**Apex Triggers**

Challenge-1

Get Started With Apex Triggers

AccountAddressTrigger.apxc:

trigger AccountAddressTrigger on Account (before insert,before update) {

for(Account account:Trigger.New){

if(account.Match\_Billing\_Address\_\_c == True){

account.ShippingPostalCode =account.BillingPostalCode;

}

}

}

Challenge-2

Bulk Apex Triggers

ClosedOpportunityTrigger.apxc:

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**

Challenge-1

Get Started With Apex Unit Tests

VerifyDate.apxc:

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.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(){

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'));

}

}

Challenge-2

Test Apex Triggers

**RestrictContactByName.apxt:**

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.apxc:**

@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" is not allowed for DML',result.getErrors()[0].getMessage());

}

}

Challenge-3

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;

}

}

**Asynchronous Apex**

Challenge-1

Use Future Methods

**AccountProcessor.apxc:**

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;

}

}

**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='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();

}

}

Challenge-2

Use Batch Apex

**LeadProcessor.apxc:**

public class LeadProcessor implements Database.Batchable<sObject> {

public Database.QueryLocator start(Database.BatchableContext bc) {

// collect the batches of records or objects to be passed to execute

return Database.getQueryLocator([Select LeadSource From Lead ]);

}

public void execute(Database.BatchableContext bc, List<Lead> leads){

// process each batch of records

for (Lead Lead : leads) {

lead.LeadSource = 'Dreamforce';

}

update leads;

}

public void finish(Database.BatchableContext bc){

}

}

**LeadProcessorTest.apxc:**

@isTest

public class LeadProcessorTest {

@testSetup

static void setup() {

List<Lead> leads = new List<Lead>();

for(Integer counter=0 ;counter <200;counter++){

Lead lead = new Lead();

lead.FirstName ='FirstName';

lead.LastName ='LastName'+counter;

lead.Company ='demo'+counter;

leads.add(lead);

}

insert leads;

}

@isTest static void test() {

Test.startTest();

LeadProcessor leadProcessor = new LeadProcessor();

Id batchId = Database.executeBatch(leadProcessor);

Test.stopTest();

}

}

Challenge-3

Control Processes With Queueable Apex

**AddPrimaryContact.apxc:**

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;

}

}

}

**AddPrimaryContactTest.apxc:**

@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();

}

}

Challenge-4

Schedule Jobs Using the Apex Scheduler

**DailyLeadProcessor.apxc:**

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.apxc:**

@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());

}

}

**APEX INTEGRATION SERVICES**

Challenge-1

Apex REST Callouts

**AnimalLocator.apxc:**

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.apxc:**

@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.apxc:**

@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 );

}

}

Challenge-2

Apex SOAP Callouts

**ParkLocator.apxc:**

public class ParkLocator {

public static string[] country(string theCountry) {

ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space

return parkSvc.byCountry(theCountry);

}

}

**ParkLocatorTest.apxc:**

@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);

}

}

**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) {

// 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);

}

}

challenge-3

Apex Web Services

**AccountManager.apxc:**

**@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;**

**}**

**}**

**AccountManagerTest.apxc:**

**@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;**

**}**

**}**