Apex basics and database

Get started with apex:

StringArrayTest.apxc

```
public class StringArrayTest {

    public static List<String> generateStringArray(Integer N){
        List<String> TestList=new List<String>();
        for(Integer i=0;i<n;i++)
        {
            TestList.add('Test '+i);
            system.debug(TestList[i]);
        }
        return TestList;
    }
}
```

Manipulate records with DML:

AccountHandler.apxc

```
public class AccountHandler {

    public static Account insertNewAccount(String AccountName){
        try {
             Account newacct=new Account(Name=AccountName);
            insert newacct;
            return newacct;
            } catch (DmlException e) {
                System.debug('A DML exception has occurred: ' +
```

```
                    e.getMessage());
            return null;
        }
    }
}
```

Write SOQL Queries:

ContactSearch.apxc

```
public class ContactSearch {

    public static List<Contact> searchForContacts(String lastName,String
postalCode){

        List<Contact> Contacts =[Select Id,Name from Contact where
                    LastName=:lastName and MailingPostalCode=:postalCode];

        return Contacts;
    }
}
```

Write SOSL Queries:

ContactAndLeadSearch.apxc

```
public class ContactAndLeadSearch {

    public static List<List< sObject>> searchContactsAndLeads(String LastName){
```

```
        List<List< sObject>> ContactLeadist=[Find :lastName IN ALL FIELDS
                                    RETURNING Contact(Name),
                                    Lead(Name)];

        return ContactLeadist;
    }
}
```

Apex Integration Services

Apex REST Callouts

Animallocator.apxc:

```
public class AnimalLocator
{

  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);
         String strResp = '';
          system.debug('******response '+response.getStatusCode());
          system.debug('******response '+response.getBody());
       // If the request is successful, parse the JSON response.
       if (response.getStatusCode() == 200)
       {
           // Deserializes the JSON string into collections of primitive data types.
          Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
           // Cast the values in the 'animals' key as a list
          Map<string,object> animals = (map<string,object>) results.get('animal');
           System.debug('Received the following animals:' + animals );
           strResp = string.valueof(animals.get('name'));
           System.debug('strResp >>>>>>' + strResp );
       }
```

```
        return strResp ;
    }

}
```

AnimalLocaterTest.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);
    }
}
```

AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
         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;
    }
}
```

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;
    }
}
```

ParkLocaterTest.apxc:

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

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

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) {
       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);
    }
}
```

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;
        }
    }
}
```

Apex Testing

Get started with Apex Unit Tests:

TestVerifyDate.apxc:

```
@isTest
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date
D=VerifyDate.CheckDates(Date.parse('01/01/2020'),date.parse('01/05/2020'));
        System.assertEquals(Date.parse('01/01/2020'),D);

    }
    @isTest static void Test_CheckDates_case2(){
        Date
D=VerifyDate.CheckDates(Date.parse('01/01/2020'),date.parse('05/05/2020'));
        System.assertEquals(Date.parse('01/31/2020'),D);
}
    @isTest static void Test_DateWithin30Days_case1(){
        Boolean
flag=VerifyDate.DateWithin30Days(Date.parse('01/01/2020'),Date.parse('12/30/20
19'));
            System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case2(){
        Boolean
flag=VerifyDate.DateWithin30Days(Date.parse('01/01/2020'),Date.parse('02/02/20
20'));
            System.assertEquals(false, flag);
}
```

```
    @isTest static void Test_DateWithin30Days_case3(){
        Boolean
flag=VerifyDate.DateWithin30Days(Date.parse('01/01/2020'),Date.parse('01/15/20
20'));
            System.assertEquals(true, flag);
}
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate=VerifyDate.SetEndOfMonthDate(Date.parse('01/01/2020'));
    }
}
```

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
    @TestVisible 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
    @TestVisible 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;
        }

}
```

Test Apex Triggers:

RestrictContactByName.apxt:

```
trigger RestrictContactByName on Contact (before insert) {

   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 static void Test_insertupdateContact(){
        Contact cnt=new Contact();
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();
        Database.SaveResult result= Database.insert(cnt,false);
        Test.stopTest();

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size()>0);
        System.assertEquals('The Last Name "INVALIDNAME"is not allowed for
DML',result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for apex tests:

RandomContactFactory.apxc:

```
public class RandomContactFactory {

    public static List<Contact>generateRandomContacts(Integer numcnt,string
lastname){
        List<Contact> contacts = new List<Contact>();
```

```
        for(Integer i=0;i<numcnt;i++){
            Contact cnt=new Contact(FirstName='Test'+i,LastName=lastname);
            contacts.add(cnt);
        }
        return contacts;

    }
}
```

Apex Triggers

Get started with apex triggers:

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;
        }
    }
}
```

Bulk Apex Trigger:

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;
    }}
```

Asynchronous Apex

Use Future methods:

AccountProcessor.apxc:

```
public class AccountProcessor {

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

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

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

        for(Account acc:accounts){
            List<Contact> contactList=acc.Contacts;
            acc.Number_Of_Contacts__c=contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

AccountProcessorTest.apxc:

```
@IsTest
private class AccountProcessorTest {

    @IsTest
    private static void testCountContacts(){
        Account newAccount =new Account(Name='Test Account');
```

```
        insert newAccount;

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

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

        List<Id> accountIds=new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();

    }
}
```

Control Processes with queueable apex:

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')]);
  }
}
```

Schedule Jobs Using the apex scheduler:

DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable {

  global void execute(SchedulableContext ctx) {
```

```
    List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

    if(!lList.isEmpty()) {
                for(Lead l: lList) {
                    l.LeadSource = 'Dreamforce';
                }
                update lList;
            }
    }
}
```

DailyLeadProcessorTest.apxc:

```
@isTest
private class DailyLeadProcessorTest {
    static testMethod void testDailyLeadProcessor() {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<Lead>();
        for (Integer i = 0; i < 200; i++) {
            lList.add(new Lead(LastName='Dreamforce'+i,
Company='Test1 Inc.', Status='Open - Not Contacted'));
        }
        insert lList;

        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP,
new DailyLeadProcessor());
    }
}
```

Quick start:visualforce

create a visualforce page:

```
<apex:page >
   Hello
</apex:page>
```

Add a standard controller to the page:

```
<apex:page standardController="Contact">
    <head>
     <meta charset="utf-8" />
     <meta name="viewport" content="width=device-width, initial-scale=1" />
     <title>Quick Start: Visualforce</title>
     <!-- Import the Design System style sheet -->
     <apex:slds />

    </head>
       <body>
       <apex:form>
     <apex:pageBlock title="New Contact">
      <!--Buttons -->
       <apex:pageBlockButtons>
          <apex:commandButton action="{!save}" value="Save"/>
       </apex:pageBlockButtons>
       <!--Input form -->
       <apex:pageBlockSection columns="1">
       <apex:inputField value="{!Contact.Firstname}"/>
```

```
        <apex:inputField value="{!Contact.Lastname}"/>
        <apex:inputField value="{!Contact.Email}"/>
       </apex:pageBlockSection>
      </apex:pageBlock>
      </apex:form>

        </body>
</apex:page>
```

Visualforce Basics

create and edit visualforce pages:

diplayImage(visualforce page):

```
<apex:page showHeader="false" sidebar="false">
    <apex:image url="https://developer.salesforce.com/files/salesforce-developer-network-logo.png"/>
</apex:page>
```

use simple variable and formula:

DisplayUserInfo:

```
<apex:page >
    <apex:pageBlock title="User Status">
        <apex:pageBlockSection columns="1">
            {! $User.FirstName}
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

Use Standard Controllers:

ContactView:

```
<apex:page standardController="Contact">
    <apex:pageBlock title="Contact Summary">
        <apex:pageBlockSection >
```

```
        First Name: {! Contact.FirstName } <br/>
        Last Name: {! Contact.LastName } <br/>
        Owner Email: {! Contact.Owner.Email} <br/>
      </apex:pageBlockSection>
   </apex:pageBlock>
</apex:page>
```

Display Records, Fields, and Tables:

OppView:

```
<apex:page standardController="Opportunity">
        <apex:outputField value="{! Opportunity.Name }"/>
      <apex:outputField value="{! Opportunity.Amount }"/>
      <apex:outputField value="{! Opportunity.CloseDate }"/>
      <apex:outputField value="{! Opportunity.Account.Name}"/>
</apex:page>
```

Input data using forms:

CreateContact:

```
<apex:page standardController="Contact" >
   <apex:form>
       <apex:pageBlock title="Edit Contact">
      <apex:pageBlockSection>
        <apex:inputField value="{! Contact.FirstName }"/>
        <apex:inputField value="{! Contact.LastName }"/>
        <apex:inputField value="{! Contact.Email }"/>
```

```
        </apex:pageBlockSection>
         <apex:pageBlockButtons>
            <apex:commandButton action="{! save }" value="Save" />
        </apex:pageBlockButtons>
       </apex:pageBlock>
    </apex:form>
</apex:page>
```

use standard list controllers:

AccountList:

```
<apex:page standardController="Account" recordSetVar="accounts">
    <apex:repeat var="a" value="{!accounts}">
        <li>
            <apex:outputLink value="/{!a.Id}">
             <apex:outputText value="{!a.Name}"></apex:outputText>
            </apex:outputLink>
        </li>

    </apex:repeat>
</apex:page>
```

use static resources:

ShowImage:

```
<apex:page >
    <apex:image alt="eye" title="eye"
```

```
            url="{!URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg')}"/>
</apex:page>
```

Create & Use Custom Controllers:

newcaselist(visualforce page):

```
<apex:page controller="NewCaseListController">
   <apex:repeat var="case" value="{!newCases}">
      <apex:outputLink value="/{!case.ID}">
         <apex:outputText value="{!case.CaseNumber}">
         </apex:outputText>
      </apex:outputLink>
   </apex:repeat>
</apex:page>
```

newcaselistcontrollers(apex class):

```
public class NewCaseListController {

   public List<Case> getNewCases(){
      List<Case> filterList=[Select Id,CaseNumber from Case Where
status='New'];
      return filterList;
   }
}
```

APEX SPECIALIST SUPERBADGE

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()){
        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;
      }
   }
}
```

Schedule Synchronization Using Apex Code:

WarehouseSyncSchedule.apxc:

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

Synchronize Salesforce data with an external System:

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();
  }

}
```

Test Automation Logic

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);
            }
        }
    }

    //When an existing maintenance request of type Repair or Routine Maintenance is closed,
    //create a new maintenance request for a future routine checkup.
    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>();

        //calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.
        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 multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to
today's date.
            //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;
    }
  }
}
```

MaintenanceRequestHelperTest.apxc:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    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;
    }

    // createMaintenanceRequest
    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;
    }

    // createEquipmentMaintenanceItem
    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);
    }
}
```

MaintenanceRequestHelperTest.apxc:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    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;
    }

    // createMaintenanceRequest
    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;
    }

    // createEquipmentMaintenanceItem
    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(equipment
List.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);
    }
}
```

Test Callout Logic:

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) {

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"qua
ntity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":
"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooli
ng
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d6
6226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceText.apxc:

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
        @isTest
```

```
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}
```

Test Scheduling Logic:

WarehouseSyncSchedule.apxc:

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

WarehouseSyncScheduleTest.apxc:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @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();
    }
}
```