
Equipment_Maintenance_Item__c workPart = [select id
                                             from Equipment_Maintenance_Item__c
                                             where Maintenance_Request__c = :emptyReq.Id];
```

```
system.assert(workPart != null);
system.assert(allRequest.size() == 1);
```

```
}
```

@istest

```
private static void testMaintenanceRequestBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new list<Equipment_Maintenance_Item__c>();
    list<case> requestList = new list<case>();
    list<id> oldRequestIds = new list<id>();

    for(integer i = 0; i < 300; i++){
        vehicleList.add(createVehicle());
        equipmentList.add(createEq());
    }
    insert vehicleList;
    insert equipmentList;

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

    for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
    }
    insert workPartList;
    test.startTest();
    for(case req : requestList){
        req.Status = CLOSED;
        oldRequestIds.add(req.Id);
    }
    update requestList;
    test.stopTest();

    list<case> allRequests = [select id
                             from case
                             where status =: STATUS_NEW];

    list<Equipment_Maintenance_Item__c> workParts = [select id
                                                       from Equipment_Maintenance_Item__c
                                                       where Maintenance_Request__c in: oldRequestIds];

    system.assert(allRequests.size() == 300);
}
}
```

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);
}
}

```

7

WarehouseSyncSchedule.apxc

```

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

```

WarehouseSyncScheduleTest.apxc

```

@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();
    }
}

```
