APPEX SUPERBADGE

Kanmanoori Kurumurthy

[COMPANY NAME] [Company address]

Get Started with Apex Trigger

AccountAddressTrigger.apxt

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

Bulk Apex Triggers

ClosedOpportunityTrigger.apxt

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

Appex Testing

Get Started With Apex Unit test

VerifyDate.apxc

```
public class VerifyDate {
   public static Date CheckDates(Date date1, Date date2) {
                  if (DateWithin30Days(date1, date2)) {
                        return date2;
```

```
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }
        @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1)
       {
         return false;
       }
         Date date30Days = date1.addDays(30);
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }
        @TestVisible private static Date SetEndOfMonthDate(Date date1) {
                Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
                Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
                return lastDay;
        }
}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
      Date D=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'),D);
   }
    @isTest static void Test_CheckDates_case2(){
      Date D=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
```

```
        System.assertEquals(date.parse('01/31/2020'),D);
    }
     @isTest static void Test_DateWithin30Days_case1(){
          Booleanflag=
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/2019'));
          System.assertEquals(false,flag);
       }
         @isTest static void Test_DateWithin30Days_case2(){
          Booleanflag=
VerifyDate.DateWithin30Days(date.parse('02/02/2020'),date.parse('12/30/2020'));
          System.assertEquals(false,flag);
         }
            @isTest static void Test_DateWithin30Days_case3(){
          Booleanflag=
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('1/15/2020'));
          System.assertEquals(true,flag);
}
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate=VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}
```

Test Apex Triggers

RestrictContactByName.apxt

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';
```

```
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt,false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name"INVALIDNAME"not allowed for DML',
result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Tests

RandomContactFactory.apxc

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){

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

ASYNCHRONUS APEX

Use Future Methods

AccountProcessor.apxc

```
public class AccountProcessor {
@future
    public static void countContacts(List<Id> accountsIds){
```

```
        List<Account> accountsToUpdate =new List<Account>();


        List<Account> accounts = [Select Id, Name, (Select Id from Contacts)from Account
Where Id in :accountsIds];


        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

AccountProcessorTest.apxc

```
@IsTest
public class AccountProcessorTest {
@IsTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;


        contact newContact1 = new Contact(FirstName='john',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact1;
        contact newContact2 = new Contact(FirstName='john',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;
        List<Id> accountIds=new List<Id>();
        accountIds.add(newAccount.Id);
```

```
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
}
```

Uses Batch Apex

LeadProcessor.apxc

```
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.apxc

```
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();
        for (Integer i=0;i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' +i;
            L.Company ='Company';
            L.Status= 'Random Status';
            L_list.add(L);
               }
        insert L_list;
        Test.startTest();
        LeadProcessor lp =new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apcx

```
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;
        }
    }
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apcx

```
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.apxc

```
@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 ='Guru' , LastName= 'Murthi');
        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')]);
    }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx){
        List<lead> leadstoupdate = new List<lead>();
        List<Lead> leads = [Select Id from Lead Where LeadSource = NULL Limit 200];
        for(Lead l:leads){
            l.LeadSource ='Dreamforce';
            leadstoupdate.add(l);
        }
       update leadstoupdate;
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 6 ? 2022';
    static testmethod void testScheduledJob(){
        List<Lead> leads= new List<lead>();
```

```
        for(Integer i=0;i<200;i++){
            Lead l= new Lead (FirstName = 'First' +i,
                              LAstName= 'LastName',
                              Company = 'The Inc'
                              );
            leads.add(l);
        }
        insert leads;
    Test.startTest();
    String jobId = System.schedule('scheduledAppexTest',CRON_EXP,new
DailyLeadProcessor());
     Test.stopTest();
    List<Lead>checkleads = new List<Lead>();
    checkleads = [Select id From Lead Where LeadSource= 'Dreamforce'and Company = 'The
Inc'];

        System.assertEquals(200,checkleads.size(),'Leads were not created');
    }
}
```

APEX INTEGRATION SERVICES

AnimalLocator.apxc

```
public class AnimalLocator {
public class cls_animal {
public Integer id;
public String name;
public String eats;
public String says;
}
public class JSONOutput{
public cls_animal animal;
```

```
}
}

    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);
        system.debug('response: ' + response.getBody());
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }
}
```

AnimalLocatorTest.apxc

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}
```

AnimalLocatorMock.apxc

```
@IsTest
```

```
global class AnimalLocatorMock implements HttpCalloutMock {


    global HTTPresponse respond(HTTPrequest request) {

        Httpresponse response = new Httpresponse();

        response.setStatusCode(200);

        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');

        return response;

    }

}
```

Apex SOAP Callouts

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;
        }
      }
    }
```

ParkLocator.apxc

```
    public class ParkLocator {

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

ParkLocatorTest.apxc

```
    @isTest
    private class ParkLocatorTest {
      @isTest
      static void testParkLocator() {
```

```
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

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

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) {
        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);
      }
    }
```

AsyncParkServices.apxc

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

Apex Web Services

AccountManager.apxc

```
@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.apxc

```
@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;
    }
}
```