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

**AccountAddressTrigger**:

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

**AccountManager:**

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {

   @HttpGet
   global static Account getAccount(){
      RestRequest request = RestContext.request;
      string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
      Account result = [SELECT Id, Name, (Select Id, 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(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;
    }
}
```

**AccountProcessorTest:**

```
@isTest
private class AccountProcessorTest {
            @isTest
    private static void  testCountContacts(){
        Account newAccount = new Account(Name='TestAccount');
```

```
        insert newAccount;

        Contact newContact1 = new
Contact(FirstName='John',LastName='Doe',AccountId=newAccount.Id);
        insert newContact1;

        Contact newContact2 = new
Contact(FirstName='John',LastName='Doe',AccountId=newAccount.Id);
        insert newContact2;

        list<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
}
```

**AddPrimaryContact:**

```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact (Contact con, String state){
        this.con=con;
        this.state=state;
    }
    public void execute(QueueableContext context){
        List<Account> accounts=[Select Id, Name, (Select FirstName, LastName, Id from
contacts)
                                from Account where BillingState=:state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            C.AccountId = acc.Id;
            primaryContacts.add(c);
        }
        if(primaryContacts.size()>0){
```

```
            insert primaryContacts;
        }
    }
}
```

**AddPrimaryContactTest:**

```
@isTest
public class AddPrimaryContactTest {
            static testmethod void testQueueable(){
    List<Account> testAccounts = new List<Account>();
    for(Integer i=0;i<50;i++){
        testAccounts.add (new Account(Name='Account '+i,BillingState='CA'));
    }
    for(Integer j=0;j<50;j++){
        testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));
    }
    insert testAccounts;

    Contact testContact=new Contact(FirstName='John', LastName ='Doe');
    insert testContact;

    AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

    Test.startTest();
    system.enqueueJob(addit);
    Test.stopTest();

        System.assertEquals(50, [Select count() from Contact where accountId in (Select Id
from Account where BillingState='CA')]);
}
}
```

**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:**

```
@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 );
    }
}
```

**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:**

```
@isTest
private class AnimalsCalloutsTest {
    @isTest static  void testGetCallout() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null,result, 'The callout returned a null response.');
        // Verify status code
        System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
        // Verify content type
        System.assertEquals('application/json;charset=UTF-8',
          result.getHeader('Content-Type'),
          'The content type value is not expected.');
        // Verify the array contains 3 items
        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
        System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
    }
}
```

**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.setStatusCode(200);
        return response;
    }
}
```

**AsyncParkService:**

```
public class AsyncParkService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
```

```
            'http://parks.services/',
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
      }
   }
}
```

**ClosedOpportunityTrigger:**

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> tasklist = new List<Task>();
   for(Opportunity opp : trigger.New) {
      if(opp.StageName == 'Closed Won'){
         tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
      }
   }
   if(tasklist.size()>0){
      insert tasklist;
   }
}
```

**ContactsTodayController:**

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

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

      Set<Id> contactIds = new Set<Id>();
      for(Task tsk : my_tasks) {
         contactIds.add(tsk.WhoId);
      }
```

```
        for(Event evt : my_events) {
            contactIds.add(evt.WhoId);
        }
        for(Case cse : my_cases) {
            contactIds.add(cse.ContactId);
        }

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

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

        return contacts;
    }

}
```

**ContactsTodayControllerTest:**

```
@IsTest
public class ContactsTodayControllerTest {
```

```
@IsTest
public static void testGetContactsForToday() {

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

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

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

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

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

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

}

@IsTest
public static void testGetNoContactsForToday() {

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

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

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

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

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

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

    }

}
```

**CreateDefaultData:**

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default
data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c      customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

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

        updateCustomSetting(true);
    }
```

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

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

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

```
    }

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

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

    }
}
```

**CreateDefaultDataTest:**

```
@isTest
private class CreateDefaultDataTest {
    @isTest
```

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

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

    }

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

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

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

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

    }}
```

**DailyLeadProcessor:**

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext sc){
      List<Lead> lstofLead = [SELECT Id FROM Lead WHERE Leadsource = null LIMIT
200];
      List<Lead> lstofupdatedLead=new List<Lead>();
      if(!lstofLead.isEmpty()){
        for (Lead ld:lstofLead){
            ld.Leadsource='Dreamforce';
            lstofupdatedLead.add(ld);
        }
         UPDATE lstofupdatedLead;
        }
      }
  }
```

**DailyLeadProcessorTest:**

```
@isTest
private class DailyLeadProcessorTest{
   @testSetup
        static void setup(){
                List<Lead> lstofLead = new List<Lead>();
                for(Integer i = 1; i <= 200; i++){
                Lead 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 Leadsource = null LIMIT
200];
                        system.assertEquals(200, lstoflead.size());
```

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

**GeocodingService:**

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

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

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

            Coordinates coords = new Coordinates();
            if (geocodingUrl != BASE_URL) {
                Http http = new Http();
                HttpRequest request = new HttpRequest();
                request.setEndpoint(geocodingUrl);
```

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

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

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

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

```
        @InvocableVariable
        public Decimal lon;
    }
}
```

**GeocodingServiceTest:**

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

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

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

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

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

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

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

        // THEN
        System.assert(
            computedCoordinates.size() == 1,
            'Expected 1 pair of coordinates were returned'
        );
        System.assert(
```

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

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

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

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

```
            'Expected null lat was returned'
        );
        System.assert(
            computedCoordinates[0].lon == null,
            'Expected null lon was returned'
        );
    }

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

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

**LeadProcessor:**

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

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }
    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new=new List<lead>();
```

```
        for (lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
    system.debug('count = '+  count);
          }
}
```

**LeadProcessorTest:**

```
@isTest
public class LeadProcessorTest{

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

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

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

**MaintenanceRequest:**

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

```
    }
}
```

**MaintenanceRequestHelper:**

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }

        //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
        //create a new maintenance request for a future routine checkup.
        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 static Vehicle__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__c
               from case
               where status ='New'];

    Equipment_Maintenance_Item__c workPart = [select id
                                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.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
  }

  @isTest
  private static void testNegative(){
    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 workP =
createEquipmentMaintenanceItem(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 equipmentMaintenanceItem = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != 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> equipmentMaintenanceItemList = 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++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.
get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
           cs.Status = 'Closed';
           oldCaseIds.add(cs.Id);
        }
        update caseList;
        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);
    }
}
```

**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
space
      return parkSvc.byCountry(theCountry);
   }
}
```

**ParkLocatorTest:**

```
@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);
   }
}
```

**ParkService:**

```
public class ParkService {
   public class byCountryResponse {
      public String[] return_x;
      private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
```

```
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
```

```
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
          response_x = response_map_x.get('response_x');
          return response_x.return_x;
      }
   }
}
```

**ParkServiceMock:**

```
@isTest
global class ParkServiceMock implements WebServiceMock {
  global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {

      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};

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

**PropertyController:**

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

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

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

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

    PagedResult result = new PagedResult();
    result.pageSize = safePageSize;
    result.pageNumber = safePageNumber;
    result.totalItemCount = [
        SELECT COUNT()
        FROM Property__c
```

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

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

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

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

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

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

**RandomContactFactory:**

```
public class RandomContactFactory {

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

   }
}
```

**RestrictContactByName:**

```
trigger RestrictContactByName on Contact (before insert, before update) {

        //check contacts prior to insert or update for invalid data
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {       //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                }
        }
}
```

**SampleDataController:**

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

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

```
}

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

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

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

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

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

**TestPropertyController:**

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

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

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

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

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

    System.assertEquals(null, items);
}

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


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

```
        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
private 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());
    }
}
```

**TestSampleDataController:**

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

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

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

**TestVerifyDate:**

```
@isTest
private class TestVerifyDate {
   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),
date.parse('01/05/2022'));
      System.assertEquals(date.parse('01/05/2022'), D);
  }
   @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),
date.parse('05/05/2022'));
      System.assertEquals(date.parse('01/31/2022'), D);
  }
   @isTest static void Test_DatesWithin30Days_case1(){
                        boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('12/30/2022'));
         System.assertEquals(false , flag);
  }
   @isTest static void Test_DatesWithin30Days_case2(){
```

```
                boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('02/02/2022'));
        System.assertEquals(false , flag);
  }
    @isTest static void Test_DatesWithin30Days_case3(){
                boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('01/15/2022'));
        System.assertEquals(true , flag);
  }
   @isTest static void Test_SetEndOfMonthDate(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
   }
}
```

**VerifyDate:**

```
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {

                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }


        @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {

        if( date2 < date1) { return false; }

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

        @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';

   @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());



         for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();

            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

            product2.Cost__c = (Integer) mapJson.get('cost');
```

```
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');

            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');

            product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');

            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

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

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

**WarehouseCalloutServiceMock:**

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request) {
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226
```

```
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","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 {
          @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(new WarehouseCalloutService());
    }
}
```

**WarehouseSyncScheduleTest:**

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
      @isTest static void test() {
      String scheduleTime = '00 00 00 * * ? *';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
      CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
      System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

      Test.stopTest();
   }
}

//End of Document
```