AnimalLocator:-

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 class:-

@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;

}

}

ParkLocator Class:-

public class ParkLocator {

public static String[] country(String ctry) {

ParkService.ParksImplPort prk =

new ParkService.ParksImplPort();

return prk.byCountry(ctry);

}

}

ParklocatorTest:-

@isTest

private class ParkLocatorTest {

@isTest static void testCallout() {

// This causes a fake response to be generated

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

// Call the method that invokes a callout

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

List<String> expectedvalue = new List<String>{'Park1','Park2','Park3'};

result = ParkLocator.country('India');

// Verify that a fake result is returned

System.assertEquals(expectedvalue, result);

}

}

ParkSErviceMock:-

AccountAddressTrigger:

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:

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

list<Task> newTask= new list<Task>();

for(Opportunity oppWon :[Select Id from Opportunity where StageName='Closed Won'

and Id in: Trigger.new]){

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

}

if(newTask.size()>0){

upsert newTask;

}

}

VerifyDate:

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;

}

}

VerifydateTest:

@isTest  
private class TestVerifyDate {  
    static testMethod void TestVerifyDate() {  
      VerifyDate.CheckDates(System.today(),System.today().addDays(10));  
       VerifyDate.CheckDates(System.today(),System.today().addDays(78));  
    }  
}

RestrictContactByName:

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

//check contacts prior to insert or update for invalid data

For (Contact c : Trigger.New) {

if(c.LastName == 'INVALIDNAME') { //invalidname is invalid

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

}

}

}

TestRestrictContactByName:

@isTest

public class TestRestrictContactByName {

@isTest static void TestRestrictContactByNameValid() {

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

ctt.add(new Contact(LastName='Welter',FirstName = 'Tiago'));

ctt.add(new Contact(LastName='AAA'));

ctt.add(new Contact(LastName='INVALIDNAME',FirstName = 'Tiago'));

ctt.add(new Contact());

Test.startTest();

Database.SaveResult[] result = Database.insert(ctt, false);

Test.stopTest();

for(Database.SaveResult sr : result){

if(!sr.isSuccess()){

System.assert(!sr.isSuccess());

}else{

System.assert(sr.isSuccess());

}

}

}

}

geberateRandomContacts:

public  List<Contact> generateRandomContacts(Integer iContacts, String sLastName){  
        List<Contact> lstRndContacts = new List<Contact>();  
        for(Integer i=0; i<10; i++){  
            String FN = 'FN\_' +i;  
            system.debug(FN);  
            system.debug('Hello World');  
              
            Contact oContact = new Contact(LastName = sLastName, FirstName = FN);  
            system.debug(sLastName);  
             system.debug(FN);  
            lstRndContacts.add(oContact);  
            system.debug(oContact);  
        }  
         system.debug(lstRndContacts);  
        return lstRndContacts;  
         
    }

AccountProcesssor:

public class AccountProcessor {  
    @future  
    public static void countContacts(set<ID> Accnts){  
        List<Account> acc = [select Name from Account where Id IN :accnts];  
        List<Account> xyz = new List<Account>();  
        for(Account a:acc){  
            a.Number\_of\_Contacts\_\_c = [select Count() from Contact where AccountId =:a.Id];  
            xyz.add(a);  
        }  
        update xyz;  
    }  
}

AccountProcessorTest:

@isTest

public class AccountProcessorTest {

@isTest public static void accnt(){

Account a = new Account();

a.Name = 'Test Account';

Insert a;

Contact cont = New Contact();

cont.FirstName ='Bob';

cont.LastName ='Masters';

cont.AccountId = a.Id;

Insert cont;

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

setAccId.add(a.id);

Test.startTest();

AccountProcessor.countContacts(setAccId);

Test.stopTest();

}

}

LeadProcessor:

global class LeadProcessor implements Database.Batchable<sObject>{

global integer recordsProcessed = 1;

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

return Database.getQueryLocator('SELECT ID, LeadSource from Lead where LeadSource = \'Dreamforce\'');

}

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

for(Lead l : scope){

l.LeadSource = 'Web';

}

update scope;

}

global void finish(Database.BatchableContext bc){

System.debug(recordsProcessed + ' records processed. Shazam!');

AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,

JobItemsProcessed,

TotalJobItems, CreatedBy.Email

FROM AsyncApexJob

WHERE Id = :bc.getJobId()];

// call some utility to send email

// EmailUtils.sendMessage(a, recordsProcessed);

}

}

LeadProcessorTest:

@isTest

public class LeadProcessorTest {

@testSetup

static void setupLeads() {

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

// insert 200 Leads

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

leads.add(new Lead(company = 'testCompany' + i,

Leadsource='DreamForce', LastName = 'LeadingEdge' + i));

}

insert leads;

}

@isTest

static void testUpdateLeadBatch(){

List<lead> leadsForUpdate =[select id, LeadSource from Lead where LeadSource = 'Dreamforce'];

Test.startTest();

LeadProcessor lp = new LeadProcessor();

id batchJobId = Database.executeBatch(lp,200);

Test.stopTest();

List<lead> leadsAfterUpdate =[select id, LeadSource from Lead where LeadSource = 'Dreamforce'];

System.debug(leadsAfterUpdate);

System.assertEquals(0, leadsAfterUpdate.size());

}

}

AddPrimaryContact:

public class AddPrimaryContact implements Queueable{

private Contact c;

private String state;

public AddPrimaryContact(Contact contact, String state){

this.c = contact;

this.state = state;

}

public void execute(QueueableContext context){

List<Account> accounts = [Select ID, Name From Account WHERE BillingState =:state LIMIT 200];

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

for(Account account: accounts){

Contact con = c.clone(false, false,false, false);

con.AccountId = account.Id;

contacts.add(con);

}

if(contacts.size()>0){

insert contacts;

}

}

}

AddprimaryContactTest:

@isTest

public class AddPrimaryContactTest {

@testSetup

static void setup(){

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

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

testAccounts.add(new Account(BillingState = 'NY', name = 'QueueTest ' + i));

}

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

testAccounts.add(new Account(BillingState = 'CA', name = 'QueueTest ' + j));

}

insert testAccounts;

}

static testMethod void test(){

Contact con = new Contact(FirstName = 'Queueable', LastName = 'Apex');

insert con;

String state = 'CA';

Test.startTest();

AddPrimaryContact primContact = new AddPrimaryContact(con, state);

System.enqueueJob(primContact);

Test.stopTest();

System.assertEquals(50, [SELECT count() FROM Contact WHERE accountId IN (Select id from Account WHERE BillingState =:state)]);

}

}

DailyLeadProcessorTest:

@isTest

private class DailyLeadProcessorTest {

public static String CRON\_EXP = '0 0 0 15 3 ? 2022';

static testmethod void testScheduledJob() {

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

Leads.add(new lead(

name='Dream force'+i

));

}

insert Leads;

}

Test.startTest();

String jobId = System.schedule('ScheduledApexTest',

CRON\_EXP,

new DailyLeadProcessorTest());

}

AccountManager:

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

global with sharing class AccountManager {

@HttpGet

global static cust\_AcnCon1 getAccount() {

//Option1 - 2 queries and minimal processing

RestRequest request = RestContext.request;

String sID = request.requestURI.substringBetween('Accounts/','/contacts');

cust\_AcnCon1 sObj1 = new cust\_AcnCon1();

Map<Id,Account> sM1 = new Map<Id,Account>([Select Id, Name from Account where Id= :sID ]);

String sString1 = 'Incorrect Id';

If (sM1.size() == 0){

System.Debug('No Records Retrieved for the Id:'+ sID);

}

else if (sM1.Size() > 1){

System.Debug('Multiple Records Retrieved for the Id:'+sId);

}

else if (sM1.size() == 1){

System.Debug('Account Id:'+ sID);

}

//why Map? Just for fun

sObj1.AccountId = sM1.get(sID).Id;

sObj1.AccountName = sM1.get(sID).Name;

sObj1.ChildCon = new List<Contact>([Select Id, Name from Contact where AccountId = :sID]);

sObj1.sTest1 = True;

return sObj1;

//can think of an

// Aggregate Result

// Object - need to check

// List/Array

}

global class cust\_AcnCon1{

Global String AccountId=null;

Global String AccountName=null;

Global List<Contact> ChildCon=null;

Global Boolean sTest1 = False;

}

}

AccountManagerTest:

@isTest

private class AccountManagerTest {

@isTest static void testGetAccountById() {

Id recordId = createTestRecord();

// Set up a test request

RestRequest request = new RestRequest();

request.requestUri =

'https://actionfire-dev-ed.my.salesforce.com/services/apexrest/Accounts/'

+ recordId +'/contacts';

request.httpMethod = 'GET';

RestContext.request = request;

// Call the method to test

Account thisAcc = AccountManager.getAcccount();

}

// Helper method

static Id createTestRecord() {

// Create test record

Account testAcc = new Account(name='Test Account123');

insert testAcc;

Contact Cont1 = new Contact();

Cont1.LastName = 'ABC';

Cont1.AccountId = testAcc.Id ;

insert Cont1;

Contact Cont2 = new Contact();

Cont2.LastName = 'XYZ';

Cont2.AccountId =testAcc.Id ;

insert Cont2 ;

return testAcc.Id;

}

}

AccountManager:-

@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');

system.debug(accountId);

Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM Contacts) FROM Account WHERE Id = :accountId LIMIT 1];

return objAccount;

}

AccountManagerTest:-

@isTest

private class AccountManagerTest{

static testMethod void testMethod1(){

Account objAccount = new Account(Name = 'test Account');

insert objAccount;

Contact objContact = new Contact(LastName = 'test Contact',

AccountId = objAccount.Id);

insert objContact;

Id recordId = objAccount.Id;

RestRequest request = new RestRequest();

request.requestUri =

'https://sandeepidentity-dev-ed.my.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 Account', thisAccount.Name);

}

}

LWC

App.html:

<template>

<div>

<div>Name: {name}</div>

<div>Description: {description}</div>

<div>Category: {category}</div>

<div>Material: {material}</div>

<div>Price: {price}</div>

<div><img src={pictureUrl}/></div>

</div>

</template>

App.js:

import { LightningElement } from 'lwc';

export default class App extends LightningElement {

name = 'Electra X4';

description = 'A sweet bike built for comfort.';

category = 'Mountain';

material = 'Steel';

price = '$2,700';

pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';

}

Bike.html:

<!-- bike.html -->

<template>

<img src={bike.picture} alt="bike picture" />

<p>{bike.name}</p>

</template>

Bike.js:

// bike.js

import { LightningElement, api } from 'lwc';

export default class Bike extends LightningElement {

@api bike;

}

bikeCard.html

<template>

<div>

<div>Name: {name}</div>

<div>Description: {description}</div>

<lightning-badge label={material}></lightning-badge>

<lightning-badge label={category}></lightning-badge>

<div>Price: {price}</div>

<div><img src={pictureUrl}/></div>

</div>

</template>

bikeCard.js

import { LightningElement } from 'lwc';

export default class BikeCard extends LightningElement {

name = 'Electra X4';

description = 'A sweet bike built for comfort.';

category = 'Mountain';

material = 'Steel';

price = '$2,700';

pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';

}

bikeCard-jsmeta.xml:

import { LightningElement } from 'lwc';

export default class BikeCard extends LightningElement {

name = 'Electra X4';

description = 'A sweet bike built for comfort.';

category = 'Mountain';

material = 'Steel';

price = '$2,700';

pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';

}

Selector.html:

<template>

<div class="wrapper">

<header class="header">Available Bikes for {name}</header>

<section class="content">

<div class="columns">

<main class="main" >

<c-list onproductselected={handleProductSelected}></c-list>

</main>

<aside class="sidebar-second">

<c-detail product-id={selectedProductId}></c-detail>

</aside>

</div>

</section>

</div>

</template>

Selector.js:

import { LightningElement, api } from 'lwc';

import { bikes } from 'c/data';

export default class Detail extends LightningElement {

product;

// Private var to track @api productId

\_productId = undefined;

// Use set and get to process the value every time it's

// requested while switching between products

set productId(value) {

this.\_productId = value;

this.product = bikes.find(bike => bike.fields.Id.value === value);

}

// getter for productId

@api get productId(){

return this.\_productId;

}

}

**Apex Specialist Superbadge-**

**Automate record creation:**

**2.MaintenanceRequestHelper-**

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;

}

}

}

MaintenanceRequest:

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

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

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

}

}

**3.Synchronize Salesforce data with an external system:**

**WarehouseCalloutService:**

**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();**

**}**

**}**

**4.Schedule synchronization:**

**WareHouseSyncSchedule:**

**global with sharing class WarehouseSyncSchedule implements Schedulable{**

**global void execute(SchedulableContext ctx){**

**System.enqueueJob(new WarehouseCalloutService());**

**}**

**}**

**5.Test automation logic:**

**@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:**

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

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

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

**}**

6.Test Callout Logic

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);

}

}

}

}

@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;

}

}

@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]);

}

}

7.Testing Scheduling Logic-

Warehousesyncscheduletest-

@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 ');

}

}

WarehouseSyncSchedule-

global class WarehouseSyncSchedule implements Schedulable {

global void execute(SchedulableContext ctx) {

WarehouseCalloutService.runWarehouseEquipmentSync();

}

}

1. **Create a Visualforce page that displays a variety of output fields**

<apex:page standardController="Opportunity">

<apex:pageBlock title="Opportunity Summary">

<apex:outputField value="{! Opportunity.Name}"/>

<apex:outputField value="{! Opportunity.Amount }"/>

<apex:outputField value="{! Opportunity.CloseDate }"/>

<apex:outputField value="{! Opportunity.Account.Name }"/>

</apex:pageBlock>

</apex:page>

1. **Create a Visualforce form which inserts a basic Contact record**

<apex:page standardController="Contact" Sidebar="False">

<apex:form >

<apex:pageBlock title="Create Contact" >

<apex:pageBlockSection title="Contact" >

<apex:inputField value="{! Contact.FirstName }"/>

<apex:inputField value="{! Contact.LastName}"/>

<apex:inputField value="{! Contact.Email }"/>

</apex:pageBlockSection>

<apex:pageBlockButtons >

<apex:commandButton action="{! save }" value="Save" />

</apex:pageBlockButtons>

</apex:pageBlock>

</apex:form>

</apex:page>

1. **Create a Visualforce page that shows a list of Accounts linked to their record pages**

<apex:page standardController="Account" recordSetVar="Accounts" >

<apex:pageBlock title="Accounts">

<apex:repeat var="a" value="{!Accounts}" rendered="true" id="account\_list">

<li>

<apex:outputLink value="/{!a.ID}" >

<apex:outputText value="{!a.Name}"/>

</apex:outputLink>

</li>

</apex:repeat>

</apex:pageBlock>

</apex:page>

4. **Use a static resource to display an image on a Visualforce Page**

<apex:page>

<apex:image url="{!URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg')}" />

</apex:page>

**5.Create a Visualforce page that displays new cases**

<apex:page controller="NewCaseListController">

<apex:pageBlock title="new Case List" id="cases\_list">

<li>

<apex:repeat var="case" value="{!newCases}" rendered="true" id="rCases">

<p><apex:outputLink value="/{!case.ID}">{!case.CaseNumber}</apex:outputLink></p>

</apex:repeat>

</li>

</apex:pageBlock>

</apex:page>

public class NewCaseListController {

public List<Case> getNewCases() {

List<Case> results = [SELECT CaseNumber FROM Case WHERE status='New'];

return results;

}

}