//This project doc contains the apex codes used in apex modules and apex specialist super badge.

**AccountAddressTrigger:**

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

**AccountManager:**

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

        return a cc;
    }
}
```

**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  a cc = AccountManager.getAccount();
```

```
 // Verify results
      System.assert(a cc != null);


   private static Id getTestAccountId(){
      Account a cc = new Account(Name = 'TestAcc2');
      Insert a cc;

      Contact con = new Contact(LastName = 'TestCont2', AccountId = a cc.Id);
      Insert con;

      return a cc.Id;
   }
}
```

**AccountProcessor:**

```
public class AccountProcessor {

   @future
   public static void countContacts(List<Id> accountId_last) {

      Map<Id,Integer> account_c no = new Map<Id,Integer>();
      List<account> account_last_all = new List<account>([select id, (select id from contacts) from account]);
      for(account a:account_last_all) {
         account_c no.put(a.id,a.contacts.size()); //populate the map

      }

      List<account> account_last = new List<account>(); // list of account that we will upset

      for(Id accountId : accountId_last) {
         if(account_c no.containsKey(accountId)) {
            account a cc = new account();
            a cc.Id = accountId;
            a cc.Number_of_Contacts__c = account_c no.get(accountId);
            account_last.add(a cc);
         }
```

```
        }
        upset account_last;
    }

}
```

**AccountProcessorTest:**

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void testFunc() {
        account a cc = new account();
        a cc.name = 'MATW INC';
        insert a cc;

        contact con = new contact();
        con.last name = 'Mann1';
        con.AccountId = a cc.Id;
        insert con;
        contact con1 = new contact();
        con1.last name = 'Mann2';
        con1.AccountId = a cc.Id;
        insert con1;


        List<Id> a cc_list = new List<Id>();
        a cc_list.add(a cc.Id);
        Test.startTest();
            AccountProcessor.countContacts(a cc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from account where id = :a cc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }

}
```

**AddPrimaryContact:**

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext q){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account a cc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = a cc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

**AddPrimaryContactTest:**

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
```

```
        INSERT lstOfAcc;
    }

    static test method void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();

        System.assertEquals(50, [select count() from Contact]);
    }
}
```

**AnimalLocator:**

```
public class AnimalLocator {

    public static String getAnimalNameById (Integer id) {
        String AnimalName = '';
        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);
        if (response.getStatusCode() == 200) {
            Map<String,Object> results = (Map<String,Object>) JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String, Object>) results.get('animal');
            animalName = (String) animal.get('name');
        }
        return animalName;
    }
}
```

**AnimalLocatorMock:**

```
@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":7,"name":"panda","eats":"bamboo","says":"i know king f"}}');
        response.setStatusCode(200);
        return response;
    }

}
```

**AnimalLocatorTest:**

```
@isTest
private class AnimalLocatorTest {
@isTest static  void testGet() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        // Call method to test
        String result = AnimalLocator.getAnimalNameById (7);
        // Verify mock response is not null
        System.assertNotEquals(null,result,
                               'The call out returned a null response.');
        System.assertEquals('panda', result,
                            'The animal name should be \'panda\');
    }
}
```

**AsyncParkService:**

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

**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;
    }
```

```
}
```
**ContactsTodayController:**
```
public class ContactsTodayController {

    @AuraEnabled
    public static List<Contact> getContactsForToday() {

        List<Task> my_tasks = [SELECT Id, Subject, WhoId FROM Task WHERE OwnerId = :UserInfo.getUserId() AND IsClosed
= false AND WhoId != null];
        List<Event> my_events = [SELECT Id, Subject, WhoId FROM Event WHERE OwnerId = :UserInfo.getUserId() AND
StartDateTime >= :Date.today() AND WhoId != null];
        List<Case> my_cases = [SELECT ID, ContactId, Status, Subject FROM Case WHERE OwnerId = :UserInfo.getUserId()
AND IsClosed = false AND ContactId != null];

        Set<Id> contactIds = new Set<Id>();
        for(Task tsk : my_tasks) {
            contactIds.add(tsk.WhoId);
        }
        for(Event evt : my_events) {
            contactIds.add(evt.WhoId);
        }
        for(Case cse : my_cases) {
            contactIds.add(cse.ContactId);
        }

        List<Contact> contacts = [SELECT Id, Name, Phone, Description FROM Contact WHERE Id IN :contactIds];

        for(Contact c : contacts) {
            c.Description = '';
            for(Task tsk : my_tasks) {
                if(tsk.WhoId == c.Id) {
                    c.Description += 'Because of Task "'+tsk.Subject+'"\n';
                }
            }
            for(Event evt : my_events) {
                if(evt.WhoId == c.Id) {
```

```
                c.Description += 'Because of Event "'+evt.Subject+'"\n';
            }
        }
        for(Case cse : my_cases) {
            if(cse.ContactId == c.Id) {
                c.Description += 'Because of Case "'+cse.Subject+'"\n';
            }
        }
    }

    return contacts;
  }

}
```

**ContactsTodayControllerTest:**

```
@IsTest
public class ContactsTodayControllerTest {

  @IsTest
  public static void testGetContactsForToday() {

    Account acct = new Account(
      Name = 'Test Account'
    );
    insert acct;

    Contact c = new Contact(
      AccountId = acct.Id,
      FirstName = 'Test',
      LastName = 'Contact'
    );
    insert c;

    Task tsk = new Task(
      Subject = 'Test Task',
      WhoId = c.Id,
```

```
        Status = 'Not Started'
    );
    insert tsk;

    Event evt = new Event(
        Subject = 'Test Event',
        WhoId = c.Id,
        StartDateTime = Date.today().addDays(5),
        EndDateTime = Date.today().addDays(6)
    );
    insert evt;

    Case cse = new Case(
        Subject = 'Test Case',
        ContactId = c.Id
    );
    insert cse;

    List<Contact> contacts = ContactsTodayController.getContactsForToday();
    System.assertEquals(1, contacts.size());
    System.assert(contacts[0].Description.containsIgnoreCase(tsk.Subject));
    System.assert(contacts[0].Description.containsIgnoreCase(evt.Subject));
    System.assert(contacts[0].Description.containsIgnoreCase(cse.Subject));

}

@IsTest
public static void testGetNoContactsForToday() {

    Account acct = new Account(
        Name = 'Test Account'
    );
    insert acct;

    Contact c = new Contact(
        AccountId = acct.Id,
```

```
            FirstName = 'Test',
            LastName = 'Contact'
        );
        insert c;

        Task tsk = new Task(
            Subject = 'Test Task',
            WhoId = c.Id,
            Status = 'Completed'
        );
        insert tsk;

        Event evt = new Event(
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(-6),
            EndDateTime = Date.today().addDays(-5)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id,
            Status = 'Closed'
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(0, contacts.size());

    }

}
```

**CreateDefaultData:**

```
public with sharing class CreateDefaultData{
  Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
```

```
    //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c      customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords = createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }


    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings__c      customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV', Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1,
Model__c = 'Toy Hauler RV'));
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV', Air_Conditioner__c = true, Bathrooms__c = 2, Bedrooms__c =
2, Model__c = 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper', Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
1, Model__c = 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper', Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
1, Model__c = 'Pop-Up Camper'));
        insert vehicles;
```

```
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c = '55d66226726b611100aaf741',name = 'Generator 1000 kW',
Replacement_Part__c = true,Cost__c = 100 ,Maintenance_Cycle__c = 100));
        equipments.add(new Product2(name = 'Fuse 20B',Replacement_Part__c = true,Cost__c = 1000,
Maintenance_Cycle__c = 30  ));
        equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part__c = true,Cost__c = 100  ,
Maintenance_Cycle__c = 15));
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part__c = true,Cost__c = 200  ,
Maintenance_Cycle__c = 60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(1).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(2).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c> createJoinRecords(List<Product2> equipment, List<Case>
maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new List<Equipment_Maintenance_Item__c>();
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
```

```
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

**CreateDefaultDataTest:**

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');
        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c     customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;
```

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

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

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

    }
}
```

**DailyLeadProcessor:**

```
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:**

```
@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());

    }
}
```

**GeocodingService:**

```
public with sharing class GeocodingService {
    private static final String BASE_URL = 'https://nominatim.openstreetmap.org/search?format=json';

    @InvocableMethod(callout=true label='Geocode address')
    public static List<Coordinates> geocodeAddresses(
        List<GeocodingAddress> addresses
    ) {
        List<Coordinates> computedCoordinates = new List<Coordinates>();

        for (GeocodingAddress address : addresses) {
            String geocodingUrl = BASE_URL;
            geocodingUrl += (String.isNotBlank(address.street))
                ? '&street=' + address.street
                : '';
```

```
    geocodingUrl += (String.isNotBlank(address.city))
        ? '&city=' + address.city
        : '';
    geocodingUrl += (String.isNotBlank(address.state))
        ? '&state=' + address.state
        : '';
    geocodingUrl += (String.isNotBlank(address.country))
        ? '&country=' + address.country
        : '';
    geocodingUrl += (String.isNotBlank(address.postalcode))
        ? '&postalcode=' + address.postalcode
        : '';

    Coordinates coords = new Coordinates();
    if (geocodingUrl != BASE_URL) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(geocodingUrl);
        request.setMethod('GET');
        request.setHeader(
            'http-referer',
            URL.getSalesforceBaseUrl().toExternalForm()
        );
        HttpResponse response = http.send(request);
        if (response.getStatusCode() == 200) {
            List<Coordinates> deserializedCoords = (List<Coordinates>) JSON.deserialize(
                response.getBody(),
                List<Coordinates>.class
            );
            coords = deserializedCoords[0];
        }
    }

    computedCoordinates.add(coords);
}
return computedCoordinates;
```

```
    }

    public class GeocodingAddress {
        @InvocableVariable
        public String street;
        @InvocableVariable
        public String city;
        @InvocableVariable
        public String state;
        @InvocableVariable
        public String country;
        @InvocableVariable
        public String postalcode;
    }

    public class Coordinates {
        @InvocableVariable
        public Decimal lat;
        @InvocableVariable
        public Decimal lon;
    }
}
```

**GeocodingServiceTest:**

```
@isTest
private with sharing class GeocodingServiceTest {
    private static final String STREET = 'Camino del Jueves 26';
    private static final String CITY = 'Armilla';
    private static final String POSTAL_CODE = '18100';
    private static final String STATE = 'Granada';
    private static final String COUNTRY = 'Spain';
    private static final Decimal LATITUDE = 3.123;
    private static final Decimal LONGITUDE = 31.333;

    @isTest
    static void successResponse() {
        // GIVEN
```

```
    GeocodingService.GeocodingAddress address = new GeocodingService.GeocodingAddress();
    address.street = STREET;
    address.city = CITY;
    address.postalcode = POSTAL_CODE;
    address.state = STATE;
    address.country = COUNTRY;

    Test.setMock(
        HttpCalloutMock.class,
        new OpenStreetMapHttpCalloutMockImpl()
    );

    // WHEN
    List<GeocodingService.Coordinates> computedCoordinates = GeocodingService.geocodeAddresses(
        new List<GeocodingService.GeocodingAddress>{ address }
    );

    // THEN
    System.assert(
        computedCoordinates.size() == 1,
        'Expected 1 pair of coordinates were returned'
    );
    System.assert(
        computedCoordinates[0].lat == LATITUDE,
        'Expected mock lat was returned'
    );
    System.assert(
        computedCoordinates[0].lon == LONGITUDE,
        'Expected mock lon was returned'
    );
}
@isTest
static void blankAddress() {
    // GIVEN
    GeocodingService.GeocodingAddress address = new GeocodingService.GeocodingAddress();
```

```
    Test.setMock(
        HttpCalloutMock.class,
        new OpenStreetMapHttpCalloutMockImpl()
    );

    // WHEN
    List<GeocodingService.Coordinates> computedCoordinates = GeocodingService.geocodeAddresses(
        new List<GeocodingService.GeocodingAddress>{ address }
    );

    // THEN
    System.assert(
        computedCoordinates.size() == 1,
        'Expected 1 pair of coordinates were returned'
    );
    System.assert(
        computedCoordinates[0].lat == null,
        'Expected null lat was returned'
    );
    System.assert(
        computedCoordinates[0].lon == null,
        'Expected null lon was returned'
    );
}
@isTest
static void errorResponse() {
    // GIVEN
    GeocodingService.GeocodingAddress address = new GeocodingService.GeocodingAddress();
    address.street = STREET;
    address.city = CITY;
    address.postalcode = POSTAL_CODE;
    address.state = STATE;
    address.country = COUNTRY;

    Test.setMock(
        HttpCalloutMock.class,
```

```
        new OpenStreetMapHttpCalloutMockImplError()
    );

    // WHEN
    List<GeocodingService.Coordinates> computedCoordinates = GeocodingService.geocodeAddresses(
        new List<GeocodingService.GeocodingAddress>{ address }
    );

    // THEN
    System.assert(
        computedCoordinates.size() == 1,
        'Expected 1 pair of coordinates were returned'
    );
    System.assert(
        computedCoordinates[0].lat == null,
        'Expected null lat was returned'
    );
    System.assert(
        computedCoordinates[0].lon == null,
        'Expected null lon was returned'
    );
}

public class OpenStreetMapHttpCalloutMockImpl implements HttpCalloutMock {
    public HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setBody('[{"lat": ' + LATITUDE + ',"lon": ' + LONGITUDE + '}]');
        res.setStatusCode(200);
        return res;
    }
}

public class OpenStreetMapHttpCalloutMockImplError implements HttpCalloutMock {
    public HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
```

```
            res.setHeader('Content-Type', 'application/json');
            res.setStatusCode(400);
            return res;
        }
    }
}
```

**LeadProcessor:**

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    // instance member to retain state across transactions
    global Integer recordsProcessed = 0;

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

    global void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {

                lead.LeadSource = 'Dreamforce';
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;

        }
        update leads;
    }

    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');

    }
}
```

**LeadProcessorTest:**

```
@isTest
public class LeadProcessorTest {
 @testSetup
   static void setup() {
      List<Lead> leads = new List<Lead>();
      // insert 200 leads
      for (Integer i=0;i<200;i++) {
         leads.add(new Lead(LastName='Lead '+i,
            Company='Lead', Status='Open - Not Contacted'));
      }
      insert leads;
   }

   static testmethod void test() {
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp, 200);
      Test.stopTest();

      // after the testing stops, assert records were updated properly
      System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
   }
}
```

**MaintenanceRequest:**

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

**MaintenanceRequestHelper:**

```
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:**

```
@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);
    }
}
```

**PagedResult:**

```
public with sharing class PagedResult {
```

```
    @AuraEnabled
    public Integer pageSize { get; set; }

    @AuraEnabled
    public Integer pageNumber { get; set; }

    @AuraEnabled
    public Integer totalItemCount { get; set; }

    @AuraEnabled
    public Object[] records { get; set; }
}
```

**ParkLocator:**

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

**ParkLocatorTest:**

```
@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:**

```
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:**

```
@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) {
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

        response.put('response_x', response_x);
    }
}
```

**PropertyController:**

```
public with sharing class PropertyController {
    private static final Decimal DEFAULT_MAX_PRICE = 9999999;
    private static final Integer DEFAULT_PAGE_SIZE = 9;

    /**
     * Endpoint that retrieves a paged and filtered list of properties
     * @param searchKey String used for searching on property title, city and tags
     * @param maxPrice Maximum price
     * @param minBedrooms Minimum number of bedrooms
```

```
     * @param minBathrooms Minimum number of bathrooms
     * @param pageSize Number of properties per page
     * @param pageNumber Page number
     * @return PagedResult object holding the paged and filtered list of properties
     */
    @AuraEnabled(cacheable=true)
    public static PagedResult getPagedPropertyList(
        String searchKey,
        Decimal maxPrice,
        Integer minBedrooms,
        Integer minBathrooms,
        Integer pageSize,
        Integer pageNumber
    ) {
        // Normalize inputs
        Decimal safeMaxPrice = (maxPrice == null
            ? DEFAULT_MAX_PRICE
            : maxPrice);
        Integer safeMinBedrooms = (minBedrooms == null ? 0 : minBedrooms);
        Integer safeMinBathrooms = (minBathrooms == null ? 0 : minBathrooms);
        Integer safePageSize = (pageSize == null
            ? DEFAULT_PAGE_SIZE
            : pageSize);
        Integer safePageNumber = (pageNumber == null ? 1 : pageNumber);

        String searchPattern = '%' + searchKey + '%';
        Integer offset = (safePageNumber - 1) * safePageSize;

        PagedResult result = new PagedResult();
        result.pageSize = safePageSize;
        result.pageNumber = safePageNumber;
        result.totalItemCount = [
            SELECT COUNT()
            FROM Property__c
            WHERE
                (Name LIKE :searchPattern
```

```
            OR City__c LIKE :searchPattern
            OR Tags__c LIKE :searchPattern)
            AND Price__c <= :safeMaxPrice
            AND Beds__c >= :safeMinBedrooms
            AND Baths__c >= :safeMinBathrooms
        ];
        result.records = [
            SELECT
                Id,
                Address__c,
                City__c,
                State__c,
                Description__c,
                Price__c,
                Baths__c,
                Beds__c,
                Thumbnail__c,
                Location__Latitude__s,
                Location__Longitude__s
            FROM Property__c
            WHERE
                (Name LIKE :searchPattern
                OR City__c LIKE :searchPattern
                OR Tags__c LIKE :searchPattern)
                AND Price__c <= :safeMaxPrice
                AND Beds__c >= :safeMinBedrooms
                AND Baths__c >= :safeMinBathrooms
            WITH SECURITY_ENFORCED
            ORDER BY Price__c
            LIMIT :safePageSize
            OFFSET :offset
        ];
        return result;
    }

    /**
```

```
     * Endpoint that retrieves pictures associated with a property
     * @param propertyId Property Id
     * @return List of ContentVersion holding the pictures
     */
    @AuraEnabled(cacheable=true)
    public static List<ContentVersion> getPictures(Id propertyId) {
        List<ContentDocumentLink> links = [
            SELECT Id, LinkedEntityId, ContentDocumentId
            FROM ContentDocumentLink
            WHERE
                LinkedEntityId = :propertyId
                AND ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')
            WITH SECURITY_ENFORCED
        ];

        if (links.isEmpty()) {
            return null;
        }

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

        for (ContentDocumentLink link : links) {
            contentIds.add(link.ContentDocumentId);
        }

        return [
            SELECT Id, Title
            FROM ContentVersion
            WHERE ContentDocumentId IN :contentIds AND IsLatest = TRUE
            WITH SECURITY_ENFORCED
            ORDER BY CreatedDate
        ];
    }
}
```

**RandomContactFactory:**

```
public class RandomContactFactory {
```

```
    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname) {
        List<Contact> contacts = new List<Contact>();

        for(Integer i=0;i<numcnt;i++) {
            Contact cnt = new Contact(FirstName='Test ' + i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}
```

**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');
                }

        }
}
```

**SampleDataController:**

```
public with sharing class SampleDataController {
    @AuraEnabled
    public static void importSampleData() {
        delete [SELECT Id FROM Case];
        delete [SELECT Id FROM Property__c];
        delete [SELECT Id FROM Broker__c];
        delete [SELECT Id FROM Contact];

        insertBrokers();
        insertProperties();
        insertContacts();
    }
```

```
private static void insertBrokers() {
    StaticResource brokersResource = [
        SELECT Id, Body
        FROM StaticResource
        WHERE Name = 'sample_data_brokers'
    ];
    String brokersJSON = brokersResource.body.toString();
    List<Broker__c> brokers = (List<Broker__c>) JSON.deserialize(
        brokersJSON,
        List<Broker__c>.class
    );
    insert brokers;
}

private static void insertProperties() {
    StaticResource propertiesResource = [
        SELECT Id, Body
        FROM StaticResource
        WHERE Name = 'sample_data_properties'
    ];
    String propertiesJSON = propertiesResource.body.toString();
    List<Property__c> properties = (List<Property__c>) JSON.deserialize(
        propertiesJSON,
        List<Property__c>.class
    );
    randomizeDateListed(properties);
    insert properties;
}

private static void insertContacts() {
    StaticResource contactsResource = [
        SELECT Id, Body
        FROM StaticResource
        WHERE Name = 'sample_data_contacts'
    ];
    String contactsJSON = contactsResource.body.toString();
```

```
        List<Contact> contacts = (List<Contact>) JSON.deserialize(
            contactsJSON,
            List<Contact>.class
        );
        insert contacts;
    }

    private static void randomizeDateListed(List<Property__c> properties) {
        for (Property__c property : properties) {
            property.Date_Listed__c =
                System.today() - Integer.valueof((Math.random() * 90));
        }
    }
}
```

**TestPropertyController:**

```
@isTest
private class TestPropertyController {
    private final static String MOCK_PICTURE_NAME = 'MockPictureName';

    public static void createProperties(Integer amount) {
        List<Property__c> properties = new List<Property__c>();
        for (Integer i = 0; i < amount; i++) {
            properties.add(
                new Property__c(
                    Name = 'Name ' + i,
                    Price__c = 20000,
                    Beds__c = 3,
                    Baths__c = 3
                )
            );
        }
        insert properties;
    }
    static testMethod void testGetPagedPropertyList() {
        TestPropertyController.createProperties(5);
        Test.startTest();
```

```
        PagedResult result = PropertyController.getPagedPropertyList(
            '',
            999999,
            0,
            0,
            10,
            1
        );
        Test.stopTest();
        System.assertEquals(5, result.records.size());
    }

    static testMethod void testGetPicturesNoResults() {
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
            property.Id
        );
        Test.stopTest();

        System.assertEquals(null, items);
    }

    static testMethod void testGetPicturesWithResults() {
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        // Insert mock picture
        ContentVersion picture = new Contentversion();
        picture.Title = MOCK_PICTURE_NAME;
        picture.PathOnClient = 'picture.png';
        picture.Versiondata = EncodingUtil.base64Decode('MockValue');
        insert picture;
```

```
        // Link picture to property record
        List<ContentDocument> documents = [
            SELECT Id, Title, LatestPublishedVersionId
            FROM ContentDocument
            LIMIT 1
        ];
        ContentDocumentLink link = new ContentDocumentLink();
        link.LinkedEntityId = property.Id;
        link.ContentDocumentId = documents[0].Id;
        link.shareType = 'V';
        insert link;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
            property.Id
        );
        Test.stopTest();

        System.assertEquals(1, items.size());
        System.assertEquals(MOCK_PICTURE_NAME, items[0].Title);
    }
}
```

**TestRestrictContactByName:**

```
@IsTest
public class TestRestrictContactByName {
    @IsTest static void createBadContact(){

        Contact c=new Contact(Firstname='John',LastName='INVALIDNAME');

        Test.startTest();
        Database.SaveResult result = Database.insert(c, false);
        Test.stopTest();

        System.assert(!result.isSuccess());
    }
}
```

**TestSampleDataController:**

```
@isTest
private class TestSampleDataController {
    @isTest
    static void importSampleData() {
        Test.startTest();
        SampleDataController.importSampleData();
        Test.stopTest();

        Integer propertyNumber = [SELECT COUNT() FROM Property__c];
        Integer brokerNumber = [SELECT COUNT() FROM Broker__c];
        Integer contactNumber = [SELECT COUNT() FROM Contact];

        System.assert(propertyNumber > 0, 'Expected properties were created.');
        System.assert(brokerNumber > 0, 'Expected brokers were created.');
        System.assert(contactNumber > 0, 'Expected contacts were created.');
    }
}
```

**VerifyDate:**

```
public class VerifyDate {

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

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

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

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

}
```

**TestVerifyDate:**

```
@IsTest
public class TestVerifyDate {
   @isTest static void dateWithin() {
      Date returnDate1 = verifyDate.CheckDates(date.valueOf('2020-02-14'), date.valueOf('2020-02-24') );
      System.assertEquals(date.valueOf('2020-02-24'), returnDate1);
   }

   @isTest static void dateNotWithin() {
      Date returnDate2 = verifyDate.CheckDates(date.valueOf('2020-02-14'), date.valueOf('2020-03-24') );
      System.assertEquals(date.valueOf('2020-02-29'), returnDate2);
   }
}
```

**WarehouseCalloutService:**

```
public with sharing class WarehouseCalloutService {

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

   //@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());

            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 = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

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

        }
    }
}
```

**WarehouseCalloutServiceTest:**

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

**WarehouseCalloutServiceMock**:

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

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

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

**WarehouseSyncScheduleTest:**

```
@isTest
public class WarehouseSyncScheduleTest {
```

```
    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

**WarehouseSyncSchedule:**

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```