AccountManager:-

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

AccountManagerTest:-

```
@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

      // Call the method to test
      Account  acc = AccountManager.getAccount();

      // Verify results
      System.assert(acc != null);
   }
```

```
        private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

AccountProcessor:-

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

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

AccountProcessorTest:-

```
@IsTest
```

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

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

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

      Test.startTest();
         AccountProcessor.countContacts(setAccId);
      Test.stopTest();

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

}
```

AddPrimaryContact:-

```
public class AddPrimaryContact implements Queueable{
   Contact con;
   String state;
```

```
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest:-

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
```

```
        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();

        System.assertEquals(50, [select count() from Contact]);
    }
}
```

AnimalLocator:-

```
public class AnimalLocator
{

  public static String getAnimalNameById(Integer id)
   {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
         String strResp = '';
          system.debug('******response '+response.getStatusCode());
          system.debug('******response '+response.getBody());
        // 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
        Map<string,object> animals = (map<string,object>) results.get('animal');
         System.debug('Received the following animals:' + animals );
         strResp = string.valueof(animals.get('name'));
         System.debug('strResp >>>>>>' + strResp );
      }
      return strResp ;
  }

}
```

AnimalLocatorMock:-

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
       HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');
      response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
      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);
    }
}
```

ContactController:-

```
public with sharing class ContactController {

    public Contact c { get; set; }

    public List<Contact> samepage { get; set; }

    public ContactController(){
      c=new Contact();
    }

    public PageReference save() {
      insert c;
     samepage= [select id,FirstName,LastName,Email,Birthdate from Contact where id=:c.id];

       return null;
    }
  }
```

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!');

    }
}
```

LeadProcessorTest:-

```
@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']);
    }
}
```

NewCaseListController:-

```
public class NewCaseListController {
    private String val = 'New';
public List<Case> getNewCases() {
        List<Case> results = Database.query(
        'SELECT Id, CaseNumber FROM Case WHERE Status = \'' + String.escapeSingleQuotes(val)+'\'');
    return results;
}
}
```

ParkLocator:-

```
public class ParkLocator {
```

```
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorTest:-

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

ParkService:-

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-
1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
```

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

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

ParkServiceMock:-

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {
      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
      List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = lstOfDummyParks;

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

RandomContactFactory:-

```
public class RandomContactFactory {
```

```
   public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
      List<Contact> contactList = new List<Contact>();

      for(Integer i=0;i<numContactsToGenerate;i++) {
         Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
         contactList.add(c);
         System.debug(c);
      }
      //insert contactList;
      System.debug(contactList.size());
      return contactList;
   }

}
```

TestRestrictContactByName:-

```
@isTest
private class TestRestrictContactByName {

   @isTest static void testInvalidName() {
      //try inserting a Contact with INVALIDNAME
      Contact myConact = new Contact(LastName='INVALIDNAME');
      insert myConact;

      // Perform test
      Test.startTest();
      Database.SaveResult result = Database.insert(myConact, false);
      Test.stopTest();
      // Verify
      // In this case the creation should have been stopped by the trigger,
```

```
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                            result.getErrors()[0].getMessage());

    }
}
```

TestVerifyDate:-

```
@isTest
private class TestVerifyDate {

    //testing that if date2 is within 30 days of date1, should return date 2
    @isTest static void testDate2within30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 04, 11);
        System.assertEquals(testDate,resultDate);
    }

    //testing that date2 is before date1. Should return "false"
    @isTest static void testDate2beforeDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 02, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }

    //Test date2 is outside 30 days of date1. Should return end of month.
```

```
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

VerifyDate:-

```
public class VerifyDate {

        //method to handle potential checks against two dates
        public static Date CheckDates(Date date1, Date date2) {
                //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of
the month
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }

        //method to check if date2 is within the next 30 days of date1
        private static Boolean DateWithin30Days(Date date1, Date date2) {
                //check for date2 being in the past
        if( date2 < date1) { return false; }

        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1
                if( date2 >= date30Days ) { return false; }
```

```
            else { return true; }
        }

        //method to return the end of the month of a given date
        private static Date SetEndOfMonthDate(Date date1) {
            Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
            Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
            return lastDay;
        }

}
```