**APEX SPECIALIST SUPER BADGE CODES**

**APEX TRIGGERS**

**AccountAddressTrigger.apxt:-**

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

for(Account a:Trigger.New){

if(a.Match\_Billing\_Address\_\_c==true){

a.ShippingPostalCode=a.BillingPostalCode;

}

}

}

**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;

}

}

**Apex Testing**

**VerifyDate.apxc:-**

public class VerifyDate {

public static Date CheckDates(Date date1, Date date2) {

if(DateWithin30Days(date1,date2)) {

return date2;

}

else {

return SetEndOfMonthDate(date1);

}

}

private static Boolean DateWithin30Days(Date date1, Date date2) {

if( date2 < date1) { return false; }

Date date30Days = date1.addDays(30);

if( date2 >= date30Days ) { return false; }

else { return true; }

}

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

public class TestVerifyDate

{

static testMethod void testMethod1()

{

Date d = VerifyDate.CheckDates(System.today(),System.today()+1);

Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);

}

}

**RestrictContactByName.apxt**

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

for (Contact c :Trigger.New) {

if(c.LastName == 'INVALIDNAME') {

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

@isTest

private class TestRestrictContactByName {

static testMethodvoid metodoTest() {

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

Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' , email='Test@test.com');

Contact c2 = new Contact(FirstName='Francesco1', LastName = 'INVALIDNAME',email='Test@test.com');

listContact.add(c1);

listContact.add(c2);

Test.startTest();

try{

insert listContact;

}

catch(Exception e){}

Test.stopTest();

}

}

**RandomContactFactory.apxc:**

public class RandomContactFactory {

public static List<Contact>generateRandomContacts(Integer numContactsToGenerate, String FName) {

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

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

Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);

contactList.add(c);

System.debug(c);

}

System.debug(contactList.size());

return contactList;

}

}

**Asynchronous Apex**

**AccountProcessor.apxc**

public class AccountProcessor {

@future

public static void countContacts(List<Id>accountIds){

List<Account> accounts = [Select Id, Name from Account Where Id IN :accountIds];

List<Account>updatedAccounts = new List<Account>();

for(Account account : accounts){

account.Number\_of\_Contacts\_\_c = [Select count() from Contact Where AccountId =: account.Id];

System.debug('No Of Contacts = ' + account.Number\_of\_Contacts\_\_c);

updatedAccounts.add(account);

}

update updatedAccounts;

}

}

**AccountProcessorTest.apxc**

@isTest

public class AccountProcessorTest {

@isTest

public static void testNoOfContacts(){

Account a = new Account();

a.Name = 'Test Account';

Insert a;

Contact c = new Contact();

c.FirstName = 'Bob';

c.LastName= 'Willie';

c.AccountId = a.Id;

Contact c2 = new Contact();

c2.FirstName = 'Tom';

c2.LastName = 'Cruise';

c2.AccountId = a.Id;

List<Id>acctIds = new List<Id>();

acctIds.add(a.Id);

Test.startTest();

AccountProcessor.countContacts(acctIds);

Test.stopTest();

}

}

**LeadProcessor.apxc:**

public class LeadProcessor implements Database.Batch able<sObject> {

public Database.QueryLocatorstart(Database.BatchableContext BC) {

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

}

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

for (Lead Lead : leads) {

lead.LeadSource = 'Dreamforce';

}

update leads;

}

public void finish(Database.BatchableContext be){

}

}

**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();

LeadProcessorleadProcessor = new LeadProcessor();

Id batchId = Database.executeBatch(leadProcessor);

Test.stopTest();

}

}

**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 a cc: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> Test e = 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 a pc = new AddPrimaryContact(co, state);

Test.startTest();

System.enqueueJob(apc);

Test.stopTest();

}

}

**DailyLeadProcessor.apxc**

public class DailyLeadProcessor implements Schedule {

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**

**AnimalLocator.apxc:**

public class AnimalLocator{

public static String getAnimalNameById(Integer x){

Http http = new Http();

HttpRequest re q = 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 rest = 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');

}

}

**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 );

}

}

**AnimalLocatorMock.apxc**

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

// Implement this interface method

global HTTPResponserespond(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;

}

}

**ParkLocator.apxc**

public class ParkLocator {

public static string[] country(string theCountry) {

ParkService.ParksImplPortparkSvc = 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.byCountryResponseresponse\_x = new ParkService.byCountryResponse();

response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

// end

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

}

}

**AccountManager.apxc**

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

global with sharing class AccountManager {

@HttpGet

global static Account getAccount(){

RestRequest request=RestContext.request;

string accountId=request.requestURI.substringBetween('Accounts/','/contacts');

Account result=[SELECT Id,Name,(Select Id,Name from Contacts) from Account where Id=:accountId Limit 1];

return result;

}

}

**AccountManagerTest.apxc**

@IsTest

private class AccountManagerTest {

@isTest static void testGetContactsByAccountId(){

Id recordId=createTestRecord();

RestRequest request=new RestRequest();

request.requestUri='https://yourInstance.my.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);

}

static Id createTestRecord(){

Account accountTest=new Account(

Name='Test record'

);

insert accountTest;

Contact contactTest=new Contact(

FirstName='John',LastName='Doe',AccountId=accountTest.Id);

insert contactTest;

return accountTest.Id;

}

}

**APEX SPECIALIST SUPER BADGE**

**Challenge 1:**

**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 (AggregateResultare : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));

}

for(Case cc : closedCasesM.values()){

Case no = 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 no :newCases){

for (Equipment\_Maintenance\_Item\_\_cwe :closedCasesM.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){

Equipment\_Maintenance\_Item\_\_cwpClone = 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);

}

}

**Challenge-2:**

**WarehouseCalloutService.apxc**

public with sharing class WarehouseCalloutService implements Queue able {

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 ex :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){

upset warehouseEq;

System.debug('Your equipment was synced with the warehouse one');

}

}

}

public static void execute (QueueableContextcontext){

runWarehouseEquipmentSync();

}

}

**Challenge-3:**

**WarehouseSyncSchedule.apxc**

global class WarehouseSyncSchedule implements Schedulable {

global void execute(SchedulableContext cox) {

WarehouseCalloutService.runWarehouseEquipmentSync();

}

}

**Challenge-4:**

**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\_\_ccreateVehicle(){

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 c = new case(Type=REPAIR,

Status=STATUS\_NEW,

Origin=REQUEST\_ORIGIN,

Subject=REQUEST\_SUBJECT,

Equipment\_\_c=equipmentId,

Vehicle\_\_c=vehicleId);

return c;

}

PRIVATE STATIC Equipment\_Maintenance\_Item\_\_ccreateWorkPart(id equipmentId,idrequestId){

Equipment\_Maintenance\_Item\_\_c we = new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipmentId,

Maintenance\_Request\_\_c = requestId);

return we;

}

@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\_\_cworkP = 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\_\_cworkPart = [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\_\_cworkP = 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\_\_cworkPart = [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 re q : requestList){

re q.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 (AggregateResultare : results){

maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));

}

for(Case cc : closedCasesM.values()){

Case no = 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 no :newCases){

for (Equipment\_Maintenance\_Item\_\_cwe :closedCasesM.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){

Equipment\_Maintenance\_Item\_\_cwpClone = 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);

}

}

**Challenge-5:**

**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 ex :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){

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 {

global static HttpResponserespond(HttpRequest request){

System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment', request.getEndpoint());

System.assertEquals('GET', request.getMethod());

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;} }

**Challenge-6:**

**WarehouseSyncSchedule.apxc**

global class WarehouseSyncSchedule implements Schedulable {

global void execute(SchedulableContext cox) {

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();

CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime> today];

System.assertEquals(jobID, a.Id,'Schedule ');

}

}