APEX SPECIALIST SUPERBADGES

APEX TRIGGERS

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;
 }
 }
}
```

Explanation: AccountAddressTrigger is a apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

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;
}
 }
```

Explanation: ClosedOpportunityTrigger is a apex trigger which fire trigger after inserting or updating an opportunity.

APEX SPECIALIST SUPERBADGES

APEX TESTING

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('02/02/2020'));
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'));

 }

}
```

Explanation: TestVerifyDate is a apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range.

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());  }

 }
```

Explanation: TestRestrictContactByName is a Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. **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;

}

}
```

Explanation:RandomContactFactory is an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name.

Asynchronous Apex   APEX INTEGRATION SERVICES

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<Object,String>)
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);
}
}

ParkLocator.apxc public class ParkLocator { public static string[]
country(String country) { parkService.parksImplPort park = new
parkService.parksImplPort(); return park.byCountry(country);

```
}

}
```

ParkLocatorMock.apxc

```
 @isTest  global class ParkServiceMock implements WebServiceMock
{
 global void doInvoke( Object stub, Object request, Map response, String endpoint, String
soapAction, String requestName, String responseNS, String responseName, String responseType) {

 parkService.byCountryResponse response_x = new parkService.byCountryResponse();
response_x.return_x = new List{'Hamburg Wadden Sea National Park', 'Hainich National Park',
'Bavarian Forest National Park'};

response.put('response_x', response_x);

 }

}
```

ParkLocatorTest.apxc

```
 @isTest   private class

ParkLocatorTest {

@isTest static void testCallout() {

 Test.setMock(WebServiceMock.class, new ParkServiceMock());

String country = 'Germany'; String[] result = ParkLocator.Country(country);
 System.assertEquals(new List{'Hamburg Wadden Sea National Park', 'Hainich National  Park',

'Bavarian Forest National Park'}, result);

 }

}
```

AccountManager.apxc

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

global with sharing class AccountManager {

@HttpGet

global static account getAccount()  {

 RestRequest request = RestContext.request;
String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
request.requestURI.lastIndexOf('/'));
List<Account>a = [select id, name, (select id, name from contacts) from account where id =
:accountId];
 List<contact> co = [select id, name from contact where account.id = :accountId]; system.debug('**
a[0]= '+ a[0]);  return a[0];
}
}
```

AccountManagerTest.apxc

```
@istest  public class
AccountManagerTest {
 @istest static void testGetContactsByAccountId() {
 Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();   request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+
recordId+'/Contacts'; request.httpMethod = 'GET';   RestContext.request = request;
Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount!= null);
System.assertEquals('Test record', thisAccount.Name);
 }
// Helper method   static Id
createTestRecord() {
 // Create test record
Account accountTest = new Account( Name='Test record');
 insert accountTest;
Contact contactTest = new Contact( FirstName='John', LastName='Doe', AccountId=accountTest.Id
);  return accountTest.Id;
}
}
```

Setting Up the Development Org:

.

APEX SPECIALIST SUPERBADGES

1. Create a new Trailhead Playground for Apex Specialist Superbadge.
2. Install the unlocked package (ID:04t6g000008av9iAAA).
3. Add picklist values Repair and Routine Maintenace to the Type field on the Case object.
4. Update the Case page layout assignment to use the Case(HowWeRoll) layout to the profile.
5. Rename the tab/label for the Case tab to Maintenace Request.
6. Click on App launcher and search Create Default Data.
7. Then, Click on Create Data in order to generate sample data to the application.

CHALLENGE 1: Automate Record Creation
CHALLENGE 3: Schedule Synchronization

Go to the Developer Console and type the below code:

WarehouseSyncSchedule.apxc:

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {
   WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

Save it.

- Go to Setup => Search in Quick Find Box => Apex classes => click Schedule Apex and Jb Name = WarehouseSyncScheduleJob, Apex Class = WarehouseSyncSchedule as it is below shown in the image:

<Picture>

CHALLENGE 4: Test Automation Logic CHALLENGE 6: Test Scheduling Logic

Go to the Developer Console and use the below given code:

WarehouseSyncSchedule.apxc: global class WarehouseSyncSchedule implements

```
Schedulable {
   global void execute(SchedulableContext ctx) {

      WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
    }
}
```

WarehouseSyncScheduleTest.apxc:

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());        String
jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX
systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

Run all atleast once.

APEX SPECIALIST SUPERBADGES

APEX SPECIALIST SUPERBADGES