

### **AccountAddressTrigger.apxt:**

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

```
    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c==True){
            account.ShippingPostalCode=account.BillingPostalCode;
        }
    }
}
```

### **AccountManager.apxc:**

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

global class AccountManager {

@HttpGet

global static Account getAccount() {

RestRequest req = RestContext.request;

String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

FROM Account WHERE Id = :accId];

return acc;

}

}

### **AccountManagerTest.apxc:**

@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;

}

}

```

#### **AccountProcessor.apxc:**

```

public class AccountProcessor {

    @future

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

        List<Account> accList=[select Id,Number_Of_Contacts__c,(select Id from Contacts) from
        Account where Id in :accountIds];
    }
}

```

```

        For(Account acc:accList){
            acc.Number_Of_Contacts__c=acc.Contacts.size();
        }
        update accList;
    }

}

```

### **AccountProcessorTest.apxc:**

@isTest

```

public class AccountProcessorTest {

    public static testmethod void testAccountProcessor(){
        Account a=new Account();
        a.Name='Test Account';
        insert a;
        Contact con=new Contact();
        con.FirstName='Binary';
        con.LastName='rogramming';
        con.AccountID=a.Id;
        insert con;
        List<Id>accListId=new List<Id>();
        accListId.add(a.id);
        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();
        Account acc=[select Number_Of_Contacts__c from Account where Id=: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c), 1);
    }

}

```

### **AddPrimaryContact.apxc:**

```
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);

        }

        if(primaryContacts.size() >0){

            insert primaryContacts;

        }

    }

}
```

### **AddPrimaryContactTest.apxc:**

```
@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='Jhon',LastName='Doe');
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')]);
}

}

```

#### **AddPrimaryContact.apxc:**

```

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);
}

if(primaryContacts.size() >0){
    insert primaryContacts;
}

}

}

```

#### **AddPrimaryContactTest.apxc:**

@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='Jhon',LastName='Doe');
        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.apxc:**

```

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.apxc:**

```

@Test
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;
    }
}

```

### **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 );

    }

}

```

### **AsyncParkService.apxc:**

//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'}
    );
}
}
}

```

#### **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;
    }
}

```

```
}
```

**ContactsTodayController.apxc:**

```
public class ContactsTodayController {
```

```
    @AuraEnabled
```

```
    public static List<Contact> getContactsForToday() {
```

```
        List<Task> my_tasks = [SELECT Id, Subject, Whold FROM Task WHERE OwnerId =  
:UserInfo.getUserId() AND IsClosed = false AND Whold != null];
```

```
        List<Event> my_events = [SELECT Id, Subject, Whold FROM Event WHERE OwnerId =  
:UserInfo.getUserId() AND StartDateTime >= :Date.today() AND Whold != 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.Whold);
```

```
        }
```

```
        for(Event evt : my_events) {
```

```
            contactIds.add(evt.Whold);
```

```
        }
```

```
        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.Whold == 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.apxc:**

@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',  
    Whold = c.Id,  
    Status = 'Not Started'  
);  
insert tsk;
```

```
Event evt = new Event(  
    Subject = 'Test Event',  
    Whold = 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',  
        Whold = c.Id,  
        Status = 'Completed'  
    );  
    insert tsk;
```

```
    Event evt = new Event(  
        Subject = 'Test Event',  
        Whold = 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.apxc:**

```
global class DailyLeadProcessor implements Schedulable{
```

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

```
        List<lead> leadstoupdate=new List<lead>();
```

```
        List<Lead>leads=[Select id From Lead Where LeadSource=NULL Limit 200];
```

```
        for(Lead l:leads){
```

```
            l.LeadSource='Dreamforce';
```

```
            leadstoupdate.add(l);
```

```
        }
```

```
        update leadstoupdate;
```

```
    }
```

```
}
```

### **DailyLeadProcessorTest.apxc:**

@isTest

```
private class DailyLeadProcessorTest {  
    public static String CRON_EXP='0 0 0 15 3 ? 2023';  
    static testmethod void testScheduledJob(){  
        List<Lead> leads=new List<lead>();  
        for(Integer i=0;i<200;i++){  
            Lead l=new Lead(  
                FirstName='First '+i,LastName='LastName',Company='The Inc');  
            leads.add(l);  
        }  
        insert leads;  
        Test.startTest();  
  
        String jobId=System.schedule('ScheduledApexTest',CRON_EXP,new DailyLeadProcessor());  
        Test.stopTest();  
        List<Lead>checkleads=new List<Lead>();  
        checkleads=[Select Id From Lead Where LeadSource='Dreamforce' and Company='The Inc'];  
        System.assertEquals(200,checkleads.size(),'Leads were not created');  
    }  
}
```

### **LeadProcessor.apxc:**

```
global class LeadProcessor implements Database.Batchable<sObject> {  
    global Integer count = 0;  
    global Database.QueryLocator start(Database.BatchableContext bc){  
        return Database.getQueryLocator('SELECT ID,LeadSource FROM Lead');  
    }  
    global void execute(Database.BatchableContext bc,List<Lead> L_list){
```

```

List<lead>L_list_new=new List<lead>();
for(lead L:L_list){
    L.leadsource='Dreamforce';
    L_list_new.add(L);
    count+=1;
}
update L_list_new;
}
global void finish(Database.BatchableContext bc)
{
    system.debug('count='+count);
}
}

```

#### **LeadProcessorTest.apxc:**

@isTest

```
public class LeadProcessorTest {
```

@isTest

```
public static void testit(){
```

```
    List<lead> L_list=new List<lead>();
```

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

```
        Lead L=new lead();
```

```
        L.LastName='name'+i;
```

```
        L.Company='Company';
```

```
        L.Status='Random Status';
```

```
        L_list.add(L);
```

```
    }
```

```
    insert L_list;
```

```
    Test.startTest();
```



```

        LeadProcessor lp=new LeadProcessor();

        Id batchId=Database.executeBatch(lp);

        Test.stopTest();
    }

}

```

#### **MaintenanceRequest.apxt:**

```

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

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

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

    }

}

```

#### **MaintenanceRequestHelper.apxc:**

```

public with sharing class MaintenanceRequestHelper {

    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap)
    {

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

        For (Case c : updWorkOrders){

            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

                    validIds.add(c.Id);

                }

            }

        }

    }
}

```

```
}
```

```
if (!validIds.isEmpty()){
```

```
    List<Case> newCases = new List<Case>();
```

```
    Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,  
Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM  
Equipment_Maintenance_Items__r)
```

```
FROM Case WHERE Id IN :validIds]);
```

```
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
```

```
    AggregateResult[] results = [SELECT Maintenance_Request__c,  
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c WHERE  
Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];
```

```
for (AggregateResult ar : results){
```

```
    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
```

```
}
```

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

```
    Case nc = new Case (
```

```
        ParentId = cc.Id,
```

```
        Status = 'New',
```

```
        Subject = 'Routine Maintenance',
```

```
        Type = 'Routine Maintenance',
```

```
        Vehicle__c = cc.Vehicle__c,
```

```
        Equipment__c = cc.Equipment__c,
```

```
        Origin = 'Web',
```

```
        Date_Reported__c = Date.Today()
```

```
    );
```

```
    If (maintenanceCycles.containsKey(cc.Id)){
```

```
        nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
```

```

        } else {
            nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
        }

        newCases.add(nc);
    }

    insert newCases;

    List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
    for (Case nc : newCases){
        for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c wpClone = wp.clone();
            wpClone.Maintenance_Request__c = nc.Id;
            ClonedWPs.add(wpClone);

        }
    }
    insert ClonedWPs;
}
}
}
}

```

#### **MaintenanceRequestHelperTest.apxc:**

@istest

```
public with sharing class MaintenanceRequestHelperTest {
```

```
    private static final string STATUS_NEW = 'New';
```

```
    private static final string WORKING = 'Working';
```

```
    private static final string CLOSED = 'Closed';
```

```
private static final string REPAIR = 'Repair';  
private static final string REQUEST_ORIGIN = 'Web';  
private static final string REQUEST_TYPE = 'Routine Maintenance';  
private static final string REQUEST_SUBJECT = 'Testing subject';
```

```
PRIVATE STATIC Vehicle__c createVehicle(){  
    Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');  
    return Vehicle;  
}
```

```
PRIVATE STATIC Product2 createEq(){  
    product2 equipment = new product2(name = 'SuperEquipment',  
        lifespan_months__C = 10,  
        maintenance_cycle__C = 10,  
        replacement_part__c = true);  
    return equipment;  
}
```

```
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){  
    case cs = new case(Type=REPAIR,  
        Status=STATUS_NEW,  
        Origin=REQUEST_ORIGIN,  
        Subject=REQUEST_SUBJECT,  
        Equipment__c=equipmentId,  
        Vehicle__c=vehicleId);  
    return cs;  
}
```

```
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){  
    Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c  
= equipmentId,
```

```

Maintenance_Request__c = requestId);

return wp;
}

@istest
private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                    from case
                    where status =:STATUS_NEW];

```

```

Equipment_Maintenance_Item__c workPart = [select id
                                           from Equipment_Maintenance_Item__c
                                           where Maintenance_Request__c =:newReq.Id];

system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

```

@istest

```

private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
}

```

```
test.stopTest();
```

```
list<case> allRequest = [select id  
                        from case];
```

```
Equipment_Maintenance_Item__c workPart = [select id  
                                           from Equipment_Maintenance_Item__c  
                                           where Maintenance_Request__c = :emptyReq.Id];
```

```
system.assert(workPart != null);
```

```
system.assert(allRequest.size() == 1);
```

```
}
```

```
@istest
```

```
private static void testMaintenanceRequestBulk(){
```

```
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
```

```
    list<Product2> equipmentList = new list<Product2>();
```

```
    list<Equipment_Maintenance_Item__c> workPartList = new  
list<Equipment_Maintenance_Item__c>();
```

```
    list<case> requestList = new list<case>();
```

```
    list<id> oldRequestIds = new list<id>();
```

```
    for(integer i = 0; i < 300; i++){
```

```
        vehicleList.add(createVehicle());
```

```
        equipmentList.add(createEq());
```

```
    }
```

```
    insert vehicleList;
```

```
    insert equipmentList;
```

```
    for(integer i = 0; i < 300; i++){
```

```
        requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
```

```

    }

    insert requestList;

    for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
    }

    insert workPartList;

    test.startTest();
    for(case req : requestList){
        req.Status = CLOSED;
        oldRequestIds.add(req.Id);
    }
    update requestList;
    test.stopTest();

    list<case> allRequests = [select id
                            from case
                            where status =: STATUS_NEW];

    list<Equipment_Maintenance_Item__c> workParts = [select id
                                                    from Equipment_Maintenance_Item__c
                                                    where Maintenance_Request__c in: oldRequestIds];

    system.assert(allRequests.size() == 300);
}
}

```

### **ParkLocator:**

```

public class ParkLocator {

    public static string[] country(string theCountry) {

        ParkService.ParksImplPort parkSvc = 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);
    }
}

```

#### **ParkService.apxc:**

//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;
    }
}

```

```

    }
}
}

```

### **ParkServiceMock.apxc:**

```

@Test
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>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
        // end
        response.put('response_x', response_x);
    }
}

```

### **RandomContactFactory.apxc:**

```

public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer num,String lastName){
        List<Contact> contactList =new List<Contact>();
        for(Integer i=1;i<=num;i++){
            Contact ct=new Contact(FirstNAME='Test'+i,LastName=lastName);
            contactList.add(ct);
        }
    }
}

```

```

    }

    return contactList;
}

}

```

#### **RestrictContactByName.apxt:**

```

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.apxc:**

```

@isTest

public class TestRestrictContactByName {

    @isTest
    public static void testContact(){
        Contact ct=new Contact();
        ct.LastName='INVALIDNAME';
        Database.SaveResult res=Database.insert(ct,false);

        System.assertEquals( 'The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());
    }

}
}

```

### **TestVerifyDate.apxc:**

@isTest

public class TestVerifyDate {

    @isTest static void test1(){

        Date d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));

        System.assertEquals(Date.parse('01/03/2020'), d);

    }

    @isTest static void test2(){

        Date d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));

        System.assertEquals(Date.parse('01/31/2020'), d);

    }

}

### **VerifyDate.apxc:**

public class VerifyDate {

    //method to handle potential checks against two dates

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

        //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month

        if(DateWithin30Days(date1,date2)) {

            return date2;

        } else {

            return SetEndOfMonthDate(date1);

        }

    }

    //method to check if date2 is within the next 30 days of date1

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

```

        //check for date2 being in the past
        if( date2 < date1) { return false; }

        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }

```

```

//method to return the end of the month of a given date
private static Date SetEndOfMonthDate(Date date1) {
    Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
    Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
    return lastDay;
}

```

```

}

```

#### **WarehouseCalloutService.apxc:**

```

public with sharing class WarehouseCalloutService implements Queueable {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

```

//class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.

//The callout's JSON response returns the equipment records that you upsert in Salesforce.

```

@future(callout=true)
public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);

```

```
request.setMethod('GET');
```

```
HttpResponse response = http.send(request);
```

```
List<Product2> warehouseEq = new List<Product2>();
```

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

```
    List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
```

```
    System.debug(response.getBody());
```

```
    //class maps the following fields: replacement part (always true), cost, current inventory,  
    lifespan, maintenance cycle, and warehouse SKU
```

```
    //warehouse SKU will be external ID for identifying which equipment records to update within  
    Salesforce
```

```
    for (Object eq : jsonResponse){
```

```
        Map<String,Object> mapJson = (Map<String,Object>)eq;
```

```
        Product2 myEq = new Product2();
```

```
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
```

```
        myEq.Name = (String) mapJson.get('name');
```

```
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
```

```
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
```

```
        myEq.Cost__c = (Integer) mapJson.get('cost');
```

```
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
```

```
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
```

```
        myEq.ProductCode = (String) mapJson.get('_id');
```

```
        warehouseEq.add(myEq);
```

```
    }
```

```
if (warehouseEq.size() > 0){
```

```
    upsert warehouseEq;
```

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

```
}
```

```
}
```

```
}
```

```
public static void execute (QueueableContext context){  
    runWarehouseEquipmentSync();  
}
```

```
}
```

#### **WarehouseCalloutServiceMock.apxc:**

@isTest

global class WarehouseCalloutServiceMock implements HttpCalloutMock {

// implement http mock callout

global static HttpResponse respond(HttpRequest request){

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

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

// Create a fake response

HttpResponse response = new HttpResponse();

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

response.setBody('{"\_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":  
"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}');

response.setStatusCode(200);

return response;

```
}
```

```
}
```

#### **WarehouseCalloutServiceTest.apxc:**

@isTest

private class WarehouseCalloutServiceTest {



```

@isTest

static void testWareHouseCallout(){

    Test.startTest();

    // implement mock callout test here

    Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());

    WarehouseCalloutService.runWarehouseEquipmentSync();

    Test.stopTest();

    System.assertEquals(1, [SELECT count() FROM Product2]);

}
}

```

#### **WarehouseSyncSchedule.apxc:**

```

global with sharing class WarehouseSyncSchedule implements Schedulable{

    global void execute(SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}

```

#### **WarehouseSyncScheduleTest.apxc:**

```

@isTest

public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){

        String scheduleTime = '00 00 01 * * ?';

        Test.startTest();

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

        String jobId=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());

        Test.stopTest();

        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX
systems.

        // This object is available in API version 17.0 and later.

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

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

```

}

}