AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert, before update) {
   for(Account account : Trigger.new){
      if((account.Match_Billing_Address__c == true) && (account.BillingPostalCode != NULL)){
         account.ShippingPostalCode = account.BillingPostalCode;
      }
   }
}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
   @HttpGet
   global static Account getAccount() {
      RestRequest request = RestContext.request;
      // grab the caseId from the end of the URL
      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.apxc

```
@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.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 = 'Programming';
        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 = '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.apxc

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer animalId) {
        String animalName;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+animalId);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
```

```
        if(response.getStatusCode() == 200) {
          Map<String, Object> r = (Map<String, Object>)
             JSON.deserializeUntyped(response.getBody());

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

           animalName = string.valueOf(animal.get('name'));
        }
        return animalName;
    }
}
```

AnimalLocatorMock.apxc

```
@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.apxc

```
@isTest
private class AnimalLocatorTest{
@isTest static void getAnimalNameByIdTest() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    String response = AnimalLocator.getAnimalNameById(1);

    // Verify that the response received contains fake values
    System.assertEquals('chicken', response);
}
}
```

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'}
            );
        }
    }
}
```

CaseManager.apxc

```
@RestResource(urlMapping='/Cases/*')
global with sharing class CaseManager {
    @HttpGet
    global static Case getCaseById() {
        RestRequest request = RestContext.request;
```

```
    // grab the caseId from the end of the URL
    String caseId = request.requestURI.substring(
      request.requestURI.lastIndexOf('/')+1);
    Case result =  [SELECT CaseNumber,Subject,Status,Origin,Priority
                    FROM Case
                    WHERE Id = :caseId];
    return result;
}
@HttpPost
global static ID createCase(String subject, String status,
    String origin, String priority) {
    Case thisCase = new Case(
        Subject=subject,
        Status=status,
        Origin=origin,
        Priority=priority);
    insert thisCase;
    return thisCase.Id;
}
@HttpDelete
global static void deleteCase() {
    RestRequest request = RestContext.request;
    String caseId = request.requestURI.substring(
        request.requestURI.lastIndexOf('/')+1);
    Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
    delete thisCase;
}
@HttpPut
global static ID upsertCase(String subject, String status,
    String origin, String priority, String id) {
```

```
        Case thisCase = new Case(
                Id=id,
                Subject=subject,
                Status=status,
                Origin=origin,
                Priority=priority);
        // Match case by Id, if present.
        // Otherwise, create new case.
        upsert thisCase;
        // Return the case ID.
        return thisCase.Id;
    }
    @HttpPatch
    global static ID updateCaseFields() {
        RestRequest request = RestContext.request;
        String caseId = request.requestURI.substring(
            request.requestURI.lastIndexOf('/')+1);
        Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
        // Deserialize the JSON string into name-value pairs
        Map<String, Object> params = (Map<String,
Object>)JSON.deserializeUntyped(request.requestbody.tostring());
        // Iterate through each parameter field and value
        for(String fieldName : params.keySet()) {
            // Set the field and value on the Case sObject
            thisCase.put(fieldName, params.get(fieldName));
        }
        update thisCase;
        return thisCase.Id;
    }
}
```

CaseManagerTest.apxc

```
@IsTest
private class CaseManagerTest {
    @isTest static void testGetCaseById() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
            + recordId;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Case thisCase = CaseManager.getCaseById();
        // Verify results
        System.assert(thisCase != null);
        System.assertEquals('Test record', thisCase.Subject);
    }
    @isTest static void testCreateCase() {
        // Call the method to test
        ID thisCaseId = CaseManager.createCase(
            'Ferocious chipmunk', 'New', 'Phone', 'Low');
        // Verify results
        System.assert(thisCaseId != null);
        Case thisCase = [SELECT Id,Subject FROM Case WHERE Id=:thisCaseId];
        System.assert(thisCase != null);
        System.assertEquals(thisCase.Subject, 'Ferocious chipmunk');
    }
    @isTest static void testDeleteCase() {
```

```
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
        + recordId;
    request.httpMethod = 'DELETE';
    RestContext.request = request;
    // Call the method to test
    CaseManager.deleteCase();
    // Verify record is deleted
    List<Case> cases = [SELECT Id FROM Case WHERE Id=:recordId];
    System.assert(cases.size() == 0);
}
@isTest static void testUpsertCase() {
    // 1. Insert new record
    ID case1Id = CaseManager.upsertCase(
            'Ferocious chipmunk', 'New', 'Phone', 'Low', null);
    // Verify new record was created
    System.assert(Case1Id != null);
    Case case1 = [SELECT Id,Subject FROM Case WHERE Id=:case1Id];
    System.assert(case1 != null);
    System.assertEquals(case1.Subject, 'Ferocious chipmunk');
    // 2. Update status of existing record to Working
    ID case2Id = CaseManager.upsertCase(
            'Ferocious chipmunk', 'Working', 'Phone', 'Low', case1Id);
    // Verify record was updated
    System.assertEquals(case1Id, case2Id);
    Case case2 = [SELECT Id,Status FROM Case WHERE Id=:case2Id];
    System.assert(case2 != null);
```

```
    System.assertEquals(case2.Status, 'Working');
}
@isTest static void testUpdateCaseFields() {
    Id recordId = createTestRecord();
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
        + recordId;
    request.httpMethod = 'PATCH';
    request.addHeader('Content-Type', 'application/json');
    request.requestBody = Blob.valueOf('{"status": "Working"}');
    RestContext.request = request;
    // Update status of existing record to Working
    ID thisCaseId = CaseManager.updateCaseFields();
    // Verify record was updated
    System.assert(thisCaseId != null);
    Case thisCase = [SELECT Id,Status FROM Case WHERE Id=:thisCaseId];
    System.assert(thisCase != null);
    System.assertEquals(thisCase.Status, 'Working');
}
// Helper method
static Id createTestRecord() {
    // Create test record
    Case caseTest = new Case(
        Subject='Test record',
        Status='New',
        Origin='Phone',
        Priority='Medium');
    insert caseTest;
    return caseTest.Id;
```

```
    }
}
```

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, 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.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',
            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.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 testSchedulejob(){
      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('ScheduledAppexTest',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));
            }

            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.apxc

```
public class ParkLocator {
    public static List<String> country(String country){
        ParkService.ParksImplPort parkservice =
            new parkService.ParksImplPort();
        return parkservice.byCountry(country);
    }
}
```

ParkLocatorTest.apxc

```
@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
      String country = 'United States';
      List<String> result = ParkLocator.country(country);
      List<String> parks = new List<String>();
            parks.add('Yosemite');
            parks.add('Yellowstone');
            parks.add('Another Park');
      // Verify that a fake result is returned
      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

```
@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
      List<String> parks = new List<string>();
            parks.add('Yosemite');
```

```
            parks.add('Yellowstone');
            parks.add('Another Park');
        ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();
        response_x.return_x = parks;
        // 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('01/01/2022'),Date.parse('01/03/2022'));
      System.assertEquals(Date.parse('01/03/2022'), d);
   }

   @istest static void test2(){
      Date d =VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
      System.assertEquals(Date.parse('01/31/2022'), 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 {

    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.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 class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest.apxc

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
```

```
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime,
new WarehouseSyncSchedule());
        Test.stopTest();
        //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 ');



    }
}
```