

```
//CalculatorCalloutMock.apxc
```

```
@isTest
```

```
global class CalculatorCalloutMock 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
```

```
    calculatorServices.doAddResponse response_x =
```

```
        new calculatorServices.doAddResponse();
```

```
    response_x.return_x = 3.0;
```

```
    // end
```

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

```
    }
```

```
}
```

```
//AwesomeCalculatorTest.apxc
```

```
@isTest
```

```
private class AwesomeCalculatorTest {
```

```
    @isTest static void testCallout() {
```

```
        // This causes a fake response to be generated
```

```
        Test.setMock(WebServiceMock.class, new CalculatorCalloutMock());
```

```
        // Call the method that invokes a callout
```

```
        Double x = 1.0;
```

```
        Double y = 2.0;
```

```
        Double result = AwesomeCalculator.add(x, y);
```

```
        // Verify that a fake result is returned
```

```
        System.assertEquals(3.0, result);
```

```
    }
```

```
}
```

```
//ParkLocatorTest.apxc
```

```
@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);
```

```
    }
```

```
}
```

```
//ParkServiceMock.apxc
```

```
@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);
```

```
}  
}
```

```
//AnimalLocatorMock.apxc
```

```
@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;
```

```
    }
```

```
}
```

```
//ParkService.apxc
```

```
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;
        }
    }
}

```

```
//AsyncParkService.apxc
```

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

```
}  
}
```

```
//ParkLocator.apxc
```

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

```
//calculatorServices.apxc
```

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

```

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

```

```

        private String[] field_order_type_info = new String[]{'arg0','arg1'};
    }
    public class doSubtractResponse {
        public Double return_x;
        private String[] return_x_type_info = new
String[]{'return','http://calculator.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://calculator.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class doMultiplyResponse {
        public Double return_x;
        private String[] return_x_type_info = new
String[]{'return','http://calculator.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://calculator.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class CalculatorImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/calculator';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://calculator.services/',
'calculatorServices'};
        public Double doDivide(Double arg0,Double arg1) {
            calculatorServices.doDivide request_x = new calculatorServices.doDivide();
            request_x.arg0 = arg0;
            request_x.arg1 = arg1;
            calculatorServices.doDivideResponse response_x;
            Map<String, calculatorServices.doDivideResponse> response_map_x = new
Map<String, calculatorServices.doDivideResponse>();
            response_map_x.put('response_x', response_x);

```



```

WebServiceCallout.invoke(
    this,
    request_x,
    response_map_x,
    new String[]{endpoint_x,
        ",
        'http://calculator.services/',
        'doDivide',
        'http://calculator.services/',
        'doDivideResponse',
        'calculatorServices.doDivideResponse'}
    );
response_x = response_map_x.get('response_x');
return response_x.return_x;
}

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

```

```

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

    public Double doAdd(Double arg0, Double arg1) {
        calculatorServices.doAdd request_x = new calculatorServices.doAdd();
        request_x.arg0 = arg0;
        request_x.arg1 = arg1;
        calculatorServices.doAddResponse response_x;
        Map<String, calculatorServices.doAddResponse> response_map_x = new
Map<String, calculatorServices.doAddResponse>();
        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
            this,
            request_x,
            response_map_x,
            new String[]{endpoint_x,

```

```

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

```

//AsyncCalculatorServices.apxc

```

public class AsyncCalculatorServices {
    public class doDivideResponseFuture extends System.WebServiceCalloutFuture {
        public Double getValue() {
            calculatorServices.doDivideResponse response =
(calculatorServices.doDivideResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class doSubtractResponseFuture extends System.WebServiceCalloutFuture {
        public Double getValue() {
            calculatorServices.doSubtractResponse response =
(calculatorServices.doSubtractResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class doMultiplyResponseFuture extends System.WebServiceCalloutFuture {
        public Double getValue() {
            calculatorServices.doMultiplyResponse response =
(calculatorServices.doMultiplyResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
}

```

```

public class doAddResponseFuture extends System.WebServiceCalloutFuture {
    public Double getValue() {
        calculatorServices.doAddResponse response =
(calculatorServices.doAddResponse)System.WebServiceCallout.endInvoke(this);
        return response.return_x;
    }
}

public class AsyncCalculatorImplPort {
    public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/calculator';
    public Map<String,String> inputHttpHeaders_x;
    public String clientCertName_x;
    public Integer timeout_x;
    private String[] ns_map_type_info = new String[]{'http://calculator.services/',
'calculatorServices'};
    public AsyncCalculatorServices.doDivideResponseFuture
beginDoDivide(System.Continuation continuation,Double arg0,Double arg1) {
        calculatorServices.doDivide request_x = new calculatorServices.doDivide();
        request_x.arg0 = arg0;
        request_x.arg1 = arg1;
        return (AsyncCalculatorServices.doDivideResponseFuture)
System.WebServiceCallout.beginInvoke(
        this,
        request_x,
        AsyncCalculatorServices.doDivideResponseFuture.class,
        continuation,
        new String[]{endpoint_x,
        ",
        'http://calculator.services/',
        'doDivide',
        'http://calculator.services/',
        'doDivideResponse',
        'calculatorServices.doDivideResponse'}
        );
    }

    public AsyncCalculatorServices.doSubtractResponseFuture
beginDoSubtract(System.Continuation continuation,Double arg0,Double arg1) {

```

```

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

```



```
}
```

```
//PropertyController.apxc
```

```
public with sharing class PropertyController {  
    @AuraEnabled(cacheable=true)  
    public static Property__c[] getPropertyList(  
        String searchKey,  
        Decimal maxPrice,  
        Integer minBedrooms,  
        Integer minBathrooms  
    ){  
        String key = '%' + searchKey + '%';  
        return [  
            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  
                (title__c LIKE :key  
                OR city__c LIKE :key  
                OR tags__c LIKE :key)  
                AND price__c <= :maxPrice  
                AND beds__c >= :minBedrooms  
                AND baths__c >= :minBathrooms  
            ORDER BY price__c  
            LIMIT 100  
        ];  
    }  
}
```

```
}
```

```
@AuraEnabled(cacheable=true)
public static PagedResult getPagedPropertyList(
    String searchKey,
    Decimal maxPrice,
    Integer minBedrooms,
    Integer minBathrooms,
    Integer pageSize,
    Integer pageNumber
) {
    maxPrice = Decimal.valueOf(maxPrice + "");
    minBedrooms = Integer.valueOf(minBedrooms + "");
    minBathrooms = Integer.valueOf(minBathrooms + "");
    pageSize = Integer.valueOf(pageSize + "");
    pageNumber = Integer.valueOf(pageNumber + "");

    Integer pSize = (Integer) pageSize;
    String key = '%' + searchKey + '%';
    Integer offset = ((Integer) pageNumber - 1) * pSize;
    PagedResult result = new PagedResult();
    result.pageSize = pSize;
    result.pageNumber = (Integer) pageNumber;
    result.totalItemCount = [
        SELECT COUNT()
        FROM property__c
        WHERE
            (title__c LIKE :key
            OR city__c LIKE :key
            OR tags__c LIKE :key)
            AND price__c <= :maxPrice
            AND beds__c >= :minBedrooms
            AND baths__c >= :minBathrooms
    ];
    result.records = [
        SELECT
            Id,
```



```

        address__c,
        city__c,
        state__c,
        description__c,
        price__c,
        baths__c,
        beds__c,
        thumbnail__c
    FROM property__c
    WHERE
        (title__c LIKE :key
        OR city__c LIKE :key
        OR tags__c LIKE :key)
        AND price__c <= :maxPrice
        AND beds__c >= :minBedrooms
        AND baths__c >= :minBathrooms
    ORDER BY price__c
    LIMIT :pSize
    OFFSET :offset
];
return result;
}

```

```

@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')
    ];

    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
    ORDER BY CreatedDate
];
}
}

```

//SampleDataController.apxc

```

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

            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;

            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
    );
    insert properties;
} catch (Exception e) {
    System.debug(e);
}
}
}

```

//TestPropertyController.apxc

```

@isTest
public class TestPropertyController {
    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(
                    Title__c = 'Name ' + i,
                    Price__c = 20000,
                    Beds__c = 3,
                    Baths__c = 3
                )
            );
        }
        insert properties;
    }

    static testMethod void testGetPropertyList() {
        TestPropertyController.createProperties(5);
        Test.startTest();
        Property__c[] properties = PropertyController.getPropertyList(
    
```

```

        ",
        9999999,
        0,
        0
    );
    Test.stopTest();
    System.assertEquals(5, properties.size());
}

static testMethod void testGetPagedPropertyList() {
    TestPropertyController.createProperties(5);
    Test.startTest();
    PagedResult result = PropertyController.getPagedPropertyList(
        ",
        9999999,
        0,
        0,
        10,
        1
    );
    Test.stopTest();
    System.assertEquals(5, result.records.size());
}

static testMethod void testGetPictures() {
    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);
}
}

//OpportunityAlertController

```

```

public class OpportunityAlertController {

    @AuraEnabled
    public static List<Opportunity> getOpportunities(Decimal daysSinceLastModified,
String oppStage, Boolean hasOpen) {
        DateTime lastModifiedDateFilter =
DateTime.now().addDays((Integer)daysSinceLastModified * -1);
        List<Opportunity> opportunities = [
            SELECT Id, Name, StageName, LastModifiedDate, CloseDate
            FROM Opportunity
            WHERE StageName = :oppStage AND LastModifiedDate <=
:lastModifiedDateFilter
        ];
        Map<Id,Opportunity> oppMap = new Map<Id,Opportunity>(opportunities);
        if(hasOpen == true) {
            List<Task> tasks = [SELECT ID, WhatId FROM TASK WHERE IsClosed = false AND
WhatId IN :oppMap.keySet()];
            List<Opportunity> opps_with_tasks = new List<Opportunity>();
            for(Task ta : tasks) {
                if(oppMap.containsKey(ta.WhatId)) {
                    opps_with_tasks.add(oppMap.get(ta.WhatId));
                }
            }
            opportunities = opps_with_tasks;
        }
        return opportunities;
    }
}

```

```

//OpportunityAlertControllerTest

```

```

@Test
public class OpportunityAlertControllerTest {

    @Test
    public static void testGetOpptyWithoutOpenTasks() {

```

```

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

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

    List<Opportunity> opps;

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

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

}

```

```

@IsTest
public static void testGetOpptyWithOpenTasks() {

```

```

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

```

```

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

```

```

);
insert tsk;

List<Opportunity> opps;

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

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

}

}

//ContactsTodayController

public class ContactsTodayController {

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

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

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

```

```
}
```

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

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

```
    return contacts;  
}
```

```
}
```

```
//ContactsTodayControllerTest
```

```
@IsTest
```

```
public class ContactsTodayControllerTest {
```

```
    @IsTest
```

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

```
        Account acct = new Account(
```



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

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

```
    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());

    }

}

//AnimalsCalloutsTest

@Test
private class AnimalsCalloutsTest {
    @Test 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.');
```

}

```

    @Test
    static void testPostCallout() {
```

```

// Set mock callout class
Test.setMock(HttpCalloutMock.class, new AnimalsHttpCalloutMock());
// This causes a fake response to be sent
// from the class that implements HttpCalloutMock.
HttpResponse response = AnimalsCallouts.makePostCallout();
// Verify that the response received contains fake values
String contentType = response.getHeader('Content-Type');
System.assert(contentType == 'application/json');
String actualValue = response.getBody();
System.debug(response.getBody());
String expectedValue = '{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}';
System.assertEquals(expectedValue, actualValue);
System.assertEquals(200, response.getStatusCode());
}
}

```

//AnimalLocatorTest

```

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

```

//AnimalsHttpCalloutMock

```

@Test
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
    }
}

```

```

    HttpResponse response = new HttpResponse();
    response.setHeader('Content-Type', 'application/json');
    response.setBody("{\"animals\": [\"majestic badger\", \"fluffy bunny\", \"scary bear\",
\"chicken\", \"mighty moose\"]}");
    response.getStatusCode(200);
    return response;
}
}

```

//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.getStatusCode());
} else {
    System.debug(response.getBody());
}
return response;
}
}

```

//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');
    }
}
}

```

//CaseManager

```

@RestResource(urlMapping='/Cases/*')
global with sharing class CaseManager {

```

```

@HttpGet
global static Case getCaseById() {
    RestRequest request = RestContext.request;
    // grab the caseId from the end of the URL
    String caseId = request.requestURI.substring(
        request.requestURI.lastIndexOf('/')+1);
    Case result = [SELECT CaseNumber,Subject,Status,Origin,Priority
        FROM Case
        WHERE Id = :caseId];
    return result;
}

@HttpPost
global static ID createCase(String subject, String status,
    String origin, String priority) {
    Case thisCase = new Case(
        Subject=subject,
        Status=status,
        Origin=origin,
        Priority=priority);
    insert thisCase;
    return thisCase.Id;
}

@HttpDelete
global static void deleteCase() {
    RestRequest request = RestContext.request;
    String caseId = request.requestURI.substring(
        request.requestURI.lastIndexOf('/')+1);
    Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
    delete thisCase;
}

@HttpPut
global static ID upsertCase(String subject, String status,
    String origin, String priority, String id) {
    Case thisCase = new Case(
        Id=id,
        Subject=subject,
        Status=status,

```

```

        Origin=origin,
        Priority=priority);
    // Match case by Id, if present.
    // Otherwise, create new case.
    upsert thisCase;
    // Return the case ID.
    return thisCase.Id;
}

@HttpPatch
global static ID updateCaseFields() {
    RestRequest request = RestContext.request;
    String caseId = request.requestURI.substring(
        request.requestURI.lastIndexOf('/')+1);
    Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
    // Deserialize the JSON string into name-value pairs
    Map<String, Object> params = (Map<String,
Object>)JSON.deserializeUntyped(request.requestbody.toString());
    // Iterate through each parameter field and value
    for(String fieldName : params.keySet()) {
        // Set the field and value on the Case sObject
        thisCase.put(fieldName, params.get(fieldName));
    }
    update thisCase;
    return thisCase.Id;
}
}

//AccountManager

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

```



```
        return result;
    }
}
```

```
//AccountManagerTest
```

```
@isTest
```

```
private class AccountManagerTest {
```

```
    @isTest static void testGetContactsByAccountId() {
```

```
        Id recordId = createTestRecord();
```

```
        // Set up a test request
```

```
        RestRequest request = new RestRequest();
```

```
        request.requestUri =
```

```
'https://yourInstance.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 accountTest = new Account(
            Name='Test record');
```

```
        insert accountTest;
```

```
        Contact contactTest= new Contact(
```

```
            FirstName='John',
```

```
            LastName='Doe',
```

```
            AccountId = accountTest.Id
```

```
        );
```

```
        insert contactTest;
```

```

        return accountTest.Id;
    }
}

```

//CaseManagerTest

@IsTest

```

private class CaseManagerTest {
    @isTest static void testGetCaseById() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
            + recordId;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Case thisCase = CaseManager.getCaseById();
        // Verify results
        System.assert(thisCase != null);
        System.assertEquals('Test record', thisCase.Subject);
    }

    @isTest static void testCreateCase() {
        // Call the method to test
        ID thisCaseId = CaseManager.createCase(
            'Ferocious chipmunk', 'New', 'Phone', 'Low');
        // Verify results
        System.assert(thisCaseId != null);
        Case thisCase = [SELECT Id,Subject FROM Case WHERE Id=:thisCaseId];
        System.assert(thisCase != null);
        System.assertEquals(thisCase.Subject, 'Ferocious chipmunk');
    }

    @isTest static void testDeleteCase() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
    }
}

```

```

request.requestUri =
    'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
    + recordId;
request.httpMethod = 'DELETE';
RestContext.request = request;
// Call the method to test
CaseManager.deleteCase();
// Verify record is deleted
List<Case> cases = [SELECT Id FROM Case WHERE Id=:recordId];
System.assert(cases.size() == 0);
}

@isTest static void testUpsertCase() {
    // 1. Insert new record
    ID case1Id = CaseManager.upsertCase(
        'Ferocious chipmunk', 'New', 'Phone', 'Low', null);
    // Verify new record was created
    System.assert(case1Id != null);
    Case case1 = [SELECT Id,Subject FROM Case WHERE Id=:case1Id];
    System.assert(case1 != null);
    System.assertEquals(case1.Subject, 'Ferocious chipmunk');
    // 2. Update status of existing record to Working
    ID case2Id = CaseManager.upsertCase(
        'Ferocious chipmunk', 'Working', 'Phone', 'Low', case1Id);
    // Verify record was updated
    System.assertEquals(case1Id, case2Id);
    Case case2 = [SELECT Id,Status FROM Case WHERE Id=:case2Id];
    System.assert(case2 != null);
    System.assertEquals(case2.Status, 'Working');
}

@isTest static void testUpdateCaseFields() {
    Id recordId = createTestRecord();
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
        + recordId;
    request.httpMethod = 'PATCH';
    request.addHeader('Content-Type', 'application/json');

```

```

request.requestBody = Blob.valueOf({'status': "Working"});
RestContext.request = request;
// Update status of existing record to Working
ID thisCaseId = CaseManager.updateCaseFields();
// Verify record was updated
System.assert(thisCaseId != null);
Case thisCase = [SELECT Id,Status FROM Case WHERE Id=:thisCaseId];
System.assert(thisCase != null);
System.assertEquals(thisCase.Status, 'Working');
}
// Helper method
static Id createTestRecord() {
    // Create test record
    Case caseTest = new Case(
        Subject='Test record',
        Status='New',
        Origin='Phone',
        Priority='Medium');
    insert caseTest;
    return caseTest.Id;
}
}

```

//AccountProcessorTest

@isTest

```

public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
        Account a = new Account();
        a.Name = 'Test Account';
        insert a;
        Contact con = new Contact();
        con.FirstName = 'Binary';
        con.LastName = 'Programming';
        con.AccountId = a.Id;
        insert con;
        List<Id> accListId= new List<Id>();
    }
}

```

```

        accListId.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();
        Account acc=[select Number_Of_Contacts__c from Account where Id=: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }

}

```

//AccountProcessor

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accList = [SELECT Id, Number_Of_Contacts__c, (select Id from
Contacts) from Account where Id in: accountIds];
        for(Account acc : accList){
            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }
        update accList;
    }

}

```

//ContactsListWithController

```

public class ContactsListWithController {
    // Controller code goes here
    private String sortOrder = 'LastName';
    public List<Contact> getContacts() {
        List<Contact> results = Database.query(
            'SELECT Id, FirstName, LastName, Title, Email ' +
            'FROM Contact ' +
            'ORDER BY ' + sortOrder + ' ASC ' +
            'LIMIT 10'
        );
        return results;
    }
}

```

```
}  
}
```

//NewCaseListController

```
public class NewCaseListController {  
    public List<Case> getNewCases(){  
        List<Case> results=[SELECT CaseNumber FROM Case Where Status='New'];  
        return results;  
    }  
}
```

//AwesomeCalculator

```
public class AwesomeCalculator {  
    public static Double add(Double x, Double y) {  
        calculatorServices.CalculatorImplPort calculator =  
            new calculatorServices.CalculatorImplPort();  
        return calculator.doAdd(x,y);  
    }  
}
```

//ContactsListWithController.vfp

```
<apex:page controller="ContactsListWithController">  
    <apex:form >  
        <apex:pageBlock title="Contacts List" id="contacts_list">  
            <!-- Contacts List -->  
            <apex:pageBlockTable value="{! contacts }" var="ct">  
                <apex:column value="{! ct.FirstName }"/>  
                <apex:column value="{! ct.LastName }"/>  
                <apex:column value="{! ct.Title }"/>  
                <apex:column value="{! ct.Email }"/>  
            </apex:pageBlockTable>  
        </apex:pageBlock>  
    </apex:form>  
</apex:page>
```

//Create Contact

```
<apex:page standardController="Contact">
  <apex:form >
    <apex:pageBlock title="Create Contact">
      <apex:pageBlockSection columns="1">
        <apex:inputField value="{! Contact.FirstName }"/>
        <apex:inputField value="{! Contact.LastName }"/>
        <apex:inputField value="{! Contact.Email }"/>
      </apex:pageBlockSection>
    <apex:pageBlockButtons >
      <apex:commandButton action="{! save }" value="Save" />
    </apex:pageBlockButtons>
  </apex:pageBlock>
</apex:form>
</apex:page>
```

//UserStatus

```
<apex:page >
  <apex:pageBlock title="User Status">
    <apex:pageBlockSection columns="1">
      {! $User.FirstName & ' ' & $User.LastName }
      ({! IF($User.isActive, $User.Username, 'inactive') })
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:page>
```

//ContactForm

```
<apex:page standardController="Contact">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Quick Start: Visualforce</title>
    <!-- Import the Design System style sheet -->
    <apex:slds />
```

```

</head>
<body>
    <apex:form >
    <apex:pageBlock title="New Contact">
    <!--Buttons -->
    <apex:pageBlockButtons >
        <apex:commandButton action="{!save}" value="Save"/>
    </apex:pageBlockButtons>
    <!--Input form -->
    <apex:pageBlockSection columns="1">
    <apex:inputField value="{!Contact.Firstname}"/>
    <apex:inputField value="{!Contact.Lastname}"/>
    <apex:inputField value="{!Contact.Email}"/>
    </apex:pageBlockSection>
    </apex:pageBlock>
    </apex:form>
</body>

```

```

</apex:page>

```

```

//DisplayImage

```

```

<apex:page showHeader="false">
    <apex:image alt="Image" url="https://developer.salesforce.com/files/salesforce-developer-network-logo.png"/>
</apex:page>

```

```

//AccountSummary

```

```

<apex:page standardController="Account">
    <apex:pageBlock title="Account Summary">
    <apex:pageBlockSection >
        Account owner: {! Account.Owner.Name } <br/>
        Name: {! Account.Name } <br/>
        Phone: {! Account.Phone } <br/>
        Industry: {! Account.Industry } <br/>
        Revenue: {! Account.AnnualRevenue } <br/>
    </apex:pageBlockSection>
    </apex:pageBlock>
    </apex:page>

```



```
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

//AccountDetail

```
<apex:page standardController="Account">
    <apex:outputField value="{! Account.Name }"/>
        <apex:outputField value="{! Account.Phone }"/>
        <apex:outputField value="{! Account.Industry }"/>
        <apex:outputField value="{! Account.AnnualRevenue }"/>
    <apex:relatedList list="Contacts"/>
        <apex:relatedList list="Opportunities" pageSize="5"/>
</apex:page>
```

//OppView

```
<apex:page standardController="Opportunity">
    <apex:pageBlock title="Opportunity Details">
        <apex:pageBlockSection >
            <apex:outputField value="{! Opportunity.Name }"/>
            <apex:outputField value="{! Opportunity.Amount }"/>
            <apex:outputField value="{! Opportunity.CloseDate }"/>
            <apex:outputField value="{! Opportunity.Account.Name }"/>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

//AccountEdit

```
<apex:page standardController="Account">
    <apex:form >
    <apex:pageBlock title="Edit Account">
        <apex:pageMessages />
        <apex:pageBlockSection columns="1">
            <apex:inputField value="{! Account.Name }"/>
            <apex:inputField value="{! Account.Phone }"/>
            <apex:inputField value="{! Account.Industry }"/>
        </apex:pageBlockSection>
    </apex:pageBlock>
    </apex:form>
</apex:page>
```

```

        <apex:inputField value="{! Account.AnnualRevenue }"/>
    </apex:pageBlockSection>
    <apex:pageBlockButtons >
        <apex:commandButton action="{! save }" value="Save" />
    </apex:pageBlockButtons>
</apex:pageBlock>
<apex:pageBlock title="Contacts">
<apex:pageBlockTable value="{!Account.contacts}" var="contact">
    <apex:column >
        <apex:outputLink value="{! URLFOR($Action.Contact.Edit, contact.Id) }">
            Edit
        </apex:outputLink>
        &nbsp;
        <apex:outputLink value="{! URLFOR($Action.Contact.Delete, contact.Id) }">
            Del
        </apex:outputLink>
    </apex:column>
    <apex:column value="{!contact.Name}"/>
    <apex:column value="{!contact.Title}"/>
    <apex:column value="{!contact.Phone}"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>

```

//AccountList

```

<apex:page standardController="Account" recordSetVar="accounts">
    <apex:repeat var="a" value="{! accounts }">
        <li>
            <apex:outputLink value="/{!a.ID}">
                {a.Name}
            </apex:outputLink>
        </li>
    </apex:repeat>
</apex:page>

```

```
//HelloJQuery
```

```
<apex:page >
  <!-- Add the static resource to page's <head> -->
  <apex:includeScript value="{! $Resource.jQuery }"/>
  <!-- A short bit of jQuery to test it's there -->
  <script type="text/javascript">
    jQuery.noConflict();
    jQuery(document).ready(function() {
      jQuery("#message").html("Hello from jQuery!");
    });
  </script>
  <!-- Where the jQuery message will appear -->
  <h1 id="message"></h1>
</apex:page>
```

```
//jQueryMobileResources
```

```
<apex:page showHeader="false" sidebar="false" standardStylesheets="false">
  <!-- Add static resources to page's <head> -->
  <apex:stylesheet value="{!
    URLFOR($Resource.jQueryMobile,'jquery/jquery.mobile-1.4.5.css')}" />
  <apex:includeScript value="{! $Resource.jQueryMobile }"/>
  <apex:includeScript value="{!
    URLFOR($Resource.jQueryMobile,'jquery/jquery.mobile-1.4.5.js')}" />
  <div style="margin-left: auto; margin-right: auto; width: 50%">
    <!-- Display images directly referenced in a static resource -->
    <h3>Images</h3>
    <p>A hidden message:
      <apex:image alt="eye" title="eye"
        url="{!URLFOR($Resource.jQueryMobile, 'jquery/images/icons-png/eye-
black.png')}" />
      <apex:image alt="heart" title="heart"
        url="{!URLFOR($Resource.jQueryMobile, 'jquery/images/icons-png/heart-
black.png')}" />
      <apex:image alt="cloud" title="cloud"
        url="{!URLFOR($Resource.jQueryMobile, 'jquery/images/icons-png/cloud-
```

```
black.png'))"/>
    </p>
    <!-- Display images referenced by CSS styles, all from a static resource. -->
    <h3>Background Images on Buttons</h3>
    <button class="ui-btn ui-shadow ui-corner-all
        ui-btn-icon-left ui-icon-action">action</button>
    <button class="ui-btn ui-shadow ui-corner-all
        ui-btn-icon-left ui-icon-star">star</button>
    </div>
</apex:page>
```

//NewCaseList

```
<apex:page controller="NewCaseListController" >
    <apex:pageBlock title="New Cases">
        <apex:repeat var="Case" value="{!newCases}">
            <apex:outputLink value="/{!Case.ID}">{!Case.CaseNumber}</apex:outputLink>
        </apex:repeat>
    </apex:pageBlock>
</apex:page>
```

//HelloWorld

```
<apex:page sidebar="false" showHeader="false">
    <h1>Hello World</h1>
    <apex:pageBlock title="A Block Title">
        <apex:pageBlockSection title="A Section Title">
            I'm three components deep!
        </apex:pageBlockSection>
        <apex:pageBlockSection title="A New Section">
            This is another section.
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

//ContactList

```

<apex:page standardController="Contact" recordSetVar="contacts">
  <apex:form >
    <apex:pageBlock title="Contacts List" id="contacts_list">
      Filter:
      <apex:selectList value="{! filterId }" size="1">
        <apex:selectOptions value="{! listViewOptions }"/>
        <apex:actionSupport event="onchange" reRender="contacts_list"/>
      </apex:selectList>
      <!-- Contacts List -->
      <apex:pageBlockTable value="{! contacts }" var="ct">
        <!-- Pagination -->
<table style="width: 100%"><tr>
  <td>
    Page: <apex:outputText value=" {!PageNumber} of {! CEILING(ResultSize /
PageSize) }"/>
  </td>
  <td align="center">
    <!-- Previous page -->
    <!-- active -->
    <apex:commandLink action="{! Previous }" value="« Previous"
      rendered="{! HasPrevious }"/>
    <!-- inactive (no earlier pages) -->
    <apex:outputText style="color: #ccc;" value="« Previous"
      rendered="{! NOT(HasPrevious) }"/>
    &nbsp;&nbsp;&nbsp;
    <!-- Next page -->
    <!-- active -->
    <apex:commandLink action="{! Next }" value="Next »"
      rendered="{! HasNext }"/>
    <!-- inactive (no more pages) -->
    <apex:outputText style="color: #ccc;" value="Next »"
      rendered="{! NOT(HasNext) }"/>
  </td>
  <td align="right">
    Records per page:
    <apex:selectList value="{! PageSize }" size="1">
      <apex:selectOption itemValue="5" itemLabel="5"/>

```

```

    <apex:selectOption itemValue="20" itemLabel="20"/>
    <apex:actionSupport event="onchange" reRender="contacts_list"/>
</apex:selectList>
</td>
</tr></table>
    <apex:column value="{! ct.FirstName }"/>
    <apex:column value="{! ct.LastName }"/>
    <apex:column value="{! ct.Email }"/>
    <apex:column value="{! ct.Account.Name }"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>

```

//DisplayUserInfo

```

<apex:page >
    {! $User.FirstName }
</apex:page>

```

//ContactView

```

<apex:page standardController="Contact">
    <apex:pageBlock title="Contact View">
        <apex:pageBlockSection >
            First Name: {! Contact.FirstName }
            Last Name: {! Contact.LastName }
            Contact owner: {! Contact.Owner.Email }
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>

```

//ShowImage

```

<apex:page >
    <apex:image url="{!URLFOR($Resource.vfimagetest,'cats/kitten1.jpg')}" />
</apex:page>

```

```
//DailyLeadProcessorTest.apxc
```

```
@isTest
```

```
private class DailyLeadProcessorTest{
```

```
//Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
```

```
public static String CRON_EXP = '0 0 0 2 6 ? 2022';
```

```
static testmethod void testScheduledJob(){
```

```
List<Lead> leads = new List<Lead>();
```

```
for(Integer i = 0; i < 200; i++){
```

```
Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company  
' + i, Status = 'Open - Not Contacted');
```

```
leads.add(lead);
```

```
}
```

```
insert leads;
```

```
Test.startTest();
```

```
// Schedule the test job
```

```
String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new  
DailyLeadProcessor());
```

```
// Stopping the test will run the job synchronously
```

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

```
}
```

```
}
```

```
//LeadProcessorTest.apxc
```

```
@isTest
```

```
public class LeadProcessorTest {
```

```
@testSetup
```

```
static void setup() {
```

```
List<Lead> leads = new List<Lead>();
```

```
// insert 200 leads
```

```
for (Integer i=0;i<200;i++) {
```

```
leads.add(new Lead(LastName='Lead '+i,
```

```
Company='Lead', Status='Open - Not Contacted')));  
}  
insert leads;  
}
```

```
static testmethod void test() {
```

```
Test.startTest();
```

```
LeadProcessor lp = new LeadProcessor();
```

```
Id batchId = Database.executeBatch(lp, 200);
```

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

```
// after the testing stops, assert records were updated properly
```

```
System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
```

```
}
```

```
}
```

```
//AddPrimaryContactTest.apxc
```

```
@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();  
}  
}
```

```
//AddPrimaryContact
```

```
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;  
        }  
  
    }  
}
```

```
}
```

```
//AccountProcessor
```

```
public class AccountProcessor {
```

```
    @future
```

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

```
        List<Account> accList = [SELECT Id, Number_Of_Contacts__c, (select Id from  
Contacts) from Account where Id in: accountIds];
```

```
        for(Account acc : accList){
```

```
            acc.Number_Of_Contacts__c = acc.Contacts.size();
```

```
        }
```

```
        update accList;
```

```
    }
```

```
}
```

```
//DailyLeadProcessor
```

```
global class DailyLeadProcessor implements Schedulable{
```

```
    global void execute(SchedulableContext ctx){
```

```
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = "];
```

```
        if(leads.size() > 0){
```

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

```
            for(Lead lead : leads){
```

```
                lead.LeadSource = 'DreamForce';
```

```
                newLeads.add(lead);
```

```
            }
```

```
            update newLeads;
```

```
        }
```

```
    }
```

```
}
```

```
//AccountProcessorTest
```

```

@isTest
public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
        Account a = new Account();
        a.Name = 'Test Account';
        insert a;
        Contact con = new Contact();
        con.FirstName = 'Binary';
        con.LastName = 'Programming';
        con.AccountId = a.Id;
        insert con;
        List<Id> accListId= new List<Id>();
        accListId.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();
        Account acc=[select Number_Of_Contacts__c from Account where Id=: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }
}

```

//LeadProcessor

```

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

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

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

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

```

```

for (Lead lead : scope) {

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

}
update leads;
}

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

}
}

//AccountManager

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

//AccountManagerTest

@isTest
private class AccountManagerTest {

    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
    }
}

```

```

        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.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 accountTest = new Account(
            Name='Test record');
        insert accountTest;
        Contact contactTest= new Contact(
            FirstName='John',
            LastName='Doe',
            AccountId = accountTest.Id
        );
        insert contactTest;
        return accountTest.Id;
    }
}

//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

```

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

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

@Test

```



```

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');

}
}

```

//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);
}
}
```

```
//WarehouseCalloutService
```

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    //Write a class that makes a REST callout to an external warehouse system to get a
list of equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in
Salesforce.
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
        }
    }
}
```

```

for (Object jR : jsonResponse){
    Map<String,Object> mapJson = (Map<String,Object>)jR;
    Product2 product2 = new Product2();
    //replacement part (always true),
    product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
    //cost
    product2.Cost__c = (Integer) mapJson.get('cost');
    //current inventory
    product2.Current_Inventory__c = (Double) mapJson.get('quantity');
    //lifespan
    product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
    //maintenance cycle
    product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
    //warehouse SKU
    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

```

@Test
global class WarehouseCalloutServiceMock implements HttpCalloutMock {

```



```

// implement http mock callout
global static HttpResponseMessage respond(HttpRequest request) {
    HttpResponseMessage response = new HttpResponseMessage();
    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": "55d66226726b611100aaf742", "replacement": true, "quantity": 183, "name": "Cooling Fan", "maintenanceperiod": 0, "lifespan": 0, "cost": 300, "sku": "100004"}, {"_id": "55d66226726b611100aaf743", "replacement": true, "quantity": 143, "name": "Fuse 20A", "maintenanceperiod": 0, "lifespan": 0, "cost": 22, "sku": "100005"}]");
    response.StatusCode = 200;
    return response;
}
}

```

//WarehouseCalloutServiceTest

```

[Test]
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    [Test]
    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());  
    }  
}
```

```
//WarehouseSyncScheduleTes
```

```
@isTest  
public with sharing class WarehouseSyncScheduleTest {  
    // implement scheduled code here  
    //  
    @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();  
    }  
}
```

```
//MaintenanceRequest.apxt
```

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