salesforce Developer project codes

Account Address Trigger:

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

}
```

closed opportunity trigger:

```
trigger ClosedOpportunitytrigger on Opportunity (after insert,after update) {
   List<Task> oppList = new List<Task>();
   for (opportunity a : [SELECT Id,stageName,(SELECT WhatId,subject FROM
Tasks)FROM Opportunity
                  WHERE Id IN :Trigger.New AND StageName LIKE '%Closed won%']){
      oppList.add(new Task(WhatId=a.Id, subject='Follow up test task'));


   }
   if (oppList.size()>0){
      insert oppList;
   }
}
```

Test verify date :

```
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;
        }

}
```

TEST VERIFY DATE :

```
@isTest
public class TestVerifyDate {

   @isTest static void test1(){
      Date d =
verifyDate.checkDates(Date.parse('01/01/2022'),Date.parse('01/03/2022'));
      system.assertEquals(Date.parse('01/03/2022'), d);
   }

   @isTest static void test2(){
      Date d =
verifyDate.checkDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
      system.assertEquals(Date.parse('01/31/2022'), d);
```

```
    }
}
```

TEST RESTRICT CONTACT BY NAME:

```
@isTest
public class TestRestrictContactByName {

    @isTest
    public static void testContact(){
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        dATABASE.SaveResult res = Database.insert(ct,false);
        system.assertEquals('the Last Name "INVALIDNAME" is not allowed for
DML',res.getErrors()[0].getMessage());
    }

}
```

RANDOM CONTACT FACTORY :

```
public class RandomContactFactory {

    public static List<contact> generateRandomContacts(Integer num,string lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i = 1;i<=num;i++){
            Contact ct = new Contact(FirstName = 'Test'+i,LastName = lastName);
            ContactList.add(ct);
            }
        return ContactList;
    }

}
```

ACCOUNT PROCESSOR :

```
public without sharing 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;
    }
}
```

ACCOUNT PROCESSOR TEST:

```
@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);

    }
}
```

LEAD PROCESSOR TEST :

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

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

   public void execute (Database.BatchableContext bc, List<Lead> l_lst){
      List<lead> l_lst_new = new List<lead>();
      for(lead l : l_lst){
         l.leadsource = 'Dreamforce';
         l_lst_new.add(l);
         Count +=1;
      }
      update l_lst_new;
   }

   public void finish(Database.BatchableContext bc){
      system.debug('count = ' + count);
   }


}
```

DAILY LEAD PROCESSOR TEST :

```
@isTest
private class DailyLeadProcessorTest {
   @testsetup
   static void setup(){
      List<lead> lstOfLead = new List<lead>();
      for(Integer i = 1; i<=200; i++){
         Lead ld = new Lead(Company ='comp' + i, LastName = 'LN' + i ,status ='working -
contacted');
         lstOfLead.add(ld);
      }

      insert lstOfLead;
   }
```

```
    static testmethod void testDailyLeadProcessorScheduledJob(){
        string sch = '0 5 12 * * ?';
        Test.startTest();
        string jobId = system.Schedule('scheduledApexTest',sch, new
DailyLeadProcessor());

        List<Lead> lstOfLead = [SELECT iD from lEAD where LeadSource = null limit 200];
        system.assertEquals(200,lstOfLead.size());

        Test.stopTest();
    }
}
```

ADD PRIMARY CONTACT :

```
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;
        }
            }

}
```

DAILY LEAD PROCESSOR :

```
global class DailyLeadProcessor implements schedulable {
    global void execute(schedulableContext sc){
```

```
    List<Lead> lstofLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT
200];

    list<Lead> lstofupdatedLead = new List<Lead>();
    if(!LstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
        ld.LeadSource = 'Dreamforce';
        lstOfUpdatedLead.add(ld);
      }

      update lstOfupdatedLead;
    }
  }

}
```

ANIMAL LOCATOR :

```
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 (response.getStatusCode() == 200)
    {
      Map<string, Object> results = (Map<string, Object>)
JSON.deserializeUntyped(response.getBody());
      Map<string, Object> animals = (Map<string, object>) results.get('animal');
      system.debug('Recieved the following animals:' + animals);
      strResp = string.valueof(animals.get('name'));
      system.debug('strResp >' + strResp);
    }
    return strResp;
```

```
    }

}
```

ANIMAL LOCATOR TEST :

```
@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":"cow","eats":"grass"}}');
        response.setStatusCode(200);
        return response;
    }

}
```

PARK LOCATOR TEST :

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

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

PARK LOCATOR :

```
@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);
    }
}
```

PARK SERVICE :

```
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
```

```
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ACCOUNT MANAGER :

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        system.debug(accountId);
        Account objAccount = [SELECT Id, Name,(SELECT Id, Name FROM Contacts) FROM
Account WHERE
```

```
                    Id = : accountId LIMIT 1];
        return objAccount;
    }
}
```

ACCOUNT MANAGER TEST :

```
@isTest
public class AccountManagerTest {
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'Test Contact');
        insert objAccount;

        Contact objContact = new Contact(LastName = 'Test Contact', AccountId =
objAccount.Id);
        insert objContact;


        Id recordId = objAccount.Id;
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://empathetic-badger-rmquep-dev-
ed.my.salesforce.com/services/apexrest/Accounts/' +
            recordId + '/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        Account thisAccount = AccountManager.getAccount();

        system.assert(thisAccount != null);
        system.assertEquals('Test Account', thisAccount.Name);
    }

}
```

ANIMAL LOCATOR :

```
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 (response.getStatusCode() == 200)
        {
            Map<string, Object> results = (Map<string, Object>)
JSON.deserializeUntyped(response.getBody());
            Map<string, Object> animals = (Map<string, object>) results.get('animal');
            system.debug('Recieved the following animals:' + animals);
            strResp = string.valueof(animals.get('name'));
            system.debug('strResp >' + strResp);
        }
        return strResp;

    }

}
```

ASYNC PARK SERVICE :

```
//Generated by wsdl2apex

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

```
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}
```