APEX TRIGGERS

1)Get Started with Apex Triggers:-

Create an Apex trigger for Account that matches Shipping Address Postal Code with Billing Address Postal Code based on a custom field.
For this challenge, you need to create a trigger that, before insert or update, checks for a checkbox, and if the checkbox field is true, sets the Shipping Postal Code (whose API name is ShippingPostalCode) to be the same as the Billing Postal Code (BillingPostalCode).
The Apex trigger must be called 'AccountAddressTrigger'.
The Account object will need a new custom checkbox that should have the Field Label 'Match Billing Address' and Field Name of 'Match_Billing_Address'. The resulting API Name should be 'Match_Billing_Address__c'.
With 'AccountAddressTrigger' active, if an Account has a Billing Postal Code and 'Match_Billing_Address__c' is true, the record should have the Shipping Postal Code set to match on insert or update.

Solution:

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; }
} }
```

2)Bulk Apex Triggers :-

Challenge:

Create an Apex trigger for Opportunity that adds a task to any opportunity set to 'Closed Won'.
To complete this challenge, you need to add a trigger for Opportunity. The trigger will add a task to any opportunity inserted or updated with the stage of 'Closed Won'. The task's subject must be 'Follow Up Test Task'.
The Apex trigger must be called 'ClosedOpportunityTrigger'
With 'ClosedOpportunityTrigger' active, if an opportunity is inserted or updated with a stage of 'Closed Won', it will have a task created with the subject 'Follow Up Test Task'.
To associate the task with the opportunity, fill the 'WhatId' field with the opportunity ID.
This challenge specifically tests 200 records in one operation.

Solution:

ClosedOpportunityTrigger:

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

APEX TESTING

1) Get Started with Apex Unit Test:-

Challenge:

Create a unit test for a simple Apex class.
Install a simple Apex class, write unit tests that achieve 100% code coverage for the class, and run your Apex tests.
The Apex class to test is called 'VerifyDate', and the code is available here. Copy and paste this class into your Developer Edition via the Developer Console.
'VerifyDate' is a class which tests if a date is within a proper range, and if not will return a date that occurs at the end of the month within the range.
The unit tests must be in a separate test class called 'TestVerifyDate'.
The unit tests must cover scenarios for all lines of code included in the Apex class, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.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;
} }
```

2.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 return date=VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
}}
```

2)Test Apex Triggers:-

Challenge:

Create a unit test for a simple Apex trigger.
Install a simple Apex trigger, write unit tests that achieves 100% code coverage for the trigger, and run your Apex tests.
The Apex trigger to test is called 'RestrictContactByName', and the code is available here. Copy and paste this trigger into your Developer Edition via the Developer Console.
'RestrictContactByName' is a trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'.
The unit tests must be in a separate Apex class called 'TestRestrictContactByName'.
The unit tests must cover scenarios for all lines of code included in the Apex trigger, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.RestrictContactByName.apxc

```
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') { //invalid name is invalid c.AddError('The Last Name
'"+c.LastName+'" is not allowed for
DML');
 }
} }
```

2.TestRestrictContactByName.apxc

```
@isTest
 public class TestRestrictContactByName { static testMethod void Test()
{
List<Contact> listContact= new List<Contact>();
 Contact c1 = new Contact(FirstName='Raam', LastName='Leela' ,
email='ramleela@test.com');
 Contact c2 = new Contact(FirstName='gatsby', LastName =
'INVALIDNAME',email='gatsby@test.com'); listContact.add(c1);
listContact.add(c2); Test.startTest();
try {
insert listContact; }
catch(Exception e) {
 }
Test.stopTest(); }
}
```

3)Create Test Data for Apex Tests:

Challenge:

Create a contact test factory.

Create an Apex class that returns a list of contacts based on two incoming parameters: one for the number of contacts to generate, and the other for the last name. The list should NOT be inserted into the system, only returned. The first name should be dynamically generated and should be unique for each contact record in the list.

The Apex class must be called 'RandomContactFactory' and be in the public scope.

The Apex class should NOT use the @isTest annotation.

The Apex class must have a public static method called 'generateRandomContacts' (without the @testMethod annotation).

The 'generateRandomContacts' method must accept an integer as the first parameter, and a string as the second. The first parameter controls the number of contacts being generated, the second is the last name of the contacts generated.

The 'generateRandomContacts' method should have a return type of List<Contact>.

The 'generateRandomContacts' method must be capable of consistently generating contacts with unique first names.

For example, the 'generateRandomContacts' might return first names based on iterated number (i.e. 'Test 1','Test 2').

The 'generateRandomContacts' method should not insert the contact records into the database.

Solution:

RandomContactFactory.apxc:

```
public class RandomContactFactory {
 public static List<Contact> generateRandomContacts(Integer num, String lastName){
List<Contact> contacts =new List<Contact>(); for(Integer i=1;i<=num;i++){
Contact s=new Contact(FirstName='Test '+ i, LastName=lastName);
contacts.add(s); }
return contacts; }
}
```

ASYNCHRONOUS APEX

1)Use Future Methods:

Challenge:

Create an Apex class that uses the @future annotation to update Account records.
Create an Apex class with a method using the @future annotation that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class.
Create a field on the Account object called 'Number_of_Contacts__c' of type Number. This field will hold the total number of Contacts for the Account.
Create an Apex class called 'AccountProcessor' that contains a 'countContacts' method that accepts a List of Account IDs. This method must use the @future annotation.
For each Account ID passed to the method, count the number of Contact records associated to it and update the 'Number_of_Contacts__c' field with this value.
Create an Apex test class called 'AccountProcessorTest'.
The unit tests must cover all lines of code included in the AccountProcessor class, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.AccountProcessor.apxc

```
public without sharing class AccountProcessor { @future
public static void countContacts(list<Id> accountIds){
 list<Account> accounts=[select Id, (select Id from Contacts) from Account where Id
```

```
IN: accountIds];
 for(Account act: accounts){
act.Number_of_Contacts__c=act.Contacts.size(); }
update accounts; }
}
```

2.AccountProcessorTest.apxc

```
@isTest
 private class AccountProcessorTest {
@isTest
 private static void countContactsTest(){
list<Account> accounts=new list<Account>(); for(Integer i=0;i<300;i++){
accounts.add(new Account(Name='Test Account' + i)); }
insert accounts;
 list<Contact> contacts=new list<Contact>(); list<Id> accountIds=new List<Id>();
for(Account act : accounts){
contacts.add(new contact(FirstName=act.Name,LastName='Test
Account',AccountId=act.Id));
accountIds.add(act.Id); }
insert contacts; Test.startTest();
AccountProcessor.countContacts(accountIds);
Test.stopTest();
 list<Account> acts=[select Id,Number_of_Contacts__c from Account]; for(Account act:
acts){
system.assertEquals(1,a cc.Number_of_Contacts__c,'ERROR: At least 1 Account
record with incorrect count contact');
} }}
```

2)Use Batch Apex:

Challenge:

Create an Apex class that uses Batch Apex to update Lead records.
Create an Apex class that implements the Database.Batch able interface to update all Lead records in the org with a specific LeadSource. Write unit tests that achieve 100% code coverage for the class.
Create an Apex class called 'LeadProcessor' that uses the Database.Batch able interface.
Use a QueryLocator in the start method to collect all Lead records in the org.

The execute method must update all Lead records in the org with the LeadSource value of 'Dreamforce'.
Create an Apex test class called 'LeadProcessorTest'.
In the test class, insert 200 Lead records, execute the 'LeadProcessor' Batch class and test that all Lead records were updated correctly.
The unit tests must cover all lines of code included in the LeadProcessor class, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.LeadProcessor.apxc

```
public without sharing class LeadProcessor implements Database.Batch able<sObject>
{
public Database.QueryLocator start(Database.BatchableContext Dbc){ return
Database.getQueryLocator([select Id, Name from Lead]);
}
 public void execute(Database.BatchableContext Dbc, List<Lead> leads){
for(Lead l:leads){ l.LeadSource='Dreamforce';
}
update leads; }
public void finish(Database.BatchableContext dbc){ System.debug('Done');
} }
```

2.LeadProcessorTest

```
@isTest
 private class LeadProcessorTest {
@isTest
 private static void testBatchClass() {
// Load test data
 List<Lead> leads = new List<Lead>(); for (Integer i=0; i<200; i++) {
leads.add(new Lead(LastName='Connock', Company='Salesforce')); }
insert leads;
 // Perform the test
 Test.startTest();
 LeadProcessor lp = new LeadProcessor();
 Id batchId = Database.executeBatch(lp, 200); Test.stopTest();
```

```
 // Check the result
List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE LeadSource =
'Dreamforce'];
System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 Lead record not
updated correctly');
} }
```

3)Control Processes with Queueable Apex:-

Challenge:

Create an Queueable Apex class that inserts Contacts for Accounts.
Create a Queueable Apex class that inserts the same Contact for each Account for a specific state. Write unit tests that achieve 100% code coverage for the class.
Create an Apex class called 'AddPrimaryContact' that implements the Queue able interface.
Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation.
The execute method must query for a maximum of 200 Accounts with the BillingState specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject clone() method.
Create an Apex test class called 'AddPrimaryContactTest'.
In the test class, insert 50 Account records for BillingState "NY" and 50 Account records for BillingState "CA". Create an instance of the AddPrimaryContact class, enqueue the job and assert that a Contact record was inserted for each of the 50 Accounts with the BillingState of "CA".
The unit tests must cover all lines of code included in the AddPrimaryContact class, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.AddPrimaryContact.apxc

```
public without sharing class AddPrimaryContact implements Queueable { private
Contact contact;
 private String state;
// Constructor - pass in Contact sObject and State abbreviation as arguments public
AddPrimaryContact(Contact inputContact, String inputState) {
 // Store in class instance variables
this.contact = inputContact;
```

```
this.state = inputState; }
public void execute(QueueableContext context) { //System.debug('Job Id ' +
context.getJobId());
// Retrieve 200 Account records
 List<Account> accounts = [SELECT Id FROM Account WHERE BillingState =
:state LIMIT 200];
 // Create empty list of Contact records
List<Contact> contacts = new List<Contact>(); // Iterate through the Account records
for ( Account acc : accounts) {
// Clone (copy) the Contact record, make the clone a child of the specific Account record
// and add to the list of Contacts
 Contact contactClone = contact.clone(); contactClone.AccountId = acc.Id;
contacts.add(contactClone);
}
 // Add the new Contact records to the database
insert contacts; }
}
```

2.AddPrimaryContactTest.apxc

```
@isTest
 private class AddPrimaryContactTest {
@isTest
 private static void testQueueableClass() {
// Load test data
 List<Account> accounts = new List<Account>(); for (Integer i=0; i<500; i++) {
Account acc = new Account(Name='Test Account'); if ( i<250 ) {
acc.BillingState = 'NY'; } else {
acc.BillingState = 'CA'; }
accounts.add(acc); }
insert accounts;
 Contact contact = new Contact(FirstName='Simon', LastName='Connock'); insert
contact;
 // Perform the test
 Test.startTest();
 Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA')); Test.stopTest();
 // Check the result
```

```
List<Contact> contacts = [SELECT Id FROM Contact WHERE
Contact.Account.BillingState = 'CA'];
System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of Contact
records found');
} }
```

4)Schedule Jobs Using the Apex Scheduler:-

Challenge:

Create an Apex class that uses Scheduled Apex to update Lead records.
Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. Write unit tests that achieve 100% code coverage for the class. This is very similar to what you did for Batch Apex.
Create an Apex class called 'DailyLeadProcessor' that uses the Schedulable interface.
The execute method must find the first 200 Leads with a blank LeadSource field and update them with the LeadSource value of 'Dreamforce'.
Create an Apex test class called 'DailyLeadProcessorTest'.
In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly.
The unit tests must cover all lines of code included in the DailyLeadProcessor class, resulting in 100% code coverage.
Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

Solution:

1.DailyLeadProcessor.apxc

```
public without sharing class DailyLeadProcessor implements Schedulable {
public void execute(SchedulableContext ctx) {
 //System.debug('Context ' + ctx.getTriggerId()); // Returns the ID of the CronTrigger
scheduled job
 // Get 200 Lead records and modify the LeadSource field
List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null
LIMIT 200];
for ( Lead l : leads) { l.LeadSource = 'Dreamforce';
}
 // Update the modified records
update leads; }
```

}

2.DailyLeadProcessorTest.apxc

```
@isTest
 private class DailyLeadProcessorTest {
private static String CRON_EXP = '0 0 0 ? * * *'; // Midnight every day
@isTest
 private static void testSchedulableClass() {
// Load test data
 List<Lead> leads = new List<Lead>(); for (Integer i=0; i<500; i++) {
if ( i < 250 ) {
 leads.add(new Lead(LastName='Connock', Company='Salesforce'));
} else {
 leads.add(new Lead(LastName='Connock', Company='Salesforce',
LeadSource='Other')); }
}
 insert leads;

 // Perform the test
 Test.startTest();
 String jobId = System.schedule('Process Leads', CRON_EXP, new
DailyLeadProcessor()); Test.stopTest();
// Check the result
List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = 'Dreamforce'];
System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 record not updated
correctly');
// Check the scheduled time
List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger
WHERE Id = :jobId];
System.debug('Next Fire Time ' + cts[0].NextFireTime);
// Not sure this works for all time-zones
 //Date-time midnight = Date-time.newInstance(Date.today(),
Time.newInstance(0,0,0,0));
 //System.assertEquals(midnight.addHours(24), cts[0].NextFireTime, 'ERROR: Not
scheduled for Midnight local time'); } }
```

APEX INTEGRATION SERVICES

1)Apex REST Callouts:-

1)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 JSONOutput parse(String json){
 //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class); //}
}
 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.setHeader('id', String.valueof(id)); --
cannot be used in this challenge :) request.setMethod('GET');
 HttpResponse response = http.send(request);
 system.debug('response: ' + response.getBody());
//Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
//Object results = (Object) map_results.get('animal'); system.debug('results= ' +
results.animal.name);
return(results.animal.name); }}
```

2)AnimalLocatorTest

```
@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);
} }
```

3)AnimalLocatorMock

```
@IsTest
 global class AnimalLocatorMock implements HttpCalloutMock {
global HTTPresponse respond(HTTPrequest request) { Httpresponse response = new
Httpresponse(); response.setStatusCode(200);
 //-- directly output the JSON, instead of creating a logic //response.setHeader('key,
value)
//Integer id = Integer.valueof(request.getHeader('id'));
 //Integer id = 1;
 //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
//system.debug('animal return value: ' + lst_body[id]);
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
return response; }
}
```

2)Apex SOAP Callouts:-

1)ParkLocator.apxc

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

2)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]);
} }
```

3)ParkServiceMock

```
@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 ParkService.byCountryResponse();
response_x = new
List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
response_x.return_x = lstOfDummyParks;
 response.put('response_x', response_x);
} }
```

3)Apex Web Services

1)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;
} }
```

2)AccountManagerTest

```
@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;
} }
```

APEX SPECIALIST SUPERBADGE

1)Automate Record Creation:

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
 public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
 Set<Id> validIds = new Set<Id>(); For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){ if (c.Type ==
'Repair' || c.Type == 'Routine Maintenance'){
validIds.add(c.Id); }
} }
if (!validIds.isEmpty()){
 List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r) FROM Case
```

```
WHERE Id IN :validIds]);
Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>(); AggregateResult[]
results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c]; for (AggregateResult ar : results){
maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
}
 for(Case cc : closedCasesM.values()){
Case nc = new Case ( ParentId = cc.Id,
Status = 'New',
 Subject = 'Routine Maintenance', Type = 'Routine Maintenance', Vehicle__c =
cc.Vehicle__c, Equipment__c =cc.Equipment__c, Origin = 'Web',
 Date_Reported__c = Date.Today()
);
 If (maintenanceCycles.containskey(cc.Id)){
maintenanceCycles.get(cc.Id)); }
newCases.add(nc); }
insert newCases;
nc.Date_Due__c = Date.today().addDays((Integer)
List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c wpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id; ClonedWPs.add(wpClone);
} }
insert ClonedWPs; }
} }
```

MaitenanceRequest.apxt :-

```
trigger MaintenanceRequest on Case (before update, after update) { if(Trigger.isUpdate
&& Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap); }
```

```
}
```

2)Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
//@future(callout=true)
 public static void runWarehouseEquipmentSync(){
Http http = new Http();
 HttpRequest request = new HttpRequest(); request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
 HttpResponse response = http.send(request); List<Product2> warehouseEq = new
List<Product2>(); if (response.getStatusCode() == 200){
List<Object>jsonResponse=(List<Object>)JSON.deserializeUntyped(response.getBody(
));
System.debug(response.getBody()); for (Object eq : jsonResponse){
Map<String,Object> mapJson = (Map<String,Object>)eq;
 Product2 myEq = new Product2();
 myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement'); myEq.Name =
(String) mapJson.get('name');
 myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan'); myEq.Cost__c =
(Decimal) mapJson.get('lifespan'); myEq.Warehouse_SKU__c = (String)
mapJson.get('sku'); myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
warehouseEq.add(myEq);
}
} }
} }
```

3)Schedule synchronization

Challenge:-

Build scheduling logic that executes your callout and runs your code daily. The name of the schedulable class should be WarehouseSyncSchedule, and the scheduled job should be named

WarehouseSyncScheduleJob.

Solution:

WarehouseSyncShedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable{ global
void execute(SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService()); }
}
```

4)Test automation logic

Challenge:-

Build tests for all cases (positive, negative, and bulk) specified in the business requirements by using a class named MaintenanceRequestHelperTest. You must have 100% test coverage to pass this section and assert values to prove that your logic is working as expected. Choose Run All Tests in the Developer Console at least once before attempting to submit this section. Be patient as it may take 10-20 seconds to process the challenge check.

Solution:

MaintenanceRequestHelperTest.apxc :-

```
@istest
 public with sharing class MaintenanceRequestHelperTest {
private static final string STATUS_NEW = 'New';
 private static final string WORKING = 'Working';
 private static final string CLOSED = 'Closed';
 private static final string REPAIR = 'Repair';
 private static final string REQUEST_ORIGIN = 'Web';
 private static final string REQUEST_TYPE = 'Routine Maintenance'; private static final
string REQUEST_SUBJECT = 'Testing subject'; PRIVATE STATIC Vehicle__c
createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
return Vehicle; }
PRIVATE STATIC Product2 createEq(){
 product2 equipment = new product2(name = 'SuperEquipment',
lifespan_months__C = 10, maintenance_cycle__C = 10, replacement_part__c = true);
```

```
return equipment; }
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){ case
cs = new case(Type=REPAIR,
Status=STATUS_NEW, Origin=REQUEST_ORIGIN, Subject=REQUEST_SUBJECT,
Equipment__c=equipmentId, Vehicle__c=vehicleId);
return cs; }
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return wp; }
@istest
 private static void testMaintenanceRequestPositive(){
Vehicle__c vehicle = createVehicle(); insert vehicle;
 id vehicleId = vehicle.Id;
 Product2 equipment = createEq(); insert equipment;
id equipmentId = equipment.Id;
 case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId); insert
somethingToUpdate;
Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
insert workP; test.startTest();
somethingToUpdate.status = CLOSED; update somethingToUpdate; test.stopTest();
Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c,
Vehicle__c, Date_Due__c
from case
where status =:STATUS_NEW]; Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newReq.Id]; system.assert(workPart != null);
system.assert(newReq.Subject != null); system.assertEquals(newReq.Type,
REQUEST_TYPE); SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}
 @istest
 private static void testMaintenanceRequestNegative(){
```

```
Vehicle__C vehicle = createVehicle(); insert vehicle;
 id vehicleId = vehicle.Id;
 product2 equipment = createEq(); insert equipment;
id equipmentId = equipment.Id;
 case emptyReq = createMaintenanceRequest(vehicleId,equipmentId); insert
emptyReq;
Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
insert workP;
 test.startTest();
 emptyReq.Status = WORKING; update emptyReq; test.stopTest();
 list<case> allRequest = [select id
from case]; Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :emptyReq.Id]; system.assert(workPart != null);
system.assert(allRequest.size() == 1); }
@istest
 private static void testMaintenanceRequestBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>(); list<Product2> equipmentList =
new list<Product2>();
list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
list<case> requestList = new list<case>(); list<id> oldRequestIds = new list<id>();
for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle()); equipmentList.add(createEq());
}
 insert vehicleList;
 insert equipmentList; for(integer i = 0; i < 300; i++){
requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
}
 insert requestList;
 for(integer i = 0; i < 300; i++){
workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id)); }
insert workPartList; test.startTest();
 for(case req : requestList){
req.Status = CLOSED;
oldRequestIds.add(req.Id); }
```

```
update requestList; test.stopTest();
 list<case> allRequests = [select id
from case
where status =: STATUS_NEW]; list<Equipment_Maintenance_Item__c> workParts =
[select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in: oldRequestIds]; system.assert(allRequests.size()
== 300);
} }
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
 public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
 Set<Id> validIds = new Set<Id>(); For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){ if (c.Type ==
'Repair' || c.Type == 'Routine Maintenance'){
validIds.add(c.Id); }
} }
if (!validIds.isEmpty()){
 List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]); Map<Id,Decimal> maintenanceCycles = new
Map<ID,Decimal>();
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds
GROUP BY Maintenance_Request__c];
for (AggregateResult ar : results){
 maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle')); }
for(Case cc : closedCasesM.values()){
Case nc = new Case ( ParentId = cc.Id,
Status = 'New',
```

```
 Subject = 'Routine Maintenance',
Type = 'Routine Maintenance', Vehicle__c = cc.Vehicle__c, Equipment__c
=cc.Equipment__c, Origin = 'Web',
Date_Reported__c = Date.Today() );
If (maintenanceCycles.contains key(cc.Id)){
 nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id)); }
newCases.add(nc); }
insert newCases;
List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c wpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id; ClonedWPs.add(wpClone);
} }
insert ClonedWPs; }
} }
```

MaintenanceRequest.apt :-

```
trigger MaintenanceRequest on Case (before update, after update)
 {
if(Trigger.isUpdate && Trigger.isAfter)
{
 MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
 }}
```

5)Test call-out logic

Challenge:

Build tests for your callout using the included class for the callout mock (WarehouseCalloutServiceMock) and call-out test class (WarehouseCalloutServiceTest) in the package. You must have 100% test coverage to pass this challenge and assert values to prove that your logic is working as expected.

Solution:

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
//@future(call-out=true)
 public static void runWarehouseEquipmentSync(){
Http http = new Http();
 HttpRequest request = new HttpRequest(); request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
 HttpResponse response = http.send(request); List<Product2> warehouseEq = new
List<Product2>(); if (response.getStatusCode() == 200){
List<Object> (List<Object>)JSON.deserializeUntyped(response.getBody());
System.debug(response.getBody()); for (Object et : jsonResponse){
jsonResponse =
Map<String,Object> mapJson = (Map<String,Object>)et;
 Product2 myEq = new Product2();
 myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement'); myEq.Name =
(String) mapJson.get('name');
 myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan'); myEq.Cost__c =
(Decimal) mapJson.get('lifespan'); myEq.Warehouse_SKU__c = (String)
mapJson.get('sku'); myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
warehouseEq.add(myEq);
}
 if (warehouseEq.size() > 0){
upset warehouseEq;
 System.debug('Your equipment was synced with the warehouse one');
```

```
System.debug(warehouseEq);
} }
} }
```

WarehouseCalloutServiceTest.apxc :-

```
@isTest
private class WarehouseCalloutServiceTest { @isTest
static void testWareHouseCallout(){
 Test.startTest();
 // implement mock call-out test here
 Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
 Test.stopTest();
 System.assertEquals(1, [SELECT count() FROM Product2]);
} }
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest
 global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock call-out
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
 System.assertEquals('GET', request.getMethod());
 // Create a fake response
 HttpResponse response = new HttpResponse(); response.setHeader('Content-Type',
'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":
5,"name":"Generator 1000 kW","maintenance
period":365,"lifespan":120,"cost":5000,"ski":"100003"}]');
response.setStatusCode(200);
return response; }
}
```

6)Test scheduling logic

Challenge:

Build unit tests for the class WarehouseSyncSchedule in a class named WarehouseSyncScheduleTest. You must have 100% test coverage to pass this challenge and assert values to prove that your logic is working as expected.

Solution:

WarehouseSyncSchedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable{ global
void execute(SchedulableContext cox){
System.enqueueJob(new WarehouseCalloutService()); }
}
```

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 ');
} }
```