APEX SPECIALIST SUPER BADGE

2. **Automate record creation**

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()){
         Map<Id,Case> closedCases = 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'));
    }

    List<Case> newCases = new List<Case>();
    for(Case cc : closedCases.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> clonedList = new
List<Equipment_Maintenance_Item__c>();
```

```
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

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

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable, Database.AllowsCallouts{
  public List<product2> equip = new List<product2>();
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  public void execute(QueueableContext context) {
    Http h = new Http();
    HttpRequest httpReq = new HttpRequest();
    httpReq.setMethod('GET');
    httpReq.setHeader('Content-Type','application/json');
    httpReq.setEndpoint(WAREHOUSE_URL);
    HttpResponse res = h.send(httpReq);
    List<Object> results = (List<Object>) JSON.deserializeUntyped(res.getBody());
    System.debug(results.size());
    for(Object fld : results){
```

```
            Map<String,Object> entry = (Map<String,Object>)fld;
            equip.add(new product2(
                Warehouse_SKU__c = String.valueOf(entry.get('_id')+''),
                Cost__c = Decimal.valueOf(entry.get('cost')+''),
                Lifespan_Months__c = Decimal.valueOf(entry.get('lifespan')+'') ,
                Maintenance_Cycle__c = Decimal.valueOf(entry.get('maintenanceperiod')+''),
                Name = String.valueOf(entry.get('name')+''),
                QuantityUnitOfMeasure = String.valueOf(entry.get('quantity')+'') ,
                Replacement_Part__c = Boolean.valueOf(entry.get('replacement') +''),
                StockKeepingUnit = String.valueOf(entry.get('sku')+'')
            ));
        }
        if(!equip.isEmpty())
        {
            upsert equip Warehouse_SKU__c;
            system.debug('list got updated. Size: '+equip.size());
        }
    }
}

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

4. **Schedule synchronization**

MaintenanceRequestHelperTest:

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

5. **Test automation logic**

MaintenanceRequestHelperTest:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id equipmentId,id
        requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
            Equipment_Maintenance_Item__c(
           Equipment__c = equipmentId,
```

```
        Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}
@isTest
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;
    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;
    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
        createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
    update createdCase;
    test.stopTest();
    Case newCase = [Select id,
                    subject,
                    type,
                    Equipment__c,
                    Date_Reported__c,
                    Vehicle__c,
                    Date_Due__c
                   from case
```

```
                where status ='New'];
    Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newCase.Id];
    list<case> allCase = [select id from case];
    system.assert(allCase.size() == 2);
    system.assert(newCase != null);
    system.assert(newCase.Subject != null);
    system.assertEquals(newCase.Type, 'Routine Maintenance');
    SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}
@isTest
private static void testNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;
    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;
    Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(equipmentId,
        createdCase.Id);
    insert workP;
    test.startTest();
    createdCase.Status = 'Working';
```

```
        update createdCase;
        test.stopTest();
        list<case> allCase = [select id from case];
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                        from Equipment_Maintenance_Item__c
                                        where Maintenance_Request__c = :createdCase.Id];
        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }
    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
            list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();
        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;
        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert caseList;
        for(integer i = 0; i < 300; i++){
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,
```

```
            caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;
        test.startTest();
        for(case cs : caseList){
           cs.Status = 'Closed';
           oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();
        list<case> newCase = [select id
                              from case
                              where status ='New'];
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldCaseIds];
        system.assert(newCase.size() == 300);
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

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()){
    Map<Id,Case> closedCases = 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'));
    }
    List<Case> newCases = new List<Case>();
    for(Case cc : closedCases.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()
```

```
            );
            newCases.add(nc);
        }
        insert newCases;
        List<Equipment_Maintenance_Item__c> clonedList = new
          List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
          closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
      }
   }
}
```

MaintenanceRequest.apxt

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

6. **Test callout logic**

WarehouseCalloutServiceMock.apxc

```
@istest
```

```
global class WarehouseCalloutServiceMock implements HttpCalloutMock{
    global HttpResponse respond(HttpRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":true,"quantity":5,"name":"Gen
            erator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"220000"}]');
        response.setStatusCode(200);
        return response;
    }

}
```

WarehouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
    @isTest static void mainTest(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        Test.startTest();
        Id jobID = System.enqueueJob(new WarehouseCalloutService());
        Test.stopTest();
        AsyncApexJob aaj = [SELECT Id, Status, NumberOfErrors FROM AsyncApexJob WHERE Id = :jobID];
        System.assertEquals('Completed',aaj.status);
        System.assertEquals(0, aaj.NumberOfErrors);
    }
}
```

7. **Test scheduling logic**

WarehouseSyncSchedule:

```
global with sharing class WarehouseSyncSchedule implements Schedulable {
    global void execute (SchedulableContext ctx){
```

```
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

WarehouseSyncScheduleTest:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
            WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');
        Test.stopTest();
    }
}
```

APEX TRIGGERS:

GET STARTED WITH APEX TRIGGERS:

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account alice : Trigger.New) {
        if (alice.Match_Billing_Address__c == true) {
          alice.ShippingPostalCode = alice.BillingPostalCode;
        }
    }
}
```

BULK APEX TRIGGERS:

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
   for (Opportunity o :[SELECT Id,Name FROM Opportunity
                        WHERE Id IN :Trigger.New]){
      taskList.add(new Task(Subject='Follow Up Test Task',
                            WhatId=o.Id,
                            Status='Not Started',
                            Priority='Normal'));
   }
   if (taskList.size() > 0){
      insert taskList;
   }
}
```

APEX TESTING:

GET STARTED WITH APEX UNIT TESTS:

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
   @isTest static void testCheckDates() {
      Date now = Date.today();
      Date lastOfTheMonth = Date.newInstance(now.year(), now.month(), Date.daysInMonth(now.year(),
now.month()));
      Date plus60 = Date.today().addDays(60);
                        Date d1 = VerifyDate.CheckDates(now, now);
```

```
        System.assertEquals(now, d1);
        Date d2 = VerifyDate.CheckDates(now, plus60);
        System.assertEquals(lastOfTheMonth, d2);
    }
}
```

TEST APEX TRIGGERS:

Code:

TestRestrictContactByName.apxc

```
@isTest
private  class TestRestrictContactByName {
        @isTest static void InvalidName() {
                Contact con = new Contact(LastName='INVALIDNAME');
                Test.startTest();
                Database.SaveResult result = Database.insert(con);
                Test.stopTest();
                System.assert(!result.isSuccess());
                System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
                System.debug('Test Result: ' + result.getErrors()[0].getMessage());
        }
        @isTest static void ValidName() {
                Contact con = new Contact(LastName='Jones');
                Test.startTest();
                Database.SaveResult result = Database.insert(con);
                Test.stopTest();
                System.assert(result.isSuccess());
        }
}
```

CREATE TEST DATA FOR APEX TESTS:

RandomContactFactory

```
//@isTest
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();
        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }
}
```

ASYNCHRONOUS APEX

USE FUTURE METHODS:

AccountProcessor

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds) {
        List<Account> accounts = [SELECT Id,
                                  Name,
                                  Number_of_Contacts__c,
                                  (
                                    SELECT Contact.Id
```

```
                                FROM Contacts
                            )
                        FROM Account
                        WHERE Id in :accountIds];
        for (Account a : accounts) {
            a.Number_of_Contacts__c = a.Contacts.size();
        }
        update accounts;
    }
}
```

AccountProcessorTest:

```
@isTest
private class AccountProcessorTest {
    static TestMethod void myTest() {
        List<Account> accounts = new List<Account>();
        for (Integer i=0; i<100; i++) {
            Account account = new Account();
            account.Name = 'AccountProcessorTest Account ' + i;
            accounts.add(account);
        }
        insert accounts;
        List<Id> accountIds = new List<Id>();
        List<Contact> contacts = new List<Contact>();
        for (Account a : accounts) {
            accountIds.add(a.Id);
            for (Integer i=0; i<5; i++) {
                Contact contact = new Contact();
                contact.FirstName = 'AccountProcessor Test Contact';
                contact.LastName = String.valueOf(i);
```

```
            contact.AccountId = a.Id;
            contacts.add(contact);
        }
    }
    insert contacts;
    Test.startTest();
    AccountProcessor.countContacts(accountIds);
    Test.stopTest();
    List<Account> results = [SELECT Id, Number_of_Contacts__c
                             FROM Account
                            WHERE Id in :accountIds];
    for (Account a : results) {
        System.AssertEquals(5, a.Number_of_Contacts__c);
    }
  }
}
```

USE BATCH APEX:

LeadProcessor.apxc

```
public class LeadProcessor implements Database.Batchable<sObject>, Database.Stateful {
  public LeadProcessor() {
  }
  public Database.QueryLocator start(Database.BatchableContext BC) {
    String query = 'SELECT Id FROM Lead';
    return Database.getQueryLocator (query);
  }
  public void execute(Database.BatchableContext BC, List<Lead> leads) {
    for (Lead l : leads) {
      l.LeadSource  = 'Dreamforce';
    }
```

```
        update leads;
    }
    public void finish(Database.BatchableContext BC) {
    }
}
```

LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    private static User testAdminUser = new User(Id = UserInfo.getUserId());
    static testMethod void LeadProcessorTest() {
        System.runAs(testAdminUser) {
            List<Lead> leads = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                leads.add(new Lead(LastName = 'Yoshikawa', Company = 'T.Yoshikawa Labs'));
            }
            insert leads;
            System.assertEquals(leads.size(), 200);
            Test.startTest();
            LeadProcessor batchable = new LeadProcessor();
            Database.executeBatch(batchable);
            Test.stopTest();
            List<Lead> results = [SELECT Id,LeadSource FROM Lead];
            for (Lead l : results) {
                System.assertEquals(l.LeadSource, 'Dreamforce');
            }
            System.assertEquals(results.size(), 200);
        }
    }
```

```
}
```

CONTROL PROCESSES WITH QUEUEABLE APEX:

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];
        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;
            lstOfConts.add(conInst);
        }
        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
```

```
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
        INSERT lstOfAcc;
    }
    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');
        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();
        System.assertEquals(50, [select count() from Contact]);
    }
}
```

SCHEDULE JOBS USING THE APEX SCHEDULER:

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {
        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = '' LIMIT 200];
        for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
   @isTest
   public static void testDailyLeadProcessor(){
      List<Lead> leads = new List<Lead>();
      for (Integer x = 0; x < 200; x++) {
         leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
      }
      insert leads;
      Test.startTest();
      String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());
      Test.stopTest();
      List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT
200];
      System.assertEquals(200, listResult.size());

   }
}
```

APEX INTEGRATION SERVICES

Apex REST Callouts

AnimalLocator.apxc

```
public class AnimalLocator {
        public class cls_animal {
                public Integer id;
                public String name;
                public String eats;
                public String says;
        }
```

```
public class JSONOutput{
        public cls_animal animal;
}
   public static String getAnimalNameById (Integer id) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      system.debug('response: ' + response.getBody());
      jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(), jsonOutput.class);
      return(results.animal.name);
   }
}
```

AnimalLocatorMock.apxc

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPresponse respond(HTTPrequest request) {
      Httpresponse response = new Httpresponse();
      response.setStatusCode(200);
      response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
      return response;
   }
}
```

AnimalLocatorTest.apxc

```
@IsTest
public class AnimalLocatorTest {
   @isTest
   public static void testAnimalLocator() {
```

```
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}
```

APEX SOAP Callouts

ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorMock.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) {
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
```

```
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;
        response.put('response_x', response_x);
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');
        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

APEX WEB SERVICES

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account  acc = AccountManager.getAccount();
        System.assert(acc != null);
    }
    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;
        return acc.Id;
    }
}
```