Apex Specialist - SuperBadges codes

1.Apex Triggers:

Get Strarted with Apex Triggers :

AccountAddressTrigger

```
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 :

```
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;
}}
```

Apex Testing :

Get Started with Apex Unit Test:

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
        @TestVisible 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
  @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:

```
@isTest
public 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(){
       Boolean flag= VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/2019'));
       System.assertEquals(false , flag);
    }
    @isTest static void Test_DateWithin30Days_case2(){
       Boolean flag= VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2019'));
       System.assertEquals(false , flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
       Boolean flag= VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('01/15/2020'));
       System.assertEquals(true , flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
```

```
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

   }
}
```

Test Apex Triggers:

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert, before update) {

          //check contacts prior to insert or update for invalid data
          For (Contact c : Trigger.New) {
                    if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid
                              c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
       } } }
```

TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {

   @isTest public static void Test_insertupdateContact(){
      Contact ct = new Contact();
      ct.LastName = 'INVALIDNAME' ;
      Test.startTest();
      Database.SaveResult res = Database.insert(ct,false);
      Test.stopTest();
     System.assert(!res.isSuccess());
     System.assert(res.getErrors().size()>0);
     System.assertEquals('The Last Name  "INVALIDNAME"  is not allowed for DML' ,
res.getErrors()[0].getMessage());
   }
}
```

Create a Test Data for Apex Tests:

random contact factory

```
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;
    }
}
```

Asychronous Apex:

Use Future methods:

Account processor

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id>  accountIds){
        List<Account> accountsToUpdate = new List<Account>();
        List<Account> accounts = [Select Id , Name , (Select Id from Contacts)from Account where Id in
:accountIds];
        for(Account acc: accounts){
            List<Contact> contactlist = acc.Contacts;
            acc.Number_of_Contacts__c=contactlist.size();
            accountsToUpdate.add(acc);
     }
        Update accountsToUpdate;
    }

}
```

Account processor Test :

```
@IsTest
private 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= 'jane', LastName='Doe' , AccountId =
newAccount.Id);
        insert newContact2;
```

```
        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();


    }
}
```

Use Batch Apex Unit :

Lead processor :

```
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);
    }
}
```

Lead Proceesor test :

```
@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 processors with Queueable apex:

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;
}
}
}
```

Add primary Contact Test:

```
@isTest
public class AddPrimaryContactTest
{
@isTest
static void TestList()
```

```
{
List<Account> Teste = new List <Account>();
for(Integer i=0;i<50;i++)
{
Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
}
for(Integer j=0;j<50;j++)
{
Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
}
insert Teste;

Contact co = new Contact();
co.FirstName='demo';
co.LastName ='demo';
insert co;
String state = 'CA';

AddPrimaryContact apc = new AddPrimaryContact(co, state);
Test.startTest();
System.enqueueJob(apc);
Test.stopTest();
}
}
```

Schedule jobs using the Apex Scheduler:

Daily lead processor:

```
global class DailyLeadProcessor implements Schedulable{
global void execute(SchedulableContext ctx){
List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){
List<Lead> newLeads = new List<Lead>();

for(Lead lead : leads){
lead.LeadSource = 'DreamForce';
newLeads.add(lead);
}

update newLeads;
}
}


}
```

Daily Lead processor Test :

```
@isTest
private class DailyLeadProcessorTest{
//Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
public static String CRON_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){
List<Lead> leads = new List<Lead>();

for(Integer i = 0; i < 200; i++){
Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status =
'Open - Not Contacted');
leads.add(lead);
}

insert leads;

Test.startTest();
// Schedule the test job
String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

// Stopping the test will run the job synchronously
Test.stopTest();
}
}
```

Apex integration services:

Apex Rest Callouts:

Animal Locator:

```
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');
```

```
    }

}
```

Animal Locator Test

```
@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 );
    }
}
```

Animal Locator Mock

```
@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;
    }
}
```

Apex SOAP callouts

Park Locator:

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

Park Locator Mock

```
@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);
  }
}
```

ParkLocator Test

```
@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);
   }
}
```

Apex Web Services

Account Manager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global 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;
    }
}
```

Account Manager Test

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }
    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```