SuperBadge

```
@isTest
private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
 recordId +'/contacts' ;
      request.httpMethod = 'GET';
      RestContext.request = request;
      // Call the method to test
      Account thisAccount = AccountManager.getAccount();
      // Verify results
      System.assert(thisAccount != null);
      System.assertEquals('Test record', thisAccount.Name);

   }

   // Helper method
      static Id createTestRecord() {
      // Create test record
      Account TestAcc = new Account(
        Name='Test record');
      insert TestAcc;
      Contact TestCon= new Contact(
      LastName='Test',
      AccountId = TestAcc.id);
      return TestAcc.Id
;
   }
}



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

```
global class AccountManager {
   @HttpGet
   global static Account getAccount() {public class ParkLocator {
   public static string[] country(string theCountry) {
      ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove
space
      return parkSvc.byCountry(theCountry);
   }
}
      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;
   }
}


@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;
   }
}


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');
    }

}


@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;

        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';
```

```
        AddPrimaryContact apc = new AddPrimaryContact(co, state);
        Test.startTest();
         System.enqueueJob(apc);
        Test.stopTest();
    }
}


public class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext bc) {
      // collect the batches of records or objects to be passed to execute
       return Database.getQueryLocator([Select LeadSource From Lead ]);
   }
   public void execute(Database.BatchableContext bc, List<Lead> leads){
       // process each batch of records
         for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
         }
      update leads;
   }
   public void finish(Database.BatchableContext bc){
    }

}


//Generated by wsdl2apex

public class AsyncParksServices {
   public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
      public String[] getValue() {
         parksServices.byCountryResponse response =
(parksServices.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/',
'parksServices'};
        public AsyncParksServices.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            parksServices.byCountry request_x = new parksServices.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParksServices.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParksServices.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'parksServices.byCountryResponse'}
            );
        }
    }


public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
          account.Number_of_Contacts__c = [Select count() from Contact Where AccountId
```

```
=: account.Id];
            System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
        update updatedAccounts;
    }


}
```

```
//Generated by wsdl2apex

public class AsyncParkServices {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkServices.byCountryResponse response =
(ParkServices.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/',
'ParkServices'};
        public AsyncParkServices.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkServices.byCountry request_x = new ParkServices.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkServices.byCountryResponseFuture)
```

```
System.WebServiceCallout.beginInvoke(
            this,
            request_x,
            AsyncParkServices.byCountryResponseFuture.class,
            continuation,
            new String[]{endpoint_x,
            '',
            'http://parks.services/',
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkServices.byCountryResponse'}
          );
      }
   }
}



@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);
    }
}


@isTest
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                        lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
Inc.', Status='Open - Not Contacted'));
                }
                insert lList;

                Test.startTest();
                String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
        }
}



@isTest
public class LeadProcessorTest {

        @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company
 ='demo'+counter;
            leads.add(lead);
```

```
        }
        insert leads;
    }

    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }

}
```

```
public class DailyLeadProcessor implements Schedulable  {
    Public void execute(SchedulableContext SC){
       List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
        for(Lead l:LeadObj){
            l.LeadSource='Dreamforce';
            update l;
        }
    }
}
```

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

        Contact c = new Contact();
```

```
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id
;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id
;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }

}



//Generated by wsdl2apex

public class AsyncParksService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParksService.byCountryResponse response =
(ParksService.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/',
'ParksService'};
        public AsyncParksService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParksService.byCountry request_x = new ParksService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParksService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParksService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParksService.byCountryResponse'}
            );
        }
    }
}




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

```
//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;
      }
   }
}



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


@isTest
private class ParkLocatorTest {
   @isTest 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', 'Mackinac National Park',
'Yosemite'};
```

```
        System.assertEquals(parks, result);
    }
}



public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
        List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName
from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
        List<Contact> lstContact = new List<Contact>();
        for (Account acc:ListAccount)
        {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId =  acc.id
;
            lstContact.add( cont );
        }

        if(lstContact.size() >0 )
        {
            insert lstContact;
        }

    }

}
```

```
public class AccountProcessor {
@future
   public static void countContacts(List<Id> AccountIds){

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

      List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account
Where Id in :accountIds];

      For(Account acc:accounts){
         List<Contact> contactList = acc.Contacts;
         acc.Number_of_Contacts__c = contactList.size();
         accountsToUpdate.add(acc);
      }
      update accountsToUpdate;
   }
}



@IsTest
public class ContactsTodayControllerTest {

   @IsTest
   public static void testGetContactsForToday() {

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

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

```
    Task tsk = new Task(
        Subject = 'Test Task',
        WhoId = c.Id,
        Status = 'Not Started'
    );
    insert tsk;

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

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

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

}

@IsTest
public static void testGetNoContactsForToday() {

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

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

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

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

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

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

    }

}
```

```
public class ContactsTodayController {

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

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

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

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

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

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

    return contacts;
  }

}


@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.');
  }
}



@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;
}

@isTest
static void testGetPagedPropertyList() {
    Profile standardUserProfile = [
        SELECT Name, Id
        FROM Profile
        WHERE
            UserType = 'Standard'
            AND PermissionsPrivacyDataAccess = FALSE
            AND PermissionsSubmitMacrosAllowed = TRUE
            AND PermissionsMassInlineEdit = TRUE
        LIMIT 1
    ];
    User testUser = new User(
        Alias = 'standt',
        Email = 'standarduser@testorg.com',
        EmailEncodingKey = 'UTF-8',
        LastName = 'Testing',
        LanguageLocaleKey = 'en_US',
        LocaleSidKey = 'en_US',
        ProfileId = standardUserProfile.Id,
```

```
        TimeZoneSidKey = 'America/Los_Angeles',
        UserName = 'standarduser@dreamhouse-testorg.com'
    );
    insert testUser;
    PermissionSet ps = [
        SELECT Id
        FROM PermissionSet
        WHERE Name = 'dreamhouse'
    ];
    insert new PermissionSetAssignment(
        AssigneeId = testUser.Id,
        PermissionSetId = ps.Id
    );

    // Insert test properties as admin
    System.runAs(new User(Id = UserInfo.getUserId())) {
        TestPropertyController.createProperties(5);
    }
    // Read properties as test user
    System.runAs(testUser) {
        Test.startTest();
        PagedResult result = PropertyController.getPagedPropertyList(
            '',
            999999,
            0,
            0,
            10,
            1
        );
        Test.stopTest();
        System.assertEquals(5, result.records.size());
    }
}

@isTest
static 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);
}

@isTest
static 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);
    }
}
```

```
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));
    }
}
```

```
}



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 scope='global')
    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 scope='global')
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
        ];
    }
}
```

```
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; }
}
```

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

```
    @isTest
    static void successResponse() {
        // GIVEN
        GeocodingService.GeocodingAddress address = new
GeocodingService.GeocodingAddress();
        address.street = STREET;
        address.city = CITY;
        address.postalcode = POSTAL_CODE;
        address.state = STATE;
        address.country = COUNTRY;

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

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

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

```
    @isTest
    static void blankAddress() {
        // GIVEN
        GeocodingService.GeocodingAddress address = new
GeocodingService.GeocodingAddress();

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

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

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

```
        address.city = CITY;
        address.postalcode = POSTAL_CODE;
        address.state = STATE;
        address.country = COUNTRY;

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

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

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

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

```
        return res;
      }
    }

    public class OpenStreetMapHttpCalloutMockImplError implements HttpCalloutMock {
      public HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setStatusCode(400);
        return res;
      }
    }
}
```

```
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;
    }
}

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

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt, String
lastname){
        List<Contact> Contacts =new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            Contacts.add(cnt);
        }
        return contacts;
    }
}
```