**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');

Account result = [SELECT Id, Name, (SELECT Id, Name from contacts) from Account where Id=:accountId];

return result;

}

}

**AccountManagerTest**

@IsTest

private class AccountManagerTest {

@isTest static void testGetContactsByAccountId() {

Id recordId = createTestRecord();

// Set up a test request

RestRequest request = new RestRequest();

request.requestUri =

'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts';

request.httpMethod = 'GET';

RestContext.request = request;

// 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 accountTest = new Account(

Name='Test record');

insert accountTest;

Contact contactTest = new Contact(

FirstName = 'John',

LastName = 'Doe',

AccountId = accountTest.Id);

insert contactTest;

return accountTest.Id;

}

}

**AccountProcessor**

public class AccountProcessor {

@future

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

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

List<Account> account = [Select Id, Name, (Select Id from Contacts) from Account Where Id in :accountIds];

For(Account acc:account){

List<Contact> contactList =acc.Contacts;

acc.Number\_of\_contacts\_\_c = contactList.size();

accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

**AccountProcessorTest**

@IsTest

public class AccountProcessorTest {

@IsTest

private static void testCountContacts(){

Account newAccount = new Account(Name='Test Account');

insert newAccount;

Contact newContact1 =new Contact(FirstName='John',LastName='Doe',AccountId = newAccount.Id);

insert newContact1;

Contact newContact2 =new Contact(FirstName='John',LastName='Doe',AccountId = newAccount.Id);

insert newContact2;

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

accountIds.add(newAccount.Id);

Test.startTest();

AccountProcessor.countContacts(accountIds);

Test.stopTest();

}

}

**AddPrimaryContact**

public class AddPrimaryContact implements Queueable{

private Contact con;

private String state;

public AddPrimaryContact(Contact con, String state){

this.con = con;

this.state = state;

}

public void execute (QueueableContext context) {

List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)

from Account where BillingState = :state Limit 200];

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

for (Account acc:accounts) {

Contact c = con.clone();

c.AccountId = acc.Id;

primaryContacts.add(c);

}

}

}

**AddPrimaryContactTest**

@isTest

public class AddPrimaryContactTest {

static testmethod void testQueueable() {

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

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

testAccounts.add(new Account (Name='Account '+i, BillingState='CA'));

}

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

testAccounts.add(new Account (Name = ' Account '+j, BillingState= 'NY'));

}

insert testAccounts;

Contact testContact = new Contact(FirstName = 'John', LastName ='Doe');

insert testContact;

AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

Test.startTest();

system.enqueueJob(addit);

Test.stopTest();

system.assertEquals(50,[select count() from Contact where accountId in (select Id from Account where Billingstate='CA')]);

}

}

**AnimalLocator**

public class AnimalLocator {

public static String getAnimalNameById(Integer ID) {

String animal = ' ';

Http http = new Http();

HttpRequest request = new HttpRequest();

String s = string.valueOf(ID);

request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+ ID);

request.setMethod('GET');

HttpResponse response = http.send(request);

Map<String,Object> animals = new Map<String,Object>();

if(response.getStatusCode() == 200) {

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

animals = (Map<String,Object>) results.get('animal');

animal = String.valueOf(animals.get('name'));

} else {

system.debug(response.getBody());

}

Return animal;

}

}

**AnimalLocatorMock**

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

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

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

response.setStatusCode(200);

return response;

}

}

**AnimalLocatorTest**

@isTest

private class AnimalLocatorTest {

@isTest

static void testGetCallout() {

Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

String animalName = AnimalLocator.getAnimalNameById(1);

system.debug('AnimalName : ' + animalName);

System.assertEquals(animalName, 'chicken');

}

}

**AsyncParkService**

public class AsyncParkService {

public class byCountryResponseFuture extends System.WebServiceCalloutFuture {

public String[] getValue() {

ParkService.byCountryResponse response = (ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);

return response.return\_x;

}

}

public class AsyncParksImplPort {

public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';

public Map<String,String> inputHttpHeaders\_x;

public String clientCertName\_x;

public Integer timeout\_x;

private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};

public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation continuation,String arg0) {

ParkService.byCountry request\_x = new ParkService.byCountry();

request\_x.arg0 = arg0;

return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(

this,

request\_x,

AsyncParkService.byCountryResponseFuture.class,

continuation,

new String[]{endpoint\_x,

'',

'http://parks.services/',

'byCountry',

'http://parks.services/',

'byCountryResponse',

'ParkService.byCountryResponse'}

);

}

}

}

**ContactsTodayController**

public class ContactsTodayController {

@AuraEnabled

public static List<Contact> getContactsForToday() {

List<Task> my\_tasks = [SELECT Id, Subject, WhoId FROM Task WHERE OwnerId = :UserInfo.getUserId() AND IsClosed = false AND WhoId != null];

List<Event> my\_events = [SELECT Id, Subject, WhoId FROM Event WHERE OwnerId = :UserInfo.getUserId() AND StartDateTime >= :Date.today() AND WhoId != null];

List<Case> my\_cases = [SELECT ID, ContactId, Status, Subject FROM Case WHERE OwnerId = :UserInfo.getUserId() AND IsClosed = false AND ContactId != null];

Set<Id> contactIds = new Set<Id>();

for(Task tsk : my\_tasks) {

contactIds.add(tsk.WhoId);

}

for(Event evt : my\_events) {

contactIds.add(evt.WhoId);

}

for(Case cse : my\_cases) {

contactIds.add(cse.ContactId);

}

List<Contact> contacts = [SELECT Id, Name, Phone, Description FROM Contact WHERE Id IN :contactIds];

for(Contact c : contacts) {

c.Description = '';

for(Task tsk : my\_tasks) {

if(tsk.WhoId == c.Id) {

c.Description += 'Because of Task "'+tsk.Subject+'"\n';

}

}

for(Event evt : my\_events) {

if(evt.WhoId == c.Id) {

c.Description += 'Because of Event "'+evt.Subject+'"\n';

}

}

for(Case cse : my\_cases) {

if(cse.ContactId == c.Id) {

c.Description += 'Because of Case "'+cse.Subject+'"\n';

}

}

}

return contacts;

}

}

**ContactsTodayControllerTest**

@IsTest

public class ContactsTodayControllerTest {

@IsTest

public static void testGetContactsForToday() {

Account acct = new Account(

Name = 'Test Account'

);

insert acct;

Contact c = new Contact(

AccountId = acct.Id,

FirstName = 'Test',

LastName = 'Contact'

);

insert c;

Task tsk = new Task(

Subject = 'Test Task',

WhoId = c.Id,

Status = 'Not Started'

);

insert tsk;

Event evt = new Event(

Subject = 'Test Event',

WhoId = c.Id,

StartDateTime = Date.today().addDays(5),

EndDateTime = Date.today().addDays(6)

);

insert evt;

Case cse = new Case(

Subject = 'Test Case',

ContactId = c.Id

);

insert cse;

List<Contact> contacts = ContactsTodayController.getContactsForToday();

System.assertEquals(1, contacts.size());

System.assert(contacts[0].Description.containsIgnoreCase(tsk.Subject));

System.assert(contacts[0].Description.containsIgnoreCase(evt.Subject));

System.assert(contacts[0].Description.containsIgnoreCase(cse.Subject));

}

@IsTest

public static void testGetNoContactsForToday() {

Account acct = new Account(

Name = 'Test Account'

);

insert acct;

Contact c = new Contact(

AccountId = acct.Id,

FirstName = 'Test',

LastName = 'Contact'

);

insert c;

Task tsk = new Task(

Subject = 'Test Task',

WhoId = c.Id,

Status = 'Completed'

);

insert tsk;

Event evt = new Event(

Subject = 'Test Event',

WhoId = c.Id,

StartDateTime = Date.today().addDays(-6),

EndDateTime = Date.today().addDays(-5)

);

insert evt;

Case cse = new Case(

Subject = 'Test Case',

ContactId = c.Id,

Status = 'Closed'

);

insert cse;

List<Contact> contacts = ContactsTodayController.getContactsForToday();

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

}

}

**DailyLeadProcessor**

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){

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

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

**DailyLeadProcessorTest**

@isTest

private class DailyLeadProcessorTest{

//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

public static String CRON\_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){

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

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

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');

leads.add(lead);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());

// Stopping the test will run the job synchronously

Test.stopTest();

}

}

**NewCaseListController**

public class NewCaseListController {

public List<Case> getNewCases(){

List<Case> filterList = [Select Id, CaseNumber from case where status = 'New'];

return filterList;

}

}

**PagedResult**

public with sharing class PagedResult {

@AuraEnabled

public Integer pageSize { get; set; }

@AuraEnabled

public Integer pageNumber { get; set; }

@AuraEnabled

public Integer totalItemCount { get; set; }

@AuraEnabled

public Object[] records { get; set; }

}

**ParkLocator**

public class ParkLocator {

public static string[] country(String country) {

parkService.parksImplPort park = new parkService.parksImplPort();

return park.byCountry(country);

}

}

**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

//Double x = 1.0;

//Double result = AwesomeCalculator.add(x, y);

String country = 'Germany';

String[] result = ParkLocator.Country(country);

// Verify that a fake result is returned

System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park', 'Hainich National Park', 'Bavarian Forest National Park'}, result);

}

}

**ParkService**

public class ParkService {

public class byCountryResponse {

public String[] return\_x;

private String[] return\_x\_type\_info = new String[]{'return','http://parks.services/',null,'0','-1','false'};

private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'return\_x'};

}

public class byCountry {

public String arg0;

private String[] arg0\_type\_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};

private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'arg0'};

}

public class ParksImplPort {

public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';

public Map<String,String> inputHttpHeaders\_x;

public Map<String,String> outputHttpHeaders\_x;

public String clientCertName\_x;

public String clientCert\_x;

public String clientCertPasswd\_x;

public Integer timeout\_x;

private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};

public String[] byCountry(String arg0) {

ParkService.byCountry request\_x = new ParkService.byCountry();

request\_x.arg0 = arg0;

ParkService.byCountryResponse response\_x;

Map<String, ParkService.byCountryResponse> response\_map\_x = new Map<String, ParkService.byCountryResponse>();

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

WebServiceCallout.invoke(

this,

request\_x,

response\_map\_x,

new String[]{endpoint\_x,

'',

'http://parks.services/',

'byCountry',

'http://parks.services/',

'byCountryResponse',

'ParkService.byCountryResponse'}

);

response\_x = response\_map\_x.get('response\_x');

return response\_x.return\_x;

}

}

}

**ParkServiceMock**

@isTest

global class ParkServiceMock implements WebServiceMock {

global void doInvoke(

Object stub,

Object request,

Map<String, Object> response,

String endpoint,

String soapAction,

String requestName,

String responseNS,

String responseName,

String responseType) {

// start - specify the response you want to send

parkService.byCountryResponse response\_x = new parkService.byCountryResponse();

response\_x.return\_x = new List<String>{'Hamburg Wadden Sea National Park', 'Hainich National Park', 'Bavarian Forest National Park'};

//calculatorServices.doAddResponse response\_x = new calculatorServices.doAddResponse();

//response\_x.return\_x = 3.0;

// end

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

}

}

**PropertyController**

public with sharing class PropertyController {

private static final Decimal DEFAULT\_MAX\_PRICE = 9999999;

private static final Integer DEFAULT\_PAGE\_SIZE = 9;

/\*\*

\* Endpoint that retrieves a paged and filtered list of properties

\* @param searchKey String used for searching on property title, city and tags

\* @param maxPrice Maximum price

\* @param minBedrooms Minimum number of bedrooms

\* @param minBathrooms Minimum number of bathrooms

\* @param pageSize Number of properties per page

\* @param pageNumber Page number

\* @return PagedResult object holding the paged and filtered list of properties

\*/

@AuraEnabled(cacheable=true)

public static PagedResult getPagedPropertyList(

String searchKey,

Decimal maxPrice,

Integer minBedrooms,

Integer minBathrooms,

Integer pageSize,

Integer pageNumber

) {

// Normalize inputs

Decimal safeMaxPrice = (maxPrice == null

? DEFAULT\_MAX\_PRICE

: maxPrice);

Integer safeMinBedrooms = (minBedrooms == null ? 0 : minBedrooms);

Integer safeMinBathrooms = (minBathrooms == null ? 0 : minBathrooms);

Integer safePageSize = (pageSize == null

? DEFAULT\_PAGE\_SIZE

: pageSize);

Integer safePageNumber = (pageNumber == null ? 1 : pageNumber);

String searchPattern = '%' + searchKey + '%';

Integer offset = (safePageNumber - 1) \* safePageSize;

PagedResult result = new PagedResult();

result.pageSize = safePageSize;

result.pageNumber = safePageNumber;

result.totalItemCount = [

SELECT COUNT()

FROM Property\_\_c

WHERE

(Name LIKE :searchPattern

OR City\_\_c LIKE :searchPattern

OR Tags\_\_c LIKE :searchPattern)

AND Price\_\_c <= :safeMaxPrice

AND Beds\_\_c >= :safeMinBedrooms

AND Baths\_\_c >= :safeMinBathrooms

];

result.records = [

SELECT

Id,

Address\_\_c,

City\_\_c,

State\_\_c,

Description\_\_c,

Price\_\_c,

Baths\_\_c,

Beds\_\_c,

Thumbnail\_\_c,

Location\_\_Latitude\_\_s,

Location\_\_Longitude\_\_s

FROM Property\_\_c

WHERE

(Name LIKE :searchPattern

OR City\_\_c LIKE :searchPattern

OR Tags\_\_c LIKE :searchPattern)

AND Price\_\_c <= :safeMaxPrice

AND Beds\_\_c >= :safeMinBedrooms

AND Baths\_\_c >= :safeMinBathrooms

WITH SECURITY\_ENFORCED

ORDER BY Price\_\_c

LIMIT :safePageSize

OFFSET :offset

];

return result;

}

/\*\*

\* Endpoint that retrieves pictures associated with a property

\* @param propertyId Property Id

\* @return List of ContentVersion holding the pictures

\*/

@AuraEnabled(cacheable=true)

public static List<ContentVersion> getPictures(Id propertyId) {

List<ContentDocumentLink> links = [

SELECT Id, LinkedEntityId, ContentDocumentId

FROM ContentDocumentLink

WHERE

LinkedEntityId = :propertyId

AND ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')

WITH SECURITY\_ENFORCED

];

if (links.isEmpty()) {

return null;

}

Set<Id> contentIds = new Set<Id>();

for (ContentDocumentLink link : links) {

contentIds.add(link.ContentDocumentId);

}

return [

SELECT Id, Title

FROM ContentVersion

WHERE ContentDocumentId IN :contentIds AND IsLatest = TRUE

WITH SECURITY\_ENFORCED

ORDER BY CreatedDate

];

}

}

**AccountAddressTrigger**

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

for (account account:trigger.new){

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

account.ShippingPostalCode = account.BillingPostalCode;

}

}

}

**ClosedOpportunityTrigger**

trigger ClosedOpportunityTrigger on Opportunity (before 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;

}

}

**RestrictContactByName**

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

For (Contact c : Trigger.New) {

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

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

}

}

}

**CreateDefaultData**

public with sharing class CreateDefaultData{

Static Final String TYPE\_ROUTINE\_MAINTENANCE = 'Routine Maintenance';

//gets value from custom metadata How\_We\_Roll\_Settings\_\_mdt to know if Default data was created

@AuraEnabled

public static Boolean isDataCreated() {

How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();

return customSetting.Is\_Data\_Created\_\_c;

}

//creates Default Data for How We Roll application

@AuraEnabled

public static void createDefaultData(){

List<Vehicle\_\_c> vehicles = createVehicles();

List<Product2> equipment = createEquipment();

List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);

List<Equipment\_Maintenance\_Item\_\_c> joinRecords = createJoinRecords(equipment, maintenanceRequest);

updateCustomSetting(true);

}

public static void updateCustomSetting(Boolean isDataCreated){

How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();

customSetting.Is\_Data\_Created\_\_c = isDataCreated;

upsert customSetting;

}

public static List<Vehicle\_\_c> createVehicles(){

List<Vehicle\_\_c> vehicles = new List<Vehicle\_\_c>();

vehicles.add(new Vehicle\_\_c(Name = 'Toy Hauler RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Toy Hauler RV'));

vehicles.add(new Vehicle\_\_c(Name = 'Travel Trailer RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 2, Bedrooms\_\_c = 2, Model\_\_c = 'Travel Trailer RV'));

vehicles.add(new Vehicle\_\_c(Name = 'Teardrop Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Teardrop Camper'));

vehicles.add(new Vehicle\_\_c(Name = 'Pop-Up Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Pop-Up Camper'));

insert vehicles;

return vehicles;

}

public static List<Product2> createEquipment(){

List<Product2> equipments = new List<Product2>();

equipments.add(new Product2(Warehouse\_SKU\_\_c = '55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement\_Part\_\_c = true,Cost\_\_c = 100 ,Maintenance\_Cycle\_\_c = 100));

equipments.add(new Product2(name = 'Fuse 20B',Replacement\_Part\_\_c = true,Cost\_\_c = 1000, Maintenance\_Cycle\_\_c = 30 ));

equipments.add(new Product2(name = 'Breaker 13C',Replacement\_Part\_\_c = true,Cost\_\_c = 100 , Maintenance\_Cycle\_\_c = 15));

equipments.add(new Product2(name = 'UPS 20 VA',Replacement\_Part\_\_c = true,Cost\_\_c = 200 , Maintenance\_Cycle\_\_c = 60));

insert equipments;

return equipments;

}

public static List<Case> createMaintenanceRequest(List<Vehicle\_\_c> vehicles){

List<Case> maintenanceRequests = new List<Case>();

maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(1).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));

maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(2).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));

insert maintenanceRequests;

return maintenanceRequests;

}

public static List<Equipment\_Maintenance\_Item\_\_c> createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){

List<Equipment\_Maintenance\_Item\_\_c> joinRecords = new List<Equipment\_Maintenance\_Item\_\_c>();

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));

joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));

insert joinRecords;

return joinRecords;

}

}

**CreateDefaultDataTest**

@isTest

private class CreateDefaultDataTest {

@isTest

static void createData\_test(){

Test.startTest();

CreateDefaultData.createDefaultData();

List<Vehicle\_\_c> vehicles = [SELECT Id FROM Vehicle\_\_c];

List<Product2> equipment = [SELECT Id FROM Product2];

List<Case> maintenanceRequest = [SELECT Id FROM Case];

List<Equipment\_Maintenance\_Item\_\_c> joinRecords = [SELECT Id FROM Equipment\_Maintenance\_Item\_\_c];

System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');

System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');

System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');

System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items created');

}

@isTest

static void updateCustomSetting\_test(){

How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();

customSetting.Is\_Data\_Created\_\_c = false;

upsert customSetting;

System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be false');

customSetting.Is\_Data\_Created\_\_c = true;

upsert customSetting;

System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be true');

}

}

**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));

}

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;

}

}

}

**class MaintenanceRequestHelperTest**

@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);

}

}

**WarehouseCalloutService**

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);

}

}

}

}

**WarehouseCalloutServiceMock**

@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;

}

}

**WarehouseCalloutServiceTest**

@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]);

}

}

WarehouseSyncSchedule

global with sharing class WarehouseSyncSchedule implements Schedulable{

global void execute(SchedulableContext ctx){

System.enqueueJob(new WarehouseCalloutService());

}

}

**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 ');

}

}

**trigger MaintenanceRequest**

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

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

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

}

}