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

```
@IsTest
```

```
public class OpportunityAlertControllerTest {

    @IsTest
    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, 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());

  }

}
```

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');
  } }
```

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 );
   }
}
```

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 testing( ) {
      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() ;
   }

}
```

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') ] ) ;

    }
}
```

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;
    }
}
```

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;
    }
}
```

DailyLeadProcessor

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

DailyLeadProcessorTest

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

      }
      insert lList ;

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

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);
    }
}
```

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 );
    }
}
```

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) {
      // 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);
  }
}
```

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'}
            );
        }
    }
}
```

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') ] ) ;

    }
}
```

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;
    }
}
```

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;
    }
}
```

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.');
    }
}
```

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);
    }
}
```

MaintenanceRequest

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

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

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
  // implement http mock callout
  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 {
    // implement your mock callout test here
          @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

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

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

WarehouseSyncSchedule

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
```

```
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

WarehouseSyncScheduleTest

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