Animal Locator:

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
      Http http = new Http();
      HttpRequest req = 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 res = 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');
   }
}
```

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('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
      response.setStatusCode(200);
      return response;
   }
```

```
}
```

Account Manager:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
   @HttpGet
   global static Account getAccount() {
      RestRequest req = RestContext.request;
      String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id = :accId];
      return acc;
   }
}
```

Account Manager Test:

```
@isTest
private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts' ;
      request.httpMethod = 'GET';
      RestContext.request = request;
      // Call the method to test
      Account thisAccount = AccountManager.getAccount();
      // Verify results
      System.assert(thisAccount != null);
```

```
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

Lead Processor :

```
public class LeadProcessor implements Database.Batchable<sObject> {

     public Database.QueryLocator start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
          return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
         // process each batch of records
           for (Lead Lead : leads) {
              lead.LeadSource = 'Dreamforce';
           }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
      }

}
```

Lead Processor Test:

```
@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();
      LeadProcessor leadProcessor = new LeadProcessor();
      Id batchId = Database.executeBatch(leadProcessor);
      Test.stopTest();
   }

}
```

Daily Lead Processor Test:

```
@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());
    }
}
```

Park Locator:

```
public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new parkService.parksImplPort();
        return park.byCountry(country);
    }
}
```

Park Locator Test:

```
@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);
    }
}
```

Park Service Mock :

```
//Generated by wsdl2apex

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;
        }
    }
}
```

Park Service:

```
@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>{'Lal Bhag', 'Cubbon Park', 'Pazhassi Dam'};
            // end
        response.put('response_x', response_x);
    }
}
```

AsyncParkService:

```
//Generated by wsdl2apex

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'}
            );
        }
    }
}
```

Closed Opportunity Trigger:

```
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 SPECIALIST Super Badges Code:

Maintaince Request :

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> caseList) {
List<case> newCases = new List<Case>();
Map<String,Integer> result=getDueDate(caseList);
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case> CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<AggregateResult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
```

```
from Work_Part__c where  Maintenance_Request__r.ID in :caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
}
```

Maintaince Reuest Trigger:

```
trigger MaintenanceRequest on Case (before update, after update) {
// ToDo: Call MaintenanceRequestHelper.updateWorkOrders
if(Trigger.isAfter)
MaintenanceRequestHelper.updateWorkOrders(Trigger.New);
}
```

Synchorinize Salesforce data with an external system:

```
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() {
//ToDo: complete this method to make the callout (using @future) to the
//      REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of products from Http
callout response
```

```
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your org based on the
external ID SKU
}
}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse response)
{
List<Object> externalProducts = (List<Object>)
JSON.deserializeUntyped(response.getBody()); //desrialize the json response
List<Product2> newProducts = new List<Product2>();
for(Object p : externalProducts)
{
Map<String, Object> productMap = (Map<String, Object>) p;
Product2 pr = new Product2();
//Map the fields in the response to the appropriate fields in the Equipment object
pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
pr.Cost__c = (Integer)productMap.get('cost');
pr.Current_Inventory__c = (Integer)productMap.get('quantity');
pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
pr.Warehouse_SKU__c = (String)productMap.get('sku');
pr.ProductCode = (String)productMap.get('_id');
pr.Name = (String)productMap.get('name');
newProducts.add(pr);
}
return newProducts;
}
// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
return response;
}
```

}Copied! Please delete : <code></code> tags

Execute anonymous window with below:

WarehouseCalloutService.runWarehouseEquipmentSync();

Schedule Synchornization:

Warhouse Sync Schedule:

```
global  class WarehouseSyncSchedule implements Schedulable{
// implement scheduled code here
global  void execute (SchedulableContext sc){
WarehouseCalloutService.runWarehouseEquipmentSync();
//optional this can be done by debug mode
String sch = '00 00 01 * * ?';//on 1 pm
System.schedule('WarehouseSyncScheduleTest', sch, new WarehouseSyncSchedule());
}
}
```

and execute anynomous window:

WarehouseSyncSchedule scheduleInventoryCheck();

Test Auotomatic logic:

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

```
@IsTest
private class InstallationTests {
```

```
private static final String STRING_TEST = 'TEST';
private static final String NEW_STATUS = '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 = 'AMC Spirit';
public static String CRON_EXP = '0 0 1 * * ?';
static testmethod void testMaintenanceRequestNegative() {
Vehicle__c vehicle = createVehicle();
insert vehicle;
Id vehicleId = vehicle.Id;
Product2 equipment = createEquipment();
insert equipment;
Id equipmentId = equipment.Id;
Case r = createMaintenanceRequest(vehicleId, equipmentId);
insert r;
Work_Part__c w = createWorkPart(equipmentId, r.Id);
insert w;
Test.startTest();
r.Status = WORKING;
update r;
Test.stopTest();
List<case> allRequest = [SELECT Id
FROM Case];
Work_Part__c workPart = [SELECT Id
FROM Work_Part__c
WHERE Maintenance_Request__c =: r.Id];
System.assert(workPart != null);
System.assert(allRequest.size() == 1);
}
static testmethod void testWarehouseSync() {
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
Test.startTest();
String jobId = System.schedule('WarehouseSyncSchedule',
CRON_EXP,
```

```
new WarehouseSyncSchedule());
CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
FROM CronTrigger
WHERE id = :jobId];
System.assertEquals(CRON_EXP, ct.CronExpression);
System.assertEquals(0, ct.TimesTriggered);
Test.stopTest();
}
private static Vehicle__c createVehicle() {
Vehicle__c v = new Vehicle__c(Name = STRING_TEST);
return v;
}
private static Product2 createEquipment() {
Product2 p = new Product2(Name = STRING_TEST,
Lifespan_Months__c = 10,
Maintenance_Cycle__c = 10,
Replacement_Part__c = true);
return p;
}
private static Case createMaintenanceRequest(Id vehicleId, Id equipmentId) {
Case c = new Case(Type = REPAIR,
Status = NEW_STATUS,
Origin = REQUEST_ORIGIN,
Subject = REQUEST_SUBJECT,
Equipment__c = equipmentId,
Vehicle__c = vehicleId);
return c;
}
private static Work_Part__c createWorkPart(Id equipmentId, Id requestId) {
Work_Part__c wp = new Work_Part__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return wp;
}
}
```

Maintaince Request Helper:

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<case> caseList) {
List<case> newCases = new List<case>();
Map<String,Integer> result=getDueDate(caseList);
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case> CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<aggregateresult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
from Work_Part__c where  Maintenance_Request__r.ID in :caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
```

```
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
}
```

Maintaince Request test:

```
@isTest
public  class MaintenanceRequestTest {
static  List<case> caseList1 = new List<case>();
static List<product2> prodList = new List<product2>();
static List<work_part__c> wpList = new List<work_part__c>();
@testSetup
static void getData(){
caseList1= CreateData( 300,3,3,'Repair');
}
public static List<case>   CreateData( Integer numOfcase, Integer numofProd, Integer
numofVehicle,
String type){
List<case> caseList = new List<case>();
//Create Vehicle
Vehicle__c vc = new Vehicle__c();
vc.name='Test Vehicle';
upsert vc;
//Create Equiment
for(Integer i=0;i<numofProd;i++){
Product2 prod = new Product2();
prod.Name='Test Product'+i;
if(i!=0)
prod.Maintenance_Cycle__c=i;
prod.Replacement_Part__c=true;
prodList.add(prod);
}
upsert  prodlist;
//Create Case
```

```
for(Integer i=0;i< numOfcase;i++){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
if( math.mod(i, 2) ==0)
newCase.Type='Routine Maintenance';
else
newCase.Type='Repair';
newCase.Subject='Routine Maintenance of Vehicle' +i;
newCase.Vehicle__c=vc.Id;
if(i<numofProd)
newCase.Equipment__c=prodList.get(i).ID;
else
newCase.Equipment__c=prodList.get(0).ID;
caseList.add(newCase);
}
upsert caseList;
for(Integer i=0;i<numofProd;i++){
Work_Part__c wp = new Work_Part__c();
wp.Equipment__c   =prodlist.get(i).Id   ;
wp.Maintenance_Request__c=caseList.get(i).id;
wplist.add(wp) ;
}
upsert wplist;
return caseList;
}
public static testmethod void testMaintenanceHelper(){
Test.startTest();
getData();
for(Case cas: caseList1)
cas.Status ='Closed';
update caseList1;
Test.stopTest();
}
}
```

Warehouse CalloutServiceTest:

```
@IsTest
private class WarehouseCalloutServiceTest {
// implement your mock callout test here
@isTest
static void testWareHouseCallout(){
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
}
}
```

Warhouse CalloutServiceMock:

```
@isTest
public class WarehouseCalloutServiceMock implements HTTPCalloutMock {
// implement http mock callout
public HTTPResponse respond (HttpRequest request){
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"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
response.setStatusCode(200);
return response;
}
}
```

WarhouseSync Schedule:

```
@isTest
```

```
private class WarehouseSyncScheduleTest {
public static String CRON_EXP = '0 0 0 15 3 ? 2022';
static testmethod void testjob(){
MaintenanceRequestTest.CreateData( 5,2,2,'Repair');
Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
String joBID= System.schedule('TestScheduleJob', CRON_EXP, new
WarehouseSyncSchedule());
// List<Case> caselist = [Select count(id) from case where case]
Test.stopTest();
}
}
```