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;
      }
   }
}
```

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;
   }
}
```

Apex testing -1

3) 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;
	}
```

```
}
```

4)TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {

   @isTest static void test1()
   {
        Date d= VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));
      System.assertEquals(Date.parse('01/03/2020'), d);
   }
    @isTest static void test2()
   {
        Date d= VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));
      System.assertEquals(Date.parse('01/31/2020'), d);
   }
}
```

Apex testing -2

5)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');
```

```
                }

            }

}
```

6) 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());
    }
}
```

7)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;

    }
}
```

APEX REST

```
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) {
      // Deserializes the JSON string into collections of primitive data types.
       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;
```

```
    }
}

2)
@isTest
private class AnimalLocatorTest{
@isTest static void getAnimalNameByIdTest() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    String response = AnimalLocator.getAnimalNameById(1);

    System.assertEquals('chicken', response);
}
}

@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;
    }
}
```

ParkLocator.apxc

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

2) 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) {
           List<String> parks = new List<string>();
           parks.add('Yosemite');
           parks.add('Yellowstone');
           parks.add('Another Park');
      // start - specify the response you want to send
```

```
        ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();
        response_x.return_x = parks;
        // end
        response.put('response_x', response_x);
    }
}
```

3) 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);
    }
}
```

Apex Wrbservices:

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.my.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;
    }
}
```

create and edit Visualforce page:

DisplayImage.vfp

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

2)DisplayUserInfo.vfp

```
<apex:page >
   {! $User.FirstName}
</apex:page>
```

3)ContactView.vfp

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

4) OppView.vfp

```
<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>
```

5) CreateContact.vfp

```
<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>
```

6)AccountList.vfp

```
<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>
```

7) ShowImage.vfp

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

8)NewCaseList.vfp

```
<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>
```

NewCaseListController.apxc

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

9)ContactForm.vfp

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

2)

ContactForm.vfp

```
<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>
```

Asynchronux apex

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='Vamsi';
        con.LastName='Krishna';
        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);
    }
}
```

10)LeadProcessor.apxc

```
public class LeadProcessor implements
    Database.Batchable<sObject>{
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead'
        );
    }
    public void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads= new List<Lead>();
```

```
        for (Lead lead : scope) {
            lead.LeadSource='Dreamforce';
           leads.add(lead);
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){



    }
}
```

LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();

        // insert 10 accounts
        for (Integer i=0;i<200;i++) {
          leads.add(new Lead(LastName='Lead '+i, Company='Test Co'));
        }
        insert leads;

    }
   static testmethod void test() {
```

```
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
                        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

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');
     // startTest/stopTest block to force async processes to run
     Test.startTest();
     System.enqueueJob(addit);
```

```
        Test.stopTest();
        // Validate the job ran. Check if record have correct parentId now
        System.assertEquals(50, [select count() from Contact where accountId in (Select Id from Account where BillingState='CA')]);
    }
}
```

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx) {
        List<Lead> leadstoupdate= new List<Lead>();
        List<Lead> leads = [SELECT Id
            FROM Lead
            WHERE LeadSource = NULL Limit 200
            ];
        for(Lead l: leads){
             l.LeadSource= 'Dreamforce';
             leadstoupdate.add(l);
        }
        update leadstoupdate;
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
    public static String CRON_EXP = '0 0 0 15 3 ? 2023';
```

```
    static testmethod void testScheduledJob() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
          Lead l = new Lead(
              FirstName = 'First ' + i,
              LastName = 'LastName',
              Company = 'The Inc'
            );
          leads.add(l);
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('ScheduledApexTest',
            CRON_EXP,
            new DailyLeadProcessor());
        Test.stopTest();
         List<Lead> checkleads = new List<Lead>();
        checkleads = [SELECT Id
            FROM Lead
            WHERE LeadSource='Dreamforce' and Company='The Inc'];
        System.assertEquals(200,
            checkleads.size(),
            'Leads were not created');
    }
}
```

Apex specialist super badge:

1)Automate record creation

MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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;
    }
  }
}
```

Synchronize Salesforce data with an external system

WarehouseCalloutService.cls

```
public with sharing class WarehouseCalloutService implements Queueable {

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

    //Write a 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(){

        System.debug('go into runWarehouseEquipmentSync');

        Http http = new Http();

        HttpRequest request = new HttpRequest();


        request.setEndpoint(WAREHOUSE_URL);

        request.setMethod('GET');

        HttpResponse response = http.send(request);


        List<Product2> product2List = new List<Product2>();

        System.debug(response.getStatusCode());

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

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

            System.debug(response.getBody());


            //class maps the following fields:

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

```
        for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            //cost
            product2.Cost__c = (Integer) mapJson.get('cost');
            //current inventory
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }

        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
```

```
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

step4 Schedule synchronization

WarehouseSyncSchedule.cls

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

step5 Test automation logic:

MaintenanceRequest.cls

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

MaintenanceRequestHelper.cls

```
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.cls

```
@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);
    }
}
```

step6 Test callout logic

WarehouseCalloutService.cls

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';


    //Write a 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(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();


        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
```

```
List<Product2> product2List = new List<Product2>();
System.debug(response.getStatusCode());
if (response.getStatusCode() == 200){
  List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
  System.debug(response.getBody());

  //class maps the following fields:
  //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
  for (Object jR : jsonResponse){
    Map<String,Object> mapJson = (Map<String,Object>)jR;
    Product2 product2 = new Product2();
    //replacement part (always true),
    product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
    //cost
    product2.Cost__c = (Integer) mapJson.get('cost');
    //current inventory
    product2.Current_Inventory__c = (Double) mapJson.get('quantity');
    //lifespan
    product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
    //maintenance cycle
    product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
    //warehouse SKU
    product2.Warehouse_SKU__c = (String) mapJson.get('sku');

    product2.Name = (String) mapJson.get('name');
    product2.ProductCode = (String) mapJson.get('_id');
    product2List.add(product2);
  }
```

```
            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

WarehouseCalloutServiceMock.cls

```
@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,"quantity":5,"name":"Ge
nerator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b61110
0aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf7
```

```
43","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceTest.cls

```
@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);
    }
}
```

step7 test scheduling logic

WarehouseSyncSchedule

****

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

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

WarehouseSyncScheduleTest

****

```
@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 ');


   }
}
```

Process Automation Specialist Super badge:

Automate Leads:

Pinned by wonder studies

wonder studies

3 days ago

```
OR(
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", State)),
LEN(State) <> 2,
NOT(OR(Country ="US",Country ="USA",Country ="United States", ISBLANK(Country)))
)
```

Automate Accounts:ValidationForBilling

****

```
OR(
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", BillingState)),
LEN(BillingState) <> 2,
NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States", ISBLANK(BillingCountry))),
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", ShippingState)),
LEN(ShippingState) <> 2,
```

NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States", ISBLANK(ShippingCountry)))

)

ValidationForType

****

ISCHANGED(Name) && (OR(ISPICKVAL(Type, 'Customer - Direct'), ISPICKVAL(Type, 'Customer - Channel')))

Create Robot Setup Object :

CASE(weekday(Date__c),

1,"Sunday",

2,"Monday",

3,"Tuesday",

4,"Wednesday",

5,"Thusday",

6,"Friday",

7,"Saturday",

Text(weekday(Date__c))

)

Create Sales Process and Validate Opportunities:

if((Amount >1000 && Approved__c = false && ispickval(StageName, "Closed Won")), true, false)