

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

Account Manager:

```
@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;
    }
}
```

Account Manager Test:

```
@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');
```

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

Insert acc;

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

 return acc.Id;

 }
}

Account Processor:

public class AccountProcessor {

 //Add annotation to declare a future method

 @future(callout=false)

 public static void countContacts(List<Id> accountIds){

 //Query all accounts in the list of Ids passed

 Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id, (SELECT Id
FROM Contacts) FROM Account WHERE Id IN:accountIds]);

 List<Account> listName = new List<Account>();

 //Loop through list of accounts

 for(Account a: accountMap.values()){

 //Assign field to number of contact

 a.Number_of_Contacts__c=accountMap.get(a.Id).Contacts.size();

 }

 //Update Accounts

 update accountMap.values();

 }
}

Account Processor Test:

@isTest

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

```
private class AccountProcessorTest {

    static TestMethod void myTest() {
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<100; i++) {
            Account account = new Account();
            account.Name = 'AccountProcessorTest Account ' + i;
            accounts.add(account);
        }
        insert accounts;

        List<Id> accountIds = new List<Id>();
        List<Contact> contacts = new List<Contact>();
        for (Account a : accounts) {
            accountIds.add(a.Id);
            for (Integer i=0; i<5; i++) {
                Contact contact = new Contact();
                contact.FirstName = 'AccountProcessor Test Contact';
                contact.LastName = String.valueOf(i);
                contact.AccountId = a.Id;
                contacts.add(contact);
            }
        }
        insert contacts;

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();

        List<Account> results = [SELECT Id, Number_of_Contacts__c
                                FROM Account
                                WHERE Id in :accountIds];
        for (Account a : results) {
            System.AssertEquals(5, a.Number_of_Contacts__c);
        }
    }
}
```

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

```
    }  
  }  
}
```

Add Primary Contact:

```
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;  
    }  
}
```

Add Primary Contact Test:

```
@isTest  
public class AddPrimaryContactTest {  
  
    @isTest  
    public static void testing() {
```

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

```
List<account> acc_lst = new List<account>();
for (Integer i=0; i<50;i++) {
    account a = new account(name=string.valueOf(i),billingstate='NY');
    //This project doc contains the apex codes used in apex modules and apex specialist super badge
    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);
}
}
```

Animal Locator:

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer animalId) {
        String animalName;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
```

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

```
callout.herokuapp.com/animals/'+animalId);
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    // If the request is successful, parse the JSON response.
    if(response.getStatusCode() == 200) {
        Map<String, Object> r = (Map<String, Object>)
            JSON.deserializeUntyped(response.getBody());
        Map<String, Object> animal = (Map<String, Object>)r.get('animal');
        animalName = string.valueOf(animal.get('name'))
        //This project doc contains the apex codes used in apex modules and apex
specialist super
badge
    }
    return animalName;
}
}
```

Animal Locator Mock:

```
@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;
    }
}
```

Animal Locator Test:

```
@isTest
private class AnimalLocatorTest {
```

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

```
@isTest static void getAnimalNameByIdTest() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    String response = AnimalLocator.getAnimalNameById(1);

    // Verify that the response received contains fake values

    System.assertEquals('chicken', response);
}
}
```

AsyncParkService:

//Generated by wsdl2apex

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

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

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

Closed Opportunity Trigger:

```
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){
```


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

```
        insert tasklist;
    }
}
```

ContactsToday Controller:

```
public class ContactsTodayController {
```

```
    @AuraEnabled
```

```
    public static List<Contact> getContactsForToday() {
```

```
        List<Task> my_tasks = [SELECT Id, Subject, Whold FROM Task WHERE OwnerId = :UserInfo.getUserId() AND IsClosed = false AND Whold != null];
```

```
        List<Event> my_events = [SELECT Id, Subject, Whold FROM Event WHERE OwnerId = :UserInfo.getUserId() AND StartDateTime >= :Date.today() AND Whold != 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.Whold);
        }
```

```
        for(Event evt : my_events) {
            contactIds.add(evt.Whold);
        }
```

```
        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) {
```

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

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

```

);
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',

```

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

badge

ContactId = c.Id

```

);
insert cse;
List 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 contacts = ContactsTodayController.getContactsForToday();

```

```

//This project doc contains the apex codes used in apex modules and apex specialist
super
badge
System.assertEquals(0, contacts.size());
}
}

```

Create Default Data:

```

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:

```

@Test
private class CreateDefaultDataTest {
    @Test
    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 sc){
        List<Lead> lstofLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT
200];
        List<Lead> lstofupdatedLead=new List<Lead>();
        if(!lstofLead.isEmpty()){
            for(Lead Id:lstofLead){
                Id.LeadSource = 'Dreamforce';
                lstofupdatedLead.add(Id);
            }
            UPDATE lstofupdatedLead;
        }
    }
}

```

DailyLeadProcessorTest:

```

    @isTest
    private class DailyLeadProcessorTest {

```



```

@TestSetup
static void setup(){
    List<Lead> lstofLead = new List<Lead>();
    for(Integer i = 1; i <= 200; i++){
        Lead Id = new Lead(Company = 'Comp' + i, LastName = 'LN' + i, status='working -
Contacted');
        lstofLead.add(Id);
    }
    Insert lstofLead;
}

static testmethod void testDailyLeadProcessorscheduledJob(){
    String sch = '0 5 12 * * ?';
    Test.startTest();
    String jobId = System.Schedule('ScheduledApexTest', sch, new
DailyLeadProcessor());
    List<Lead> lstofLead=[SELECT Id FROM Lead WHERE Leadsources = null LIMIT 200];
    system.assertEquals(200, lstofLead.size());
    Test.stopTest();
}
}

```

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 = 'Armillá';  
    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'
);
}
@Test
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'
);
}
@Test
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:

```

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

```

```

    /**

```

```

     * コンストラクタ

```

```

    */

```

```

    public LeadProcessor() {

```

```

    }

```

```

    /**

```

```

     * Start

```

```

    */

```

```

    public Database.QueryLocator start(Database.BatchableContext BC) {

```

```

        String query = 'SELECT Id FROM Lead';

```

```

        return Database.getQueryLocator (query);

```

```

    }

```

```

    /**

```

```

     * Execute

```

```

    */

```

```

    public void execute(Database.BatchableContext BC, List<Lead> leads) {

```

```

        for (Lead l : leads) {

```

```

            l.LeadSource = 'Dreamforce';

```

```

        }

```

```

        update leads;

```

```

    }

```

```

    /**

```

```

    * Finish
    */
    public void finish(Database.BatchableContext BC) {

    }
}

```

LeadProcessorTest:

```

@Test
private class LeadProcessorTest {

    private static User testAdminUser = new User(Id = UserInfo.getUserId());

    static testMethod void LeadProcessorTest() {

        System.runAs(testAdminUser) {

            List<Lead> leads = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                leads.add(new Lead(LastName = 'Yoshikawa', Company = 'T.Yoshikawa Labs'));
            }
            insert leads;
            System.assertEquals(leads.size(), 200);

            Test.startTest();

            LeadProcessor batchable = new LeadProcessor();
            Database.executeBatch(batchable);

            Test.stopTest();

            List<Lead> results = [SELECT Id,LeadSource FROM Lead];
            for (Lead l : results) {
                System.assertEquals(l.LeadSource, 'Dreamforce');
            }
            System.assertEquals(results.size(), 200);
        }
    }
}

```



```
}
```

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


```

        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:

```

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

```

```

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

```

ParkService:

//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;
    }
}
}

```

ParkServiceMock:

```

@Test
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:

```

//@isTest
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }
}

```

RestrictContactByName:

```

trigger RestrictContactByName on Contact (before insert) {
    //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));
    }
}
}

```

TestRestrictContactByName:

@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.');
```

```

    }
}
```

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:

```
public class TestRestrictContactByName {  
    @isTest  
    public class TestRestrictContactByName {  
        @isTest static void Test_insertupdateContact(){  
            Contact cnt = new Contact();  
            cnt.LastName = 'INVALIDNAME';
```

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

```
System.assert(!result.isSuccess());  
System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',  
result.getErrors()[0].getMessage());  
}  
}
```

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.');
```

```

    }
}

```

TestVerifyDate:

```

@Test
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }
}

```

```

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}

```

VerifyDate:

```

public class VerifyDate {

    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
    @TestVisible 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
    @TestVisible 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;
    }
}

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;
}
}

```

WarehouseSyncSchedule:

```

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

WarehouseSyncSheduleTest:

```

@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 ');
    }
}

```

```
}  
}
```

AnimalCallouts:

```
public class AnimalsCallouts {  
    public static String getAnimalNameById(Integer animalId) {  
        String animalName;  
        Http http = new Http();  
        HttpRequest request = new HttpRequest();  
        request.setEndpoint('https://th-apex-http-  
callout.herokuapp.com/animals'+animalId);  
        request.setMethod('GET');  
        HttpResponse response = http.send(request);  
        // If the request is successful, parse the JSON response.  
        if(response.getStatusCode() == 200) {  
            Map<String, Object> r =(Map<String, Object>)  
                JSON.deserializeUntyped(response.getBody());  
            Map<String, Object> animal = (Map<String , Object>)r.get('animal');  
            animalName = string.valueOf(animal.get('name'));  
        }  
        return animalName;  
    }  
}
```

AnimalCalloutTest:

```
@isTest  
private class AnimalsCalloutsTest {  
    @isTest static void testGetCallout() {  
        // Create the mock response based on a static resource  
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();  
        mock.setStaticResource('GetAnimalResource');  
        mock.setStatusCode(200);  
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');  
        // Associate the callout with a mock response  
        Test.setMock(HttpCalloutMock.class, mock);  
        // Call method to test  
        HttpResponse result = AnimalsCallouts.makeGetCallout();  
        // Verify mock response is not null
```



```
System.assertNotEquals(null,result, 'The callout returned a null response.');
```

```
// Verify status code
```

```
System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
```

```
// Verify content type
```

```
System.assertEquals('application/json;charset=UTF-8',
```

```
    result.getHeader('Content-Type'),
```

```
    'The content type value is not expected.');
```

```
// Verify the array contains 3 items
```

```
Map<String, Object> results = (Map<String, Object>)
```

```
    JSON.deserializeUntyped(result.getBody());
```

```
List<Object> animals = (List<Object>) results.get('animals');
```

```
System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
```

```
}
```

```
}
```