

1) Asynchronous Apex

Account Processor code:

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> aid){
        List<Account> accup = new List<Account>();
        List<Account> acc = [Select Id , Name ,(Select Id from Contacts) from Account
where Id in : aid];
        For(Account acc1: acc){
            List<Contact> con = acc1.Contacts;
            acc1.Number_Of_Contacts__c = con.size();
            accup.add(acc1);
        }
        update accup;
    }
}
```

Account Processor Test Code:

```
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts(){
        Account newacc = new Account(Name='Test Account');
        insert newacc;
        Contact newcon = new Contact(FirstName='John',LastName='Doe',AccountId =
newacc.Id);
        insert newcon;
        Contact newcon1 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newacc.Id);
        insert newcon1;
        List<Id> aid = new List<Id>();
        aid.add(newacc.Id);
```

```

        Test.startTest();
        AccountProcessor.countContacts(aid);
        Test.stopTest();
    }
}

```

2) Use Batch Apex

Lead Processor code:

```

global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer c=0;
    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }
    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();
        for(lead L :L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            c+=1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count =' + c);
    }
}

```

Lead processor Test code:

```

public class LeadProcessorTest {
    @isTest
    public static void testit(){

```

```

List<lead> L_list = new List<lead>();
for(Integer i=0;i<200;i++){
    Lead L = new lead();
    L.LastName = 'name' + i ;
    L.Company = 'Company' ;
    L.Status = 'Random Status' ;
    L_list.add(L);
}
insert L_list;
Test.startTest();
LeadProcessor lp = new LeadProcessor();
Id batchId = DataBase.executeBatch(lp);
Test.stopTest();
}
}

```

(3) control process with queueable apex

AddPrimaryContact code:

```

public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;
    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
    }
}

```

```

    }
    insert c_lst;
}
}

```

AddPrimary contact test code:

```

@Test
public class AddPrimaryContactTest {
    @Test
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }
}

```

(4) Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor Code :

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = "];
        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();
            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }
            update newLeads;
        }
    }
}
```

DailyLeadProcessorTest Code :

```
@isTest
private class DailyLeadProcessorTest{
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();
        for(Integer i = 0; i < 200; i++){
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = "", Company = 'Test
Company ' + i, Status = 'Open - Not Contacted');
            leads.add(lead);
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
        Test.stopTest();
    }
}
```

(5)

```
public class AnimalsCallouts {
    public static HttpResponse makeGetCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('GET');
        HttpResponse response = http.send(request);

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

            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

            List<Object> animals = (List<Object>) results.get('animals');
            System.debug('Received the following animals:');
            for(Object animal: animals) {
                System.debug(animal);
            }
        }
        return response;
    }
    public static HttpResponse makePostCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json;charset=UTF-8');
        request.setBody('{"name":"mighty moose"}');
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if(response.getStatusCode() != 201) {
            System.debug('The status code returned was not expected: ' +
                response.getStatusCode() + ' ' + response.getStatus());
        } else {
            System.debug(response.getBody());
        }
    }
}
```

```

        return response;
    }
}
@isTest
private class AnimalsCalloutsTest {
    @isTest static void testGetCallout() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null,result, 'The callout returned a null response.');
```

// Verify status code

```

        System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
```

// Verify content type

```

        System.assertEquals('application/json;charset=UTF-8',
            result.getHeader('Content-Type'),
            'The content type value is not expected.');
```

// Verify the array contains 3 items

```

        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
```

System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');

```

    }
}

```

(6)

```

Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('GET');

```

```

HttpResponse response = http.send(request);
// If the request is successful, parse the JSON response.
if(response.getStatusCode() == 200) {
    // Deserialize 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
    List<Object> animals = (List<Object>) results.get('animals');
    System.debug('Received the following animals:');
    for(Object animal: animals) {
        System.debug(animal);
    }
}

@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{ "animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
        response.getStatusCode(200);
        return response;
    }
}

```

[Apex Integration Services](#)

[Apex REST Callouts](#)

[AnimalLocator Code :](#)

```

public class AnimalLocator {
    public static String getAnimalNameById(Integer x){
        Http http=new Http();
        HttpRequest req=new HttpRequest();
    }
}

```



```

req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' +x);
req.setMethod('GET');
Map<String,Object> animal =new Map<String,Object>();
HttpResponse res=http.send(req);
if(res.getStatusCode()==200){
    Map<String,Object>
results=(Map<String,Object>)JSON.deserializeUntyped(res.getBody());
    animal=(Map<String,Object>) results.get('animal');
}
return (String)animal.get('name');
}
}

```

AnimalLocatorTest Code :

```

@isTest
private class AnimalLocatorTest {
    @isTest static void AnimalsHttpCalloutMock(){
        Test.setMock(HttpCalloutMock.class,new AnimalsHttpCalloutMock());
        string result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult);
    }
}

```

AnimalLocatorMock Code :

```

@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{ "animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
        response.getStatusCode(200);
    }
}

```

```

        return response;
    }
}

```

(8)

ParkLocatorTest Code :

```

@Test
private class ParkLocatorTest {
    @Test static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        //Double x = 1.0;
        //Double result = AwesomeCalculator.add(x, y);
        String country = 'Germany';
        String[] result = ParkLocator.Country(country);
        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park',
'Hainich National Park', 'Bavarian Forest National Park'}, result);
    }
}

```

ParkServiceMock Code :

```

@Test
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,

```

```

        String responseType) {
    // start - specify the response you want to send
    parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
    response_x.return_x = new List<String>{'Hamburg Wadden Sea National Park',
'Hainich National Park', 'Bavarian Forest National Park'};
    //calculatorServices.doAddResponse response_x = new
calculatorServices.doAddResponse();
    //response_x.return_x = 3.0;
    // end
    response.put('response_x', response_x);
}
}

```

ParkLocator Code :

```

public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new parkService.parksImplPort();
        return park.byCountry(country);
    }
}
(9)

```

APEX WEB SERVICES

AccountManager Code :

```

@RestResource(urlMapping={'/Accounts/*/contacts'})
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
            FROM Account WHERE Id = :accId];
        return acc;
    }
}

```

```
}  
}
```

AccountManagerTest Code :

```
@isTest  
private class AccountManagerTest {  
  
    private static testMethod void getAccountTest1() {  
        Id recordId = createTestRecord();  
        // Set up a test request  
        RestRequest request = new RestRequest();  
        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+  
recordId + '/contacts' ;  
        request.httpMethod = 'GET';  
        RestContext.request = request;  
        // Call the method to test  
        Account thisAccount = AccountManager.getAccount();  
        // Verify results  
        System.assert(thisAccount != null);  
        System.assertEquals('Test record', thisAccount.Name);  
  
    }  
  
    // Helper method  
    static Id createTestRecord() {  
        // Create test record  
        Account TestAcc = new Account(  
            Name='Test record');  
        insert TestAcc;  
        Contact TestCon= new Contact(  
            LastName='Test',  
            AccountId = TestAcc.id);  
        return TestAcc.Id;  
    }  
}
```

(10)

APEX SPECIALIST SUPERBADGE :

Automate Record Creation

MaintenanceRequest Code :

```
trigger MaintenanceRequest on Case (before update, after update) {  
  // ToDo: Call MaintenanceRequestHelper.updateWorkOrders  
  if(trigger.isAfter)  
    MaintenanceRequestHelper.updateWorkOrders(trigger.New);  
}
```

MaintenanceRequestHelper Code :

```
public with sharing class MaintenanceRequestHelper {  
  public static void updateWorkOrders(List<Case> caseList) {  
    List<Case> newCases = new List<Case>();  
    Map<String,Integer> result=getDueDate(caseList);  
    for(Case c : caseList){  
      if(c.status=='closed')  
        if(c.type=='Repair' || c.type=='Routine Maintenance'){  
          Case newCase = new Case();  
          newCase.Status='New';  
          newCase.Origin='web';  
          newCase.Type='Routine Maintenance';  
          newCase.Subject='Routine Maintenance of Vehicle';  
          newCase.Vehicle_c=c.Vehicle_c;  
          newCase.Equipment_c=c.Equipment_c;  
          newCase.Date_Reported__c=Date.today();  
          if(result.get(c.Id)!=null)  
            newCase.Date_Due__c=Date.today()+result.get(c.Id);  
          else  
            newCase.Date_Due__c=Date.today();  
          newCases.add(newCase);  
        }  
      }  
    }  
    insert newCases;  
  }  
  //
```

```

public static Map<String,Integer> getDueDate(List<case> CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<AggregateResult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment_r.Maintenance_Cycle_c)cycle
from Work_Part__c where Maintenance_Request__r.ID in :caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
}
(11)

```

Synchronize Salesforce Data

WarehouseCalloutService Code :

```

public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@future(callout=true)
public static void runWarehouseEquipmentSync() {
//ToDo: complete this method to make the callout (using @future) to the
// REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of products from Http
callout response
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your org based on the
external ID SKU
}
}

```

```

}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse response)
{
    List<Object> externalProducts = (List<Object>)
    JSON.deserializeUntyped(response.getBody()); //desrialize the json response
    List<Product2> newProducts = new List<Product2>();
    for(Object p : externalProducts)
    {
        Map<String, Object> productMap = (Map<String, Object>) p;
        Product2 pr = new Product2();
        //Map the fields in the response to the appropriate fields in the Equipment object
        pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
        pr.Cost__c = (Integer)productMap.get('cost');
        pr.Current_Inventory__c = (Integer)productMap.get('quantity');
        pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
        pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
        pr.Warehouse_SKU__c = (String)productMap.get('sku');
        pr.ProductCode = (String)productMap.get('_id');
        pr.Name = (String)productMap.get('name');
        newProducts.add(pr);
    }
    return newProducts;
}

// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    return response;
}
}

WarehouseCalloutService.runWarehouseEquipmentSync();

```

Schedule Synchronization

WarehouseSyncSchedule Code :

```
global class WarehouseSyncSchedule implements Schedulable { global void  
execute(SchedulableContext ctx) {  
WarehouseCalloutService.runWarehouseEquipmentSync(); } }
```

Test Automatic Logic

MaintenanceRequestHelperTest Code :

```
@istest  
public with sharing class MaintenanceRequestHelperTest {  
    private static final string STATUS_NEW = 'New';  
    private static final string WORKING = 'Working';  
    private static final string CLOSED = 'Closed';  
    private static final string REPAIR = 'Repair';  
    private static final string REQUEST_ORIGIN = 'Web';  
    private static final string REQUEST_TYPE = 'Routine Maintenance';  
    private static final string REQUEST_SUBJECT = 'Testing subject';  
    PRIVATE STATIC Vehicle__c createVehicle(){  
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');  
        return Vehicle;  
    }  
    PRIVATE STATIC Product2 createEq(){  
        product2 equipment = new product2(name = 'SuperEquipment',  
            lifespan_months__C = 10,  
            maintenance_cycle__C = 10,  
            replacement_part__c = true);  
        return equipment;  
    }  
    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){  
        case cs = new case(Type=REPAIR,  
            Status=STATUS_NEW,  
            Origin=REQUEST_ORIGIN,  
            Subject=REQUEST_SUBJECT,  
            Equipment__c=equipmentId,  
            Vehicle__c=vehicleId);
```



```

        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item_c wp = new
Equipment_Maintenance_Itemc(Equipment_c = equipmentId,
                                Maintenance_Request__c = requestId);

        return wp;
    }
    @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;
        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;
        Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;
        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();
        Case newReq = [Select id, subject, type, Equipment_c, Date_Reportedc, Vehiclec,
Date_Due_c
                        from case
                        where status =:STATUS_NEW];
        Equipment_Maintenance_Item__c workPart = [select id
                                                    from Equipment_Maintenance_Item__c
                                                    where Maintenance_Request__c =:newReq.Id];
        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    }

```

```

        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }
    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;
        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;
        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,
emptyReq.Id);
        insert workP;
        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();
        list<case> allRequest = [select id
                                from case];
        Equipment_Maintenance_Item__c workPart = [select id
                                                    from Equipment_Maintenance_Item__c
                                                    where Maintenance_Request__c = :emptyReq.Id];
        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }
    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle_C> vehicleList = new list<Vehicle_C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item_c> workPartList = new
list<Equipment_Maintenance_Item_c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();
        for(integer i = 0; i < 300; i++){

```

```

        vehicleList.add(createVehicle());
        equipmentList.add(createEq());
    }
    insert vehicleList;
    insert equipmentList;
    for(integer i = 0; i < 300; i++){
        requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
    }
    insert requestList;
    for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
    }
    insert workPartList;
    test.startTest();
    for(case req : requestList){
        req.Status = CLOSED;
        oldRequestIds.add(req.Id);
    }
    update requestList;
    test.stopTest();
    list<case> allRequests = [select id
        from case
        where status =: STATUS_NEW];
    list<Equipment_Maintenance_Item__c> workParts = [select id
        from Equipment_Maintenance_Item__c
        where Maintenance_Request__c in: oldRequestIds];
    system.assert(allRequests.size() == 300);
}
}

```

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,
Equipmentc, Equipmenttr.Maintenance_Cyclec,(SELECT Id,Equipmentc,Quantityc 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(Equipmenttr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Itemc
WHERE Maintenance_Requestc IN :ValidIds GROUP BY Maintenance_Request_c];
        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
        }
        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
                Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle_c = cc.Vehicle_c,
                Equipment_c =cc.Equipment_c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );
            If (maintenanceCycles.containsKey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            }
        }
    }
}

```

```

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

```

MaintenanceRequest.apxt :-

```

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

```

Test Callout Logic

WarehouseCalloutService Code :

```

public with sharing class WarehouseCalloutService { private static final String
WAREHOUSE_URL = 'https://thsUPERBADGE-apex.herokuapp.com/equipment';
//@future(callout=true)
public static void runWarehouseEquipmentSync(){
    Http http = new Http();
}
}

```

```

HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
List warehouseEq = new List();
if (response.getStatusCode() == 200){
    List jsonResponse = (List)JSON.deserializeUntyped(response.getBody());
    System.debug(response.getBody());
    for (Object eq : jsonResponse){
        Map mapJson = (Map)eq;
        Product2 myEq = new Product2();
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan'); myEq.Cost__c =
        (Decimal) mapJson.get('lifespan');
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
        warehouseEq.add(myEq);
    }
    if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse one');
        System.debug(warehouseEq);
    }
}
}
}
}
}

```

WarehouseCalloutServiceTest Code :

```

@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
    Test.startTest();
    // implement mock callout test here

```

```

Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}

```

WarehouseCalloutServiceMock Code :

```

@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadgeapex.herokuapp.com/equipment',
request.getEndpoint());
System.assertEquals('GET', request.getMethod());
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10 0003"}');
response.setStatusCode(200);
return response;
}
}

```

Test Scheduling Logic

WarehouseSyncSchedule Code :

```

global class WarehouseSyncSchedule implements Schedulable {
global void execute(SchedulableContext ctx) {
WarehouseCalloutService.runWarehouseEquipmentSync();
}
}

```

WarehouseSyncScheduleTest Code :

```
@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 ');
    }
}
```

Apex Triggers :

Get Started with Apex Trigger

AccountAddressTrigger Code :

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for (Account a : Trigger.new) {
        if (a.Match_Billing_Address__c == TRUE){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers Unit

ClosedOpportunityTrigger Code :


```

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List taskList = new List();
    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 Testing

VerifyDate Code :

```

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;
}
}

```

TestVerifyDate Code :

```

@Test
private class TestVerifyDate {
@Test
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 Unit

RestrictContactByName Code :

```

trigger RestrictContactByName on Contact (before insert, before update) {
//check contacts prior to insert or update for invalid data

```

```

For (Contact c : Trigger.New) {
    if(c.LastName == 'INVALIDNAME') {
        //invalidname is invalid
        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
    }
}
}

```

TestRestrictContactByName Code :

```

@isTest
private class TestRestrictContactByName {
    @isTest
    static void invalidName() {
        try { Contact c = new Contact(LastName='INVALIDNAME');
            insert c;
        }
        catch (Exception e) {
            System.assert(true);
        }
    }
}

```

Create Test Data for Apex Tests :

RandomContactFactory Code :

```

public class RandomContactFactory {
    public static List generateRandomContacts(Integer num, String lastName) {
        List contacts = new List();
        for (Integer i = 0; i < num; i++) {
            Contact c = new Contact(FirstName=i.format(), LastName=lastName);
            contacts.add(c);
        }
        return contacts;
    }
}

```

