Challenge 1: 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:

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

for(Account a: Trigger.New){

if(a.Match\_Billing\_Address\_\_c == true && a.BillingPostalCode!= null){

a.ShippingPostalCode=a.BillingPostalCode;

}

}

}

Challenge 2:

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:

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

List<Task> taskList = new List<Task>();

//first way

for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName='Closed Won' AND Id IN : Trigger.New]){

taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));

}

//second way and we should use this

/\*

for(opportunity opp: Trigger.New){

if(opp.StageName!=trigger.oldMap.get(opp.id).stageName)

{

taskList.add(new Task(Subject = 'Follow Up Test Task',

WhatId = opp.Id));

}

}

\*/

if(taskList.size()>0){

insert tasklist;

}

}

Challenge 3:

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

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

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 {

static testMethod void TestVerifyDate() {

VerifyDate.CheckDates(System.today(),System.today().addDays(10));

VerifyDate.CheckDates(System.today(),System.today().addDays(78));

}

}

Challenge 4:

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') { //invalidname 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 ee)

{

}

Test.stopTest();

}

}

Challenge 5:

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:

1.RandomContactFactory.apxc

public class RandomContactFactory {

public static List<Contact> generateRandomContacts(Integer NumberofContacts, String lName){

List<Contact> con = new List<Contact>();

for(Integer i=0; i<NumberofContacts; i++){

lName = 'Test'+i;

Contact c = new Contact(FirstName=lName, LastName=lName);

con.add(c);

}

return con;

}

}

Challenge 6:

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 class AccountProcessor {

@future

public static void countContacts(Set<Id> setId){

List<Account> lstAccount = [select Id,Number\_of\_Contacts\_\_c,(select id from contacts) from account where id in :setId];

for(Account acc : lstAccount){

List<Contact> lstCont = acc.contacts;

acc.Number\_of\_Contacts\_\_c = lstCont.size();

}

update lstAccount;

}

}

2.AccountProcessorTest.apxc

@isTest

public class AccountProcessorTest {

public static testMethod void testAccountProcessorTest(){

Test.startTest();

Account a = new Account();

a.Name = 'The Pirates';

insert a;

Contact cont = new Contact();

cont.FirstName ='jack';

cont.LastName ='Sparrow';

cont.AccountId = a.Id;

insert cont;

Set<Id> setAccId = new Set<ID>();

setAccId.add(a.Id);

AccountProcessor.countContact(setAccId);

Account acc = [select Number\_of\_Contacts\_\_c from Account where id = :a.id LIMIT 1];

System.assertEquals(Integer.valueOf(acc.Number\_of\_Contacts\_\_c) ,1);

Test.stopTest();

}

}

Challenge 7:

Create an Apex class that uses Batch Apex to update Lead records.

Create an Apex class that implements the Database.Batchable 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.Batchable 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

global class LeadProcessor implements Database.Batchable<sObject>, Database.Stateful {

global Integer leadsProcessed = 0;

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

return Database.getQueryLocator('select id, lastname ,status, company from Lead');

}

global void execute(Database.BatchableContext bc, List<Lead> allLeads){

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

for(Lead l: allLeads){

l.LeadSource='Dreamforce';

}

update leads;

}

global void finish(Database.BatchableContext bc){

System.debug(leadsProcessed + ' leads processed. Nigga!');

AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,

JobItemsProcessed,

TotalJobItems, CreatedBy.Email

FROM AsyncApexJob

WHERE Id = :bc.getJobId()];

EmailManager.sendMail('jgatsby1996@gmail.com','Total Leads Porcessed are ',' '+leadsProcessed);

}

}

2.LeadProcessorTest

@isTest

public class LeadProcessorTest {

@testSetup

static void setup(){

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

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

leads.add(new Lead(Lastname='Last '+i,

status='Open - Not Contacted'

, company='LeadCompany'+i));

}

insert leads;

}

static testmethod void test() {

Test.startTest();

LeadProcessor lp = new LeadProcessor();

Id batchId = Database.executeBatch(lp);

Test.stopTest();

// after the testing stops, assert records were updated properly

System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);

}

}

Challenge 8:

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 Queueable 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 class AddPrimaryContact implements Queueable {

public contact c;

public String state;

public AddPrimaryContact(Contact c, String state) {

this.c = c;

this.state = state;

}

public void execute(QueueableContext qc) {

system.debug('this.c = '+this.c+' this.state = '+this.state);

List<Account> acc\_lst = new List<account>([select id, name, BillingState from account where account.BillingState = :this.state limit 200]);

List<contact> c\_lst = new List<contact>();

for(account a: acc\_lst) {

contact c = new contact();

c = this.c.clone(false, false, false, false);

c.AccountId = a.Id;

c\_lst.add(c);

}

insert c\_lst;

}

}

2.AddPrimaryContactTest.apxc

@isTest

public class AddPrimaryContactTest {

@testSetup

public static void setup(){

List<account> acc\_lst = new List<account>();

for (Integer i=0; i<50;i++) {

account a = new account(name=string.valueOf(i),billingstate='NY');

system.debug('account a = '+a);

acc\_lst.add(a);

}

for (Integer i=0; i<50;i++) {

account a = new account(name=string.valueOf(50+i),billingstate='CA');

system.debug('account a = '+a);

acc\_lst.add(a);

}

insert acc\_lst;

}

public static testMethod void TestQueueable(){

List<Account> ac\_ca=[select id from Account where billingstate='CA'];

contact c = new contact(lastname='bhau');

AddPrimaryContact apc = new AddPrimaryContact(c,'CA');

Test.startTest();

System.enqueueJob(apc);

Test.stopTest();

system.assertEquals(50, [select count() from contact where AccountId IN :ac\_ca]);

}

}

Challenge 9:

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 class DailyLeadProcessor implements schedulable{

public void execute(schedulableContext sc) {

List<lead> l\_lst\_new = new List<lead>();

List<lead> l\_lst = new List<lead>([select id, leadsource from lead where leadsource = null]);

for(lead l : l\_lst) {

l.leadsource = 'Dreamforce';

l\_lst\_new.add(l);

}

update l\_lst\_new;

}

}

2.DailyLeadProcessorTest.apxc

@isTest

public class DailyLeadProcessorTest {

@testSetup

static void setup(){

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

for(Integer i = 1; i <= 200; i++){

Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working - Contacted');

lstOfLead.add(ld);

}

Insert lstOfLead;

}

static testmethod void testDailyLeadProcessorScheduledJob(){

String sch = '0 5 12 \* \* ?';

Test.startTest();

String jobId = System.schedule('ScheduledApexTest', sch, new DailyLeadProcessor());

List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];

System.assertEquals(200, lstOfLead.size());

Test.stopTest();

}

}

* Challenge 10:Create an Apex class:
  + Name: AnimalLocator
  + Method name: getAnimalNameById
  + The method must accept an Integer and return a String.
  + The method must call https://th-apex-http-callout.herokuapp.com/animals/<id>, replacing <id> with the ID passed into the method
  + The method returns the value of the **name** property (i.e., the animal name)
* Create a test class:
  + Name: AnimalLocatorTest
  + The test class uses a mock class called AnimalLocatorMock to mock the callout response
* Create unit tests:
  + Unit tests must cover all lines of code included in the **AnimalLocator** 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

##AnimalLocator.apxc

public class AnimalLocator

{

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);

String strResp = '';

system.debug('\*\*\*\*\*\*response '+response.getStatusCode());

system.debug('\*\*\*\*\*\*response '+response.getBody());

// If the request is successful, parse the JSON response.

if (response.getStatusCode() == 200)

{

// Deserializes the JSON string into collections of primitive data types.

Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());

// Cast the values in the 'animals' key as a list

Map<string,object> animals = (map<string,object>) results.get('animal');

System.debug('Received the following animals:' + animals );

strResp = string.valueof(animals.get('name'));

System.debug('strResp >>>>>>' + strResp );

}

return strResp ;

}

}

##AnimalLocatorTest

@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);

}

}

##AnimalLocatorMock

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

global HTTPResponse respond(HTTPRequest request) {

HttpResponse response = new HttpResponse();

response.setHeader('Content-Type', 'application/json');

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

response.setStatusCode(200);

return response;

}

}

* Challenge 11:Generate a class using this using [this WSDL file](https://th-apex-soap-service.herokuapp.com/assets/parks.xml):
  + Name: ParkService (Tip: After you click the **Parse WSDL** button, change the Apex class name from **parksServices** to ParkService)
  + Class must be in public scope
* Create a class:
  + Name: ParkLocator
  + Class must have a **country** method that uses the **ParkService** class
  + Method must return an array of available park names for a particular country passed to the web service (such as Germany, India, Japan, and United States)
* Create a test class:
  + Name: ParkLocatorTest
  + Test class uses a mock class called ParkServiceMock to mock the callout response
* Create unit tests:
  + Unit tests must cover all lines of code included in the **ParkLocator** 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.

##ParkLocator.apxc

public class ParkLocator {

public static String[] country(String country){

ParkService.ParksImplPort Locator = new ParkService.ParksImplPort();

return Locator.byCountry(country);

}

}

##ParkLocatorTest.apxc

@isTest

public class ParkLocatorTest {

@isTest static void testMock(){

test.setMock(WebserviceMock.class, new ParkServiceMock());

String[] parksName = ParkLocator.Country('India');

List<String> country = new List<String>();

country.add('Inamdar National Park');

country.add('Riza National Park');

country.add('Shilpa National Park');

System.assertEquals(country, parksName, 'park names are not as expected');

}

}

##ParkServiceMock

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> country = new List<String>();

country.add('Inamdar Shola National Park');

country.add('Riza National Park');

country.add('Shilpa National Park');

response\_x.return\_x = country;

response.put('response\_x', response\_x);

}

}

* Challenge 12:Create an Apex class
  + Name: AccountManager
  + Class must have a method called getAccount
  + Method must be annotated with **@HttpGet** and return an **Account** object
  + Method must return the **ID** and **Name** for the requested record and all associated contacts with their **ID** and **Name**
* Create unit tests
  + Unit tests must be in a separate Apex class called AccountManagerTest
  + Unit tests must cover all lines of code included in the **AccountManager** 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

##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

@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;

}

}

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 1

* Go to the App Launcher -> Search How We Roll Maintenance -> click on Maintenance Requests -> click on first case -> click Details -> change the type Repair to Routine Maintenance -> select Origin = Phone -> Vehicle = select Teardrop Camper , save it.
* Feed -> Close Case = save it..
* Go to the Object Manager -> Maintenance Request ->Field & Relationships ->New ->Lookup Relationship -> next -> select Equipment ->next -> Field Label = Equipment ->next->next->next -> save it .
* Now go to the developer console use below code

**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)){*  
 *nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));*  
 *} else {*  
 *nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);*  
 *}*  
  
 *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;*  
 *}*  
 *}*  
*}*

### 

### MaitenanceRequest.apxt :-

trigger MaintenanceRequest on Case (before update, after update) {

if(Trigger.isUpdate && Trigger.isAfter){

MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

}

}

* After saving the code go back the How We Roll Maintenance ,
* click on Maintenance Requests -> click on 2nd case -> click Details -> change the type Repair to Routine Maintenance -> select Origin = Phone -> Vehicle = select Teardrop Camper , save it.
* Feed -> Close Case = save it..

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 2

* Setup -> Search in quick find box -> click Remote Site Settings -> Name = Warehouse URL , Remote Site URL = https://th-superbadge-apex.herokuapp.com , make sure active is selected.
* Go to the developer console use below code

### WarehouseCalloutService.apxc :-

public with sharing class WarehouseCalloutService implements Queueable {  
 private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';  
   
 //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.  
 //The callout’s JSON response returns the equipment records that you upsert in Salesforce.   
   
 @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());  
   
 //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU  
 //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce  
 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 = (Integer) mapJson.get('cost');  
 myEq.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');  
 myEq.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');  
 myEq.ProductCode = (String) mapJson.get('\_id');  
 warehouseEq.add(myEq);  
 }  
   
 if (warehouseEq.size() > 0){  
 upsert warehouseEq;  
 System.debug('Your equipment was synced with the warehouse one');  
 }  
 }  
 }  
   
 public static void execute (QueueableContext context){  
 runWarehouseEquipmentSync();  
 }  
   
}

**After saving the code open execute anonymous window ( CTRl+E ) and run this method ,**

System.enqueueJob(new WarehouseCalloutService());

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 3

* Go to the developer console use below code :

### WarehouseSyncShedule.apxc :-

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

Save it , after that...

* Go to setup -> Seacrh in Quick find box -> Apex Classes -> click Schedule Apex and Jb Name = WarehouseSyncScheduleJob , Apex Class = WarehouseSyncSchedule

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 4

* Go to the developer console use below code :

### 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.containskey(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.apxt :-

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

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 5

* Go to the developer console use below code ,

### 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);  
 }  
   
 if (warehouseEq.size() > 0){  
 upsert 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 callout 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 callout  
 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","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');  
 response.setStatusCode(200);  
 return response;  
 }  
}

***Note-:Deleted all scheduled jobs that are under Setup -> Monitoring -> Scheduled Jobs***

# APEX SPECIALIST SUPERBADGE- -CHALLENGE 6

* Go to the developer console use below 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 ');  
   
   
 }  
}

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |