

AccountAddressTrigger:

```
trigger AccountAddressTrigger on Account (beforeinsert, 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 request = RestContext.request;
    string accountId = request.requestURI.substringBetween('Accounts/', '/contacts');
    Account result = [SELECT Id, Name, (SelectId, Name from Contacts) from Account where
Id=:accountId Limit 1];
    return result;
  }
}
```

AccountManagerTest:

```
@IsTest
private class AccountManagerTest {
```

```

@isTest static void
testGetContactsByAccountId(){Id recordId =
createTestRecord();
RestRequest request = new RestRequest();
request.requestUri =
'https:// yourInstance.my.salesforce.com/services/apexrest/Accounts/'

+ recordId+'/contacts';
request.httpMethod = 'GET';
RestContext.request = request;
Account thisAccount = AccountManager.getAccount();

System.assert(thisAccount !=
null);System.assertEquals('Test
record',thisAccount.Name);
}

static Id createTestRecord(){
Account accountTest = new
Account(Name ='Test record');
insert accountTest;

Contact contactTest = new Contact(
    FirstName='John',
    LastName = 'Doe',
    AccountId=
    accountTest.Id

);
insert contactTest;

return accountTest.Id
;
}
}

```

AccountProcessor:

```
public class AccountProcessor
{
    @future
    public static void countContacts(Set<id> setId)
    {
        List<Account> lstAccount = [select id,Number_of_Contacts_c , (selectid from contacts)
from account where id in :setId];
        for( Accountacc : lstAccount )
        {
            List<Contact> lstCont = acc.contacts ;

            acc.Number_of_Contacts_c= lstCont.size();
        }
        update lstAccount;
    }
}
```

AccountProcessorTest:

```
@IsTest
public class AccountProcessorTest {
    public static testmethod void TestAccountProcessorTest()
    {
        Account a = new
        Account();a.Name = 'Test
        Account'; Insert a;

        Contact cont = New
        Contact();cont.FirstName
        ='Bob'; cont.LastName
        ='Masters'; cont.AccountId
        = a.Id;
```

```
Insert cont;
```

```
set<Id> setAcclId = new Set<ID>();  
setAcclId.add(a.id);
```

```
Test.startTest();  
    AccountProcessor.countContacts(setAcclId);  
Test.stopTest();
```

```
Account ACC = [select Number_of_Contacts_c from Accountwhere id = :a.id LIMIT 1];  
System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts_c),1);  
}  
  
}
```

AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable {  
    public Contact;   
    public String state;  
  
    public AddPrimaryContact(Contact c, Stringstate) {  
        this.c= c;  
        this.state = state;  
    }  
  
    public void execute(QueueableContext qc) {  
  
        system.debug('this.c = '+this.c+' this.state = '+this.state);  
        List<Account> acc_lst = new List<account>([select id, name, BillingState from account  
where account.BillingState = :this.state limit 200]);  
        List<contact> c_lst = new List<contact>();  
        for(account a: acc_lst) {  
            contact c = new contact();  
            c = this.c.clone(false, false,false, false);  
            c.AccountId = a.Id;  
            c_lst.add(c);  
        }  
    }  
}
```

```

    }
    insert c_lst;
}

}

```

AddPrimaryContactTest:

```

@Test
public class AddPrimaryContactTest {

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

        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }
}

```

```
}
```

AnimalLocator:

```
public class AnimalLocator{
    public static String getAnimalNameById(Integer
        x){Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal= new Map<String, Object>();
        HttpResponse res = http.send(req);
        if(res.getStatusCode() == 200) {
            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String,
Object>)results.get('animal');
        }
        return (String)animal.get('name');
    }
}
```

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('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
            "mighty moose"]}');
        response.setStatusCode(200);
    }
}
```

```

        return response;
    }
}

```

AnimalLocatorTest:

```

@Test
private class AnimalLocatorTest{
    @Test static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String result = AnimalLocator.getAnimalNameById(3);
        String expectedResult =
            'chicken'; System.assertEquals(result, expectedResult );
    }
}

```

AnimalsCallouts:

```

public class AnimalsCallouts {
    public static HttpResponse makeGetCallout()
    {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        / If the request is successful, parse the JSON
        response.if(response.getStatusCode() == 200) {
            / Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            / Cast the values in the 'animals' key as a list
            List<Object> animals = (List<Object>)
            results.get('animals'); System.debug('Received the
            following animals:'); for(Object animal:animals) {
                System.debug(animal);
            }
        }
    }
}

```

```

    }
}
return response;
}

public static HttpResponse makePostCallout() {
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
    request.setMethod('POST');
    request.setHeader('Content-Type', 'application/json;charset=UTF-8');

    request.setBody('{"name":"mighty moose"}');
    HttpResponse response = http.send(request);
    / Parse the JSON response
    if(response.getStatusCode() != 201) {
        System.debug('The status code returned was not expected: ' +
            response.getStatusCode() + ' ' + response.getStatus());
    } else {
        System.debug(response.getBody());
    }
    return response;
}
}

```

AnimalsCalloutsTest:

```

@Test
private class AnimalsCalloutsTest {
    @Teststatic void testGetCallout() {
        / Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new
        StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.getStatusCode(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 statuscode
    System.assertEquals(200,result.getStatusCode(), 'The status code is not 200. ');
    / Verify contenttype
    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. ');
}
}

```

AnimalsHttpCalloutMock:

```

@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    / Implement this interface method
    global HttpResponse respond(HTTPRequest request) {
        / Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
            'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
            "mighty moose"]}');
        response.getStatusCode(200);
        return response;
    }
}

```

AsyncCalculatorServices:

```
public class AsyncCalculatorServices {  
    public class doDivideResponseFuture extends System.WebServiceCalloutFuture  
    {public Double getValue() {  
        calculatorServices.doDivideResponse response =  
(calculatorServices.doDivideResponse)System.WebServiceCallout.endInvoke(this);  
        return response.return_x;  
    }  
    }  
    public class doSubtractResponseFuture extends System.WebServiceCalloutFuture  
    {public DoublegetValue() {  
        calculatorServices.doSubtractResponse response =  
(calculatorServices.doSubtractResponse)System.WebServiceCallout.endInvoke(this);  
        return response.return_x;  
    }  
    }  
    public class doMultiplyResponseFuture extends System.WebServiceCalloutFuture  
    {public DoublegetValue() {  
        calculatorServices.doMultiplyResponse response =  
(calculatorServices.doMultiplyResponse)System.WebServiceCallout.endInvoke(this);  
        return response.return_x;  
    }  
    }  
  
    public class doAddResponseFuture extends System.WebServiceCalloutFuture  
    {public Double getValue() {  
        calculatorServices.doAddResponse response =  
(calculatorServices.doAddResponse)System.WebServiceCallout.endInvoke(this);  
        return response.return_x;  
    }  
    }  
    public class AsyncCalculatorImplPort {  
        public String endpoint_x = 'https:// th-apex-soap-  
service.herokuapp.com/service/calculator'; public Map<String,String>  
inputHttpHeaders_x;  
        publicString clientCertName_x;
```

```

        public Integertimeout_x;
        private String[]ns_map_type_info = new String[]{"http:/ calculator.services/",
'calculatorServices'};
        public AsyncCalculatorServices.doDivideResponseFuture
beginDoDivide(System.Continuation continuation,Double arg0,Double arg1) {
        calculatorServices.doDivide request_x= new
calculatorServices.doDivide();request_x.arg0= arg0;
request_x.arg1 = arg1;
        return (AsyncCalculatorServices.doDivideResponseFuture)
System.WebServiceCallout.beginInvoke(
        this,
        request
        _x,
        AsyncCalculatorServices.doDivideResponseFuture.class,
        continuation,
        new
        String[]{"endpoint_x,",
'http:/calculator.services/',
'doDivide',
'http:/ calculator.services/',
'doDivideResponse',
'calculatorServices.doDivideResponse'}
        );
    }

    public AsyncCalculatorServices.doSubtractResponseFuture
beginDoSubtract(System.Continuation continuation,Double arg0,Double arg1) {
        calculatorServices.doSubtract request_x = new calculatorServices.doSubtract();
request_x.arg0 = arg0;
request_x.arg1 = arg1;
        return (AsyncCalculatorServices.doSubtractResponseFuture)
System.WebServiceCallout.beginInvoke(

        this,
        request
        _x,
        AsyncCalculatorServices.doSubtractResponseFuture.class,
        continuation,
        new
        String[]{"endpoint_x,",
'http:/calculator.services/',

```

```

        'doSubtract',
        'http:/ calculator.services/',
        'doSubtractResponse',
        'calculatorServices.doSubtractResponse'}
    );
}

public AsyncCalculatorServices.doMultiplyResponseFuture
beginDoMultiply(System.Continuation continuation,Double arg0,Double arg1) {
    calculatorServices.doMultiply request_x= new calculatorServices.doMultiply();
    request_x.arg0 = arg0;
    request_x.arg1 = arg1;
    return (AsyncCalculatorServices.doMultiplyResponseFuture)
System.WebServiceCallout.beginInvoke(
    this,
    request
    _x,
    AsyncCalculatorServices.doMultiplyResponseFuture.class,
    continuation,
    new
    String[]{endpoint_x,"
'http:/calculator.services/',
'doMultiply',
'http:/ calculator.services/',
'doMultiplyResponse',
'calculatorServices.doMultiplyResponse'}
    );
}

public AsyncCalculatorServices.doAddResponseFuture
beginDoAdd(System.Continuation continuation,Double arg0,Double arg1) {
    calculatorServices.doAdd request_x= new calculatorServices.doAdd();
    request_x.arg0 = arg0;
    request_x.arg1 = arg1;
    return (AsyncCalculatorServices.doAddResponseFuture)
System.WebServiceCallout.beginInvoke(
    this,

    request_x,
    AsyncCalculatorServices.doAddResponseFuture.class,
    continuation,

```

```

        new
        String[]{endpoint_x,
        'http://calculator.services
        /','doAdd',
        'http:// calculator.services/',
        'doAddResponse',
        'calculatorServices.doAddResponse'}
    );
}
}
}

```

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 {
        publicString endpoint_x = 'https:// th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        publicString clientCertName_x;
        public Integertimeout_x;
        private String[] ns_map_type_info = new String[]{'http:// parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture
        beginByCountry(System.Continuation
continuation,Stringarg0) {
            ParkService.byCountry request_x= new ParkService.byCountry();
            request_x.arg0 = arg0;

```

```

        return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
    this,
    request
    _x,

    AsyncParkService.byCountryResponseFuture.clas
    s,continuation,
    new
    String[]{endpoint_x,",
    'http:/parks.services/',
    'byCountry',
    'http:/ parks.services/',
    'byCountryResponse',
    'ParkService.byCountryResponse'}
    );
    }
}
}
}

```

CalculatorServices:

```

public class calculatorServices {
    public class
    doDivideResponse {
        public Double return_x;
        private String[] return_x_type_info = new
String[]{'return','http:/ calculator.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class
    doMultiply {public
    Double arg0;
    publicDouble arg1;
    private String[] arg0_type_info = new
String[]{'arg0','http:/

```

```

calculator.services/',null,'0','1','false');
    private String[] arg1_type_info = new
String[]{'arg1','http:/
calculator.services/',null,'0','1','false');
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false};
    private String[] field_order_type_info = new String[]{'arg0','arg1'};
}
public class doAdd {
    public Double arg0;
    public Double arg1;
    private String[] arg0_type_info = new
String[]{'arg0','http:/
calculator.services/',null,'0','1','false'};

    private String[] arg1_type_info = new
String[]{'arg1','http:/
calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false};
    private String[] field_order_type_info = new String[]{'arg0','arg1'};
}
public class doAddResponse{
    public Double return_x;
    private String[] return_x_type_info = new
String[]{'return','http:/ calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false};
    private String[] field_order_type_info = new String[]{'return_x'};
}
public class
doDivide {public
Double arg0;
public Double
arg1;
    private String[] arg0_type_info = new
String[]{'arg0','http:/
calculator.services/',null,'0','1','false'};
    private String[] arg1_type_info = new

```

```

String[]{'arg1','http:/
calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false'};
    private String[] field_order_type_info = new String[]{'arg0','arg1'};
}
public class
doSubtract {public
    Double arg0; public
    Double arg1;
    private String[] arg0_type_info = new
String[]{'arg0','http:/
calculator.services/',null,'0','1','false'};
    private String[] arg1_type_info = new
String[]{'arg1','http:/
calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false'};
    private String[] field_order_type_info = new String[]{'arg0','arg1'};
}
public class doSubtractResponse {
    public Double return_x;
    private String[] return_x_type_info = new
String[]{'return','http:/ calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new

String[]{'http:/ calculator.services/',false,false'};
    private String[] field_order_type_info = new String[]{'return_x'};
}
public class doMultiplyResponse
{public Double return_x;
    private String[] return_x_type_info = new
String[]{'return','http:/ calculator.services/',null,'0','1','false'};
    private String[] apex_schema_type_info = new
String[]{'http:/ calculator.services/',false,false'};
    private String[] field_order_type_info = new String[]{'return_x'};
}
public class CalculatorImplPort {
    public String endpoint_x = 'https:/ th-apex-soap-

```

```

service.herokuapp.com/service/calculator'; 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:/ calculator.services/',
'calculatorServices'};
public Double doDivide(Double arg0,Double arg1) {
    calculatorServices.doDivide request_x= new
    calculatorServices.doDivide();request_x.arg0 = arg0;
    request_x.arg1 = arg1;
    calculatorServices.doDivideResponse response_x;
    Map<String, calculatorServices.doDivideResponse> response_map_x = new Map<String,
calculatorServices.doDivideResponse>();
    response_map_x.put('response_x', response_x);
    WebServiceCallout.invoke(
        this,
        request
        _x,
        response_map_x,
        new
        String[]{endpoint_x,"
'http:/calculator.services/',
'doDivide',
'http:/ calculator.services/',
'doDivideResponse',
'calculatorServices.doDivideResponse'}
    );

    response_x =
    response_map_x.get('response_x');return
    response_x.return_x;
}
public Double doSubtract(Double arg0,Double arg1) {
    calculatorServices.doSubtract request_x = new calculatorServices.doSubtract();
    request_x.arg0 = arg0;
    request_x.arg1 = arg1;

```

```

        calculatorServices.doSubtractResponse response_x;
        Map<String, calculatorServices.doSubtractResponse> response_map_x =
new Map<String, calculatorServices.doSubtractResponse>();
        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
            this,
            request
            _x,
            response_map_x,
            new
            String[]{endpoint_x,"
            'http:/calculator.services/',
            'doSubtract',
            'http:/ calculator.services/',
            'doSubtractResponse',
            'calculatorServices.doSubtractResponse'}
        );
        response_x =
        response_map_x.get('response_x');return
        response_x.return_x;
    }

    public Double doMultiply(Double arg0,Double arg1) {
        calculatorServices.doMultiply request_x= new
        calculatorServices.doMultiply(); request_x.arg0 = arg0;
        request_x.arg1 = arg1;
        calculatorServices.doMultiplyResponse response_x;
        Map<String, calculatorServices.doMultiplyResponse> response_map_x =
new Map<String, calculatorServices.doMultiplyResponse>();
        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
            this,
            request
            _x,
            response_map_x,
            new
            String[]{endpoint_x,"
            'http:/ calculator.services/',

            'doMultiply',

```

```

        'http:/ calculator.services/',
        'doMultiplyResponse',
        'calculatorServices.doMultiplyResponse'}
    );
    response_x =
    response_map_x.get('response_x');return
    response_x.return_x;
}

public Double doAdd(Double arg0,Double arg1) {
    calculatorServices.doAdd request_x= new
    calculatorServices.doAdd(); request_x.arg0 = arg0;
    request_x.arg1 = arg1;
    calculatorServices.doAddResponse response_x;
    Map<String, calculatorServices.doAddResponse> response_map_x = new Map<String,
calculatorServices.doAddResponse>();
    response_map_x.put('response_x', response_x);
    WebServiceCallout.invoke(
        this,
        request
        _x,
        response_map_x,
        new
        String[]{endpoint_x,"
'http:/calculator.services
/','doAdd',
'http:/ calculator.services/',
'doAddResponse',
'calculatorServices.doAddResponse'}
    );
    response_x =
    response_map_x.get('response_x');return
    response_x.return_x;
}
}
}

```

ClosedOpportunityTrigger:

```

triggerClosedOpportunityTrigger on Opportunity (after insert, afterupdate) {
    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;}
}

```

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){
        if(tsk.Whold == c.Id) {
            c.Description += 'Becauseof Task "' + tsk.Subject + "'\n';
        }

    }

    for(Event evt :
    my_events) {
        if(evt.Whold == c.Id) {
            c.Description += 'Becauseof Event "' + evt.Subject + "'\n';
        }
    }

    for(Case cse : my_cases) {
        if(cse.ContactId == c.Id){
            c.Description += 'Becauseof Case "' + cse.Subject + "'\n';
        }
    }

    }

    return contacts;
}

```

```

}

```

ContactsToday ControllerTest:

@IsTest

public class ContactsTodayControllerTest {

@IsTest

public static void testGetContactsForToday() {

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

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

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

Status = 'Not Started'
);
inserttsk;

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

```
insertevt;
```

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

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

```
}
```

```
@IsTest
```

```
public static void testGetNoContactsForToday() {
```

```
Account acct = new  
    Account(Name = 'Test  
    Account'  
);  
insertacct;
```

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

```
Task tsk = new  
    Task( Subject =  
        'Test Task',Whold
```

```
        = c.Id,  
        Status = 'Completed'  
    );  
    inserttsk;
```

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

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

```
    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 metadataHow_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_Settingsc.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_Settingsc.getOrgDefaults();
    customSetting.Is_Data_Createdc = 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 TrailerRV', Air_Conditioner_c = true,
Bathrooms_c = 2, Bedrooms_c = 2, Model_c = 'TravelTrailer 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'));
        insertvehicles;
        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));
        insertequipments;
        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()));
        insertmaintenanceRequests;
        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_Requestc = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipment_c
                =equipment.get(1).Id, Maintenance_Requestc =
                maintenanceRequest.get(0).Id));joinRecords.add(new
Equipment_Maintenance_Item_c(Equipment_c =equipment.get(2).Id,
                Maintenance_Requestc = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipment_c
                =equipment.get(0).Id, Maintenance_Requestc =
                maintenanceRequest.get(1).Id));joinRecords.add(new
Equipment_Maintenance_Item_c(Equipment_c =equipment.get(1).Id,
                Maintenance_Requestc = 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;retu
rn 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_Settingsc.getOrgDefaults();
        customSetting.Is_Data_Createdc = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom
settingHow_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){
    List<Lead> leads = [SELECTId, LeadSource FROM Lead WHERE LeadSource = "];

    if(leads.size() > 0){

```

```

List<Lead> newLeads = new List<Lead>();

for(Lead lead :
    leads){lead.LeadSource =
    'DreamForce';
    newLeads.add(lead);
}

    update newLeads;
}
}
}

```

DailyLeadProcessorTest:

```

@isTest
private class DailyLeadProcessorTest{
    @testSetup
    static void setup(){
        List<Lead> lstofLead = new List<Lead>();
        for(Integer i = 1; i <=200; i++){
            Lead ld = new Lead(Company = 'Comp' + i, LastName= 'LN' + i, status='working -
Contacted');

                                }

lstofLead.add(ld);

        Insert lstofLead;
    }

    static testmethod void testDailyLeadProcessorscheduledJob(){
        String sch = '0 5 12 * * ?';
        Test.startTest();
        String jobId = System.Schedule('ScheduledApexText', sch, new
DailyLeadProcessor());
    }
}

```

```

        List<Lead> lstofLead=[SELECT Id FROM Lead WHERE Leadsourcesource = 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
    postcode;
}

public class
Coordinates{
    @InvocableVariable
    public Decimal lat;
    @InvocableVariable
    public Decimallon;
}
}

```

GeocodingServiceTest:

```

@Test
private with sharing class GeocodingServiceTest {
    private static final String STREET = 'Caminodel Jueves 26';
    private staticfinal String CITY = 'Armillla';
    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 staticfinal Decimal LONGITUDE = 31.333;

    @Test
    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'
    );
}
@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:

```

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

    / Creating a variable that will keep the count of Leads processed:
    global Integer recordsProcessed = 0;

    / Retrieving all Leads records(First step in Batch)
    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([SELECT ID, LeadSource FROM
            Lead]);
    }

    / Processing all retrieved records(Second step in Batch)
    global void execute(Database.BatchableContext bc, List<Lead> scope)
    {
        for (Lead lead : scope){
            lead.LeadSource = 'Dreamforce';
            recordsProcessed = recordsProcessed + 1;
            System.debug(lead.LeadSource);
        }
        updatescope;
    }
}

```

```

    }

    / Finishing(Final step in Batch)
    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed+ ' records processed. Shazam!');
    }
}

```

LeadProcessorTest:

```

@Test
private class LeadProcessorTest {

    / Creating 200 lead records to Test
    @TestSetup
    static void setup(){
        List<Lead> leads = new List<Lead>();

        for (Integer i = 0; i < 200; i++) {
            / Adding a new lead to the lead list
            leads.add(new Lead(LastName='Lead ' + i, Company='Company Number ' + i,
                Status='Open - Not Contacted'));
        }

        / Inserting the lead
        listinsert leads;
    }

    static testMethod void test() {

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

```
    / When an existingmaintenance request of type Repairor Routine Maintenance is closed,
    / create a new maintenance request for a future routinecheckup.
    if (!validIds.isEmpty()){
        Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle_c, Equipment_c,
        Equipment_r.Maintenance_Cycle_c,
        (SELECT Id,Equipment_c,Quantity_c FROM
```

```

Equipment_Maintenance_Items_r)
                FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

    / calculate the maintenance request due dates by using the maintenance cycle defined
on the related equipment records.
    AggregateResult[] results = [SELECT Maintenance_Request_c,
                MIN(Equipment_r.Maintenance_Cycle_c) cycle
                FROM Equipment_Maintenance_Item_c
                WHERE Maintenance_Request_c IN :ValidIds GROUP BY
Maintenance_Request_c];

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

    List<Case> newCases = new List<Case>();
    for(Case cc : closedCases.values()){
        Case nc = new
            Case (ParentId=
                cc.Id,

                Status = 'New',
                Subject= 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle_c = cc.Vehicle_c,
                Equipment_c =cc.Equipment_
                c,Origin = 'Web',
                Date_Reported_c = Date.Today()
            );

        / If multiple pieces of equipment are used in the maintenance request,
        / define the due date by applying the shortest maintenance cycle to today's date.
        / If (maintenanceCycles.containsKey(cc.Id)){
            nc.Date_Due_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
        / } else {
        /     nc.Date_Due_c = Date.today().addDays((Integer)

```

```

cc.Equipment_r.maintenance_Cycle_c);
    / }

    newCases.add(nc);
}

insert newCases;

List<Equipment_Maintenance_Item_c> clonedList = new
List<Equipment_Maintenance_Item_c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item_c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items_r){
        Equipment_Maintenance_Item_c item = clonedListItem.clone();
        item.Maintenance_Request_c= nc.Id;
        clonedList.add(item);
    }
}
insert clonedList;
}
}
}
}

```

MaintenanceRequestHelperTest:

@isTest

```

public with sharing class MaintenanceRequestHelperTest {

    / createVehicle
    private staticVehicle_c createVehicle(){
        Vehicle_c vehicle= new Vehicle_C(name = 'Testing Vehicle');
        return vehicle;
    }

    / createEquipment

```

```

private static Product2 createEquipment(){
    product2 equipment = new product2(name = 'Testing equipment',
        lifespan_months_c = 10,
        maintenance_cycle_c = 10,
        replacement_part_c = true);
    return equipment;
}

/ createMaintenanceRequest
private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cse = new case(Type='Repair',
        Status='New',
        Origin='Web',
        Subject='Testing
subject',
        Equipment_c=equipmentId,
        Vehicle_c=vehicleId);
    return cse;
}

/ createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item_c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
    Equipment_Maintenance_Item_c equipmentMaintenanceItem = new
Equipment_Maintenance_Item_c(
    Equipment_c = equipmentId,
    Maintenance_Request_c = requestId);
    return equipmentMaintenanceItem;
}

@isTest
private static void testPositive(){
    Vehicle_c vehicle = createVehicle();

    insert vehicle;
    id vehicleId = vehicle.Id;

```

```
Product2 equipment =  
createEquipment();insert equipment;  
id equipmentId = equipment.Id;
```

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

```
Equipment_Maintenance_Item_c equipmentMaintenanceItem =  
createEquipmentMaintenanceItem(equipmentId,createdCase.id);  
insert equipmentMaintenanceItem;
```

```
test.startTest();  
createdCase.status=  
'Closed';update  
createdCase;  
test.stopTest();
```

```
Case newCase = [Select  
    id,subject,  
    type,  
    Equipment_  
    c,  
    Date_Reported_c,  
    Vehicle_c,  
    Date_Due_  
    cfrom case  
    where status ='New'];
```

```
Equipment_Maintenance_Item_cworkPart = [selectid  
    from Equipment_Maintenance_Item_c  
    where Maintenance_Request_c  
    =:newCase.Id];list<case> allCase= [select id from case];  
system.assert(allCase.size() == 2);
```

```
system.assert(newCase != null);  
system.assert(newCase.Subject != null);  
system.assertEquals(newCase.Type, 'Routine
```

```

Maintenance');
SYSTEM.assertEquals(newCase.Equipmentc, equipmentId);
SYSTEM.assertEquals(newCase.Vehicle_c, vehicleId);
SYSTEM.assertEquals(newCase.Date_Reported_c, system.today());

}

```

```

@Test
private static void testNegative(){
    Vehicle_C vehicle = createVehicle();
    insertvehicle;
    id vehicleId = vehicle.Id;

    product2 equipment =
    createEquipment();insert equipment;
    id equipmentId = equipment.Id;

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

    Equipment_Maintenance_Item_c workP = createEquipmentMaintenanceltem(equipmentId,
createdCase.Id);
    insert workP;

    test.startTest();
    createdCase.Status= 'Working';
    update createdCase;
    test.stopTest();

    list<case> allCase= [select id from case];

    Equipment_Maintenance_Item_c equipmentMaintenanceltem = [select id
                                                                from Equipment_Maintenance_Item_c
                                                                where Maintenance_Request_c= :createdCase.Id];

    system.assert(equipmentMaintenanceltem != null);

```

```

    system.assert(allCase.size() == 1);
}

@isTest
private static void testBulk(){
    list<Vehicle_C> vehicleList = new list<Vehicle_C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item_c> equipmentMaintenanceltemList = new
list<Equipment_Maintenance_Item_c>();
    list<case> caseList = new list<case>();

    list<id> oldCaseIds = new list<id>();

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

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

    for(integer i = 0; i < 300; i++){

equipmentMaintenanceltemList.add(createEquipmentMaintenanceltem(equipmentList.get(i).id,
caseList.get(i).id));
    }
    insert equipmentMaintenanceltemList;

    test.startTest();
    for(case cs :
    caseList){

```

```

        cs.Status = 'Closed';
        oldCaseIds.add(cs.Id);
    }
    updatecaseList;
    test.stopTest();

    list<case> newCase= [select id
                        from case
                        where status ='New'];

```

```

list<Equipment_Maintenance_Item_c>workParts = [select id
                                                from Equipment_Maintenance_Item_c
                                                where Maintenance_Request_c in: oldCaseIds];

```

```

system.assert(newCase.size()== 300);

```

```

    list<case> allCase = [select id from
    case];system.assert(allCase.size() == 600);
}
}

```

OpportunityAlertController:

```

public class OpportunityAlertController {

```

```

    @AuraEnabled

```

```

    public static List<Opportunity> getOpportunities(Decimal daysSinceLastModified, String
oppStage, Boolean hasOpen){
        DateTime lastModifiedDateFilter =
DateTime.now().addDays((Integer)daysSinceLastModified * -1);
        List<Opportunity> opportunities = [
            SELECT Id, Name, StageName, LastModifiedDate, CloseDate
            FROM Opportunity

```

```

        WHERE StageName = :oppStage AND LastModifiedDate <= :lastModifiedDateFilter
    ];
    Map<Id,Opportunity> oppMap = new
    Map<Id,Opportunity>(opportunities); if(hasOpen == true) {
        List<Task> tasks = [SELECT ID, WhatId FROM TASK WHERE IsClosed = false AND WhatId
    IN :oppMap.keySet()];
        List<Opportunity> opps_with_tasks = new List<Opportunity>();
        for(Task ta : tasks){
            if(oppMap.containsKey(ta.WhatId)) {
                opps_with_tasks.add(oppMap.get(ta.WhatId));
            }
        }
        opportunities = opps_with_tasks;
    }
    return opportunities;
}

}

```

OpportunityAlertControllerTest:

```

@Test
public class OpportunityAlertControllerTest {

    @Test
    public static void testGetOpptyWithoutOpenTasks() {

        Opportunity oppty = new
            Opportunity(Name = 'Test Oppty',
                CloseDate = Date.today(),
                StageName = 'Prospecting'
            );
        insert oppty;

        Task tsk = new
            Task( Subject =

```

```

        'Test Task',WhatId
        = oppty.Id, Status
        = 'Completed'
    );
    insert tsk;

    List<Opportunity>

    opps;

    opps = OpportunityAlertController.getOpportunities(0, 'Prospecting', false);
    System.assertEquals( 1, opps.size() );

    opps = OpportunityAlertController.getOpportunities(0, 'Prospecting', true);
    System.assertEquals( 0, opps.size() );

}

@IsTest
public static void testGetOpptyWithOpenTasks() {

    Opportunity oppty = new
        Opportunity(Name = 'Test Oppty',
        CloseDate = Date.today(),
        StageName = 'Prospecting'
    );
    insert oppty;

    Task tsk = new Task(
        Subject = 'Test
        Task', WhatId =
        oppty.Id, Status =
        'Not Started'

    );
    insert tsk;

```

```
List<Opportunity>
```

```
opps;
```

```
opps = OpportunityAlertController.getOpportunities(0, 'Prospecting', false);  
System.assertEquals( 1, opps.size() );
```

```
opps = OpportunityAlertController.getOpportunities(0, 'Prospecting', true);  
System.assertEquals( 1, opps.size() );
```

```
}
```

```
}
```

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 theCountry) {  
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); / remove  
        spacereturn parkSvc.byCountry(theCountry);  
    }  
}
```

```

    }
}

```

ParkLocatorTest:

```

@Test
private class ParkLocatorTest {
    @Test static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock
        ());String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'MackinacNational Park', 'Yosemite'};
        System.assertEquals(parks, result);
    }
}

```

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'};
    privateString[] apex_schema_type_info = new String[]{'http:/
parks.services/',false,false'}; privateString[] field_order_type_info = new
String[]{'return_x'};
    }
    public class
    byCountry {
    publicString arg0;
    private String[] arg0_type_info = new String[]{'arg0','http:/
parks.services/',null,'0','1','false'}; privateString[] apex_schema_type_info = new
String[]{'http:/ parks.services/',false,false'}; privateString[] field_order_type_info = new
String[]{'arg0'};

```

```

}
public class ParksImplPort {
    publicString endpoint_x = 'https:/ th-apex-soap-service.herokuapp.com/service/parks';
    public Map<String,String> inputHttpHeaders_x;
    publicMap<String,String>
    outputHttpHeaders_x; public
    StringclientCertName_x;
    public String clientCert_x;
    publicString clientCertPasswd_x;
    public Integertimeout_x;
    privateString[] 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) {
        / start - specify the response you want to send
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
        'Yosemite'};
        / end
        response.put('response_x', response_x);
    }

}
```

PropertyController:

```
public with sharing class PropertyController {
    private static final Decimal DEFAULT_MAX_PRICE = 99999999;
    private static final Integer DEFAULT_PAGE_SIZE = 9;

    /**
    1. Endpoint that retrieves a paged and filtered list of properties
```

2. @param searchKey String used for searching on property title,city and tags
3. @param maxPrice Maximumprice
4. @param minBedrooms Minimumnumber of bedrooms
5. @param minBathrooms Minimum number of bathrooms
6. @param pageSize Number of properties per page
7. @param pageNumber Page number
8. @return PagedResult objectholding the pagedand filtered list of properties

*/

@AuraEnabled(cacheable=true)

e)

public static PagedResult getPagedPropertyList(

String searchKey,

Decimal maxPrice,

Integer

minBedrooms,Integer

er minBathrooms,

Integer pageSize,

Integer pageNumber

) {

/ Normalize inputs

DecimalsafeMaxPrice = (maxPrice == null

? DEFAULT_MAX_PRICE

: maxPrice);

Integer safeMinBedrooms = (minBedrooms == null ? 0 : minBedrooms);

Integer safeMinBathrooms = (minBathrooms == null ? 0 : minBathrooms);

IntegersafePageSize = (pageSize == null

? DEFAULT_PAGE_SIZE

: pageSize);

IntegersafePageNumber = (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 Tagsc LIKE :searchPattern)
    AND Price_c <= :safeMaxPrice
    AND Beds_c >= :safeMinBedrooms
    AND Bathsc >= :safeMinBathrooms
];
result.records
= [SELECT
    Id,
    Address_
    c,Cityc,
    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 Bathsc >= :safeMinBathrooms
WITH SECURITY_ENFORCED
ORDER BY Price_c
LIMIT
:safePageSize
OFFSET :offset
];
return result;

```

```
}
```

```
/**
```

9. Endpoint that retrieves pictures associated with a property

10. @param propertyId Property Id

11. @return List of ContentVersion holding the pictures

```
*/
```

```
@AuraEnabled(cacheable=true)
```

```
e)
```

```
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 numOfContacts,
StringlastName){
        List<Contact> contacts = new List<Contact>();

        for(Integer i=0;i<numOfContacts;i++) {
            Contact c = new Contact(FirstName='Test ' + i, LastName=lastName);
            contacts.add(c);
        }
        system.debug(contacts);
        return contacts;
    }
}

```

RestrictContactByName:

```

trigger RestrictContactByName on Contact (beforeinsert, before update){

    / check contactsprior to insertor update for invalid
dataFor (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();i
        nsertContacts();
    }

    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:

```

@Test
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;

    / Insertmock 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 contactswere
created. ');
```

```
}
```

```
}
```

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);
```

```
}  
}
```

Verify Date:

```
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
```

```
@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 implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

```

/ Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
 / The callout's JSON response returns the equipment records that you upsert in Salesforce.

```

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into
runWarehouseEquipmentSync'); Http http = new
Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() ==
            200){List<Object> jsonResponse
            =
            (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            / class maps the following fields:

```

/ warehouse SKU will be external ID for identifying which equipment records to update within Salesforce

```
for (Object jR : jsonResponse){
    Map<String,Object> mapJson= (Map<String,Object>)jR;
    Product2 product2= new Product2();
    / replacement part (always true),
    product2.Replacement_Part_c = (Boolean) mapJson.get('replacement');
    / cost
    product2.Cost_c = (Integer) mapJson.get('cost');
    / current inventory
    product2.Current_Inventory_c = (Double) mapJson.get('quantity');
    / lifespan
    product2.Lifespan_Months_c = (Integer) mapJson.get('lifespan');
    / maintenance cycle
    product2.Maintenance_Cycle_c = (Integer) mapJson.get('maintenanceperiod');
    / warehouseSKU
    product2.Warehouse_SKU_c = (String) mapJson.get('sku');

    product2.Name
        = (String)
    mapJson.get('name');
    product2.ProductCode = (String) mapJson.get('_id');
    product2List.add(product2);
}

if (product2List.size() >
    0){upsertproduct2List;
    System.debug('Your equipment was synced with the warehouse one');
}
}
}
```

```
public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
```

```
runWarehouseEquipmentSync();
```

```

        System.debug('end
        runWarehouseEquipmentSync');
    }

}

```

WarehouseCalloutServiceMock:

```

@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    / implementhttp mock callout
    global static HttpResponse respond(HttpRequestrequest) {

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

        response.setBody('{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name
        ": "Generator 1000
        kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b6
        11 100aaf742","replacement":true,"quantity":183,"name":"Cooling
        Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
        af743","replacement":true,"quantity":143,"name":"Fuse
        20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}

```

WarehouseCalloutServiceTest:

```

@IsTest
private class WarehouseCalloutServiceTest {
    / implement your mock callout test here
    @isTest

```

```

static void testWarehouseCallout() {
    test.startTest();
    test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.execute(null);
    test.stopTest();

    List<Product2> product2List = new List<Product2>();
    product2List = [SELECT ProductCode FROM Product2];

    System.assertEquals(3, product2List.size());
    System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
    System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
    System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
}
}

```

WarehouseSyncSchedule:

```

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

```

WarehouseSyncScheduleTest:

```

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

```

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