AccountManager:-

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

AccountManagerTest:-

```
@IsTest
private class AccountManagerTest{
 @isTest static void testAccountManager(){
 Id recordId = getTestAccountId();
 // Set up a test request
 RestRequest request = new RestRequest();
 request.requestUri =
 'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
 request.httpMethod = 'GET';
 RestContext.request = request;

 // Call the method to test
 Account acc = AccountManager.getAccount();

 // Verify results
 System.assert(acc != null);
 }
 private static Id getTestAccountId(){
 Account acc = new Account(Name = 'TestAcc2');
 Insert acc;

 Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
 Insert con;

 return acc.Id;
 }
}
```

AccountProcessor:-

```
public class AccountProcessor
{
 @future
 public static void countContacts(Set<id> setId)
 {
 List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from contacts ) from
account where id in :setId ];
 for( Account acc : lstAccount )
 {
 List<Contact> lstCont = acc.contacts ;

 acc.Number_of_Contacts__c = lstCont.size();
 }
 update lstAccount;
 }
}
```

AccountProcessorTest:-

```
@IsTest
public class AccountProcessorTest {
 public static testmethod void TestAccountProcessorTest()
 {
 Account a = new Account();
 a.Name = 'Test Account';
 Insert a;
 Contact cont = New Contact();
 cont.FirstName ='Bob';
 cont.LastName ='Masters';
 cont.AccountId = a.Id;
 Insert cont;

 set<Id> setAccId = new Set<ID>();
 setAccId.add(a.id);
 Test.startTest();
 AccountProcessor.countContacts(setAccId);
 Test.stopTest();

 Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT 1];
 System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
 }

}
```

AddPrimaryContact:-

```
public class AddPrimaryContact implements Queueable{
 Contact con;
 String state;

 public AddPrimaryContact(Contact con, String state){
 this.con = con;
 this.state = state;
 }
 public void execute(QueueableContext qc){
 List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

 List<Contact> lstOfConts = new List<Contact>();
 for(Account acc : lstOfAccs){
 Contact conInst = con.clone(false,false,false,false);
 conInst.AccountId = acc.Id;

 lstOfConts.add(conInst);
 }

 INSERT lstOfConts;
 }
}
```

AddPrimaryContactTest:-

```
@isTest
public class AddPrimaryContactTest{
 @testSetup
 static void setup(){
 List<Account> lstOfAcc = new List<Account>();
 for(Integer i = 1; i <= 100; i++){
 if(i <= 50)
 lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
 else
 lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
 }

 INSERT lstOfAcc;
 }
 static testmethod void testAddPrimaryContact(){
 Contact con = new Contact(LastName = 'TestCont');
 AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');
```

```
 Test.startTest();
 System.enqueueJob(addPCIns);
 Test.stopTest();

 System.assertEquals(50, [select count() from Contact]);
 }
}
```

AnimalLocator:-

```
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 ;
 }

}
```

AnimalLocatorMock:-

```
@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;
 }
}
```
AnimalLocatorTest:-
```
@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);
 }
}
```
ContactController:-
```
public with sharing class ContactController {
 public Contact c { get; set; }
 public List<Contact> samepage { get; set; }

 public ContactController(){
 c=new Contact();
 }
 public PageReference save() {
 insert c;
 samepage= [select id,FirstName,LastName,Email,Birthdate from Contact where id=:c.id];

 return null;
 }
 }
```
LeadProcessor:-
```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

 // instance member to retain state across transactions
 global Integer recordsProcessed = 0;
 global Database.QueryLocator start(Database.BatchableContext bc) {
 return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
```

```
 }
 global 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';
 // increment the instance member counter
 recordsProcessed = recordsProcessed + 1;

 }
 update leads;
 }
 global void finish(Database.BatchableContext bc){
 System.debug(recordsProcessed + ' records processed. Shazam!');

 }
}
```

LeadProcessorTest:-

```
@isTest
public class LeadProcessorTest {
@testSetup
 static void setup() {
 List<Lead> leads = new List<Lead>();
 // insert 200 leads
 for (Integer i=0;i<200;i++) {
 leads.add(new Lead(LastName='Lead '+i,
 Company='Lead', Status='Open - Not Contacted'));
 }
 insert leads;
 }
 static testmethod void test() {
 Test.startTest();
 LeadProcessor lp = new LeadProcessor();
 Id batchId = Database.executeBatch(lp, 200);
 Test.stopTest();
 // after the testing stops, assert records were updated properly
 System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
 }
}
```

NewCaseListController:-

```
public class NewCaseListController {
 private String val = 'New';
public List<Case> getNewCases() {
 List<Case> results = Database.query(
 'SELECT Id, CaseNumber FROM Case WHERE Status = \" + String.escapeSingleQuotes(val)+'\");
 return results;
}
}
```

ParkLocator:-

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

ParkLocatorTest:-

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

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

ParkService:-

```
public class ParkService {
 public class byCountryResponse {
 public String[] return_x;
 private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-
1','false'};
 private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
 private String[] field_order_type_info = new String[]{'return_x'};
 }
 public class byCountry {
 public String arg0;
 private String[] arg0_type_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
 private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
 private String[] field_order_type_info = new String[]{'arg0'};
```

```
 }
 public class ParksImplPort {
 public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
 public Map<String,String> inputHttpHeaders_x;
 public Map<String,String> outputHttpHeaders_x;
 public String clientCertName_x;
 public String clientCert_x;
 public String clientCertPasswd_x;
 public Integer timeout_x;
 private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
 public String[] byCountry(String arg0) {
 ParkService.byCountry request_x = new ParkService.byCountry();
 request_x.arg0 = arg0;
 ParkService.byCountryResponse response_x;
 Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
 response_map_x.put('response_x', response_x);
 WebServiceCallout.invoke(
 this,
 request_x,
 response_map_x,
 new String[]{endpoint_x,
 '',
 'http://parks.services/',
 'byCountry',
 'http://parks.services/',
 'byCountryResponse',
 'ParkService.byCountryResponse'}
 );
 response_x = response_map_x.get('response_x');
 return response_x.return_x;
 }
 }
}
```

ParkServiceMock:-

```
@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);
 }
}
```

RandomContactFactory:-

```
public class RandomContactFactory {
 public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
 List<Contact> contactList = new List<Contact>();

 for(Integer i=0;i<numContactsToGenerate;i++) {
 Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
 contactList.add(c);
 System.debug(c);
 }
 //insert contactList;
 System.debug(contactList.size());
 return contactList;
 }
}
```

TestRestrictContactByName:-

```
@isTest
private class TestRestrictContactByName {
 @isTest static void testInvalidName() {
 //try inserting a Contact with INVALIDNAME
 Contact myConact = new Contact(LastName='INVALIDNAME');
 insert myConact;

 // Perform test
 Test.startTest();
 Database.SaveResult result = Database.insert(myConact, false);
 Test.stopTest();
```

```
 // Verify
 // In this case the creation should have been stopped by the trigger,
 // so verify that we got back an error.
 System.assert(!result.isSuccess());
 System.assert(result.getErrors().size() > 0);
 System.assertEquals('Cannot create contact with invalid last name.',
 result.getErrors()[0].getMessage());

 }
}
```

TestVerifyDate:-

```
@isTest
private class TestVerifyDate {
 //testing that if date2 is within 30 days of date1, should return date 2
 @isTest static void testDate2within30daysofDate1() {
 Date date1 = date.newInstance(2018, 03, 20);
 Date date2 = date.newInstance(2018, 04, 11);
 Date resultDate = VerifyDate.CheckDates(date1,date2);
 Date testDate = Date.newInstance(2018, 04, 11);
 System.assertEquals(testDate,resultDate);
 }

 //testing that date2 is before date1. Should return "false"
 @isTest static void testDate2beforeDate1() {
 Date date1 = date.newInstance(2018, 03, 20);
 Date date2 = date.newInstance(2018, 02, 11);
 Date resultDate = VerifyDate.CheckDates(date1,date2);
 Date testDate = Date.newInstance(2018, 02, 11);
 System.assertNotEquals(testDate, resultDate);
 }

 //Test date2 is outside 30 days of date1. Should return end of month.
 @isTest static void testDate2outside30daysofDate1() {
 Date date1 = date.newInstance(2018, 03, 20);
 Date date2 = date.newInstance(2018, 04, 25);
 Date resultDate = VerifyDate.CheckDates(date1,date2);
 Date testDate = Date.newInstance(2018, 03, 31);
 System.assertEquals(testDate,resultDate);
 }
}
```

VerifyDate:-

```
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;
}
}
```

MaitenanceRequest.apxt :-

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

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL = 'https://th-
superbadgeapex.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();
}

}
```
WarehouseSyncShedule.apxc :-
```
global with sharing class WarehouseSyncSchedule implements Schedulable{
global void execute(SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService());
}
}
```
MaintenanceRequestHelperTest.apxc :-
```
@istest
public with sharing class MaintenanceRequestHelperTest {

private static final string STATUS_NEW = 'New';
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';
private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';

PRIVATE STATIC Vehicle__c createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
return Vehicle;
}

PRIVATE STATIC Product2 createEq(){
product2 equipment = new product2(name = 'SuperEquipment',
lifespan_months__C = 10,
maintenance_cycle__C = 10,
replacement_part__c = true);
return equipment;
}

PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
case cs = new case(Type=REPAIR,
Status=STATUS_NEW,
```

```
Origin=REQUEST_ORIGIN,
Subject=REQUEST_SUBJECT,
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cs;
}

PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return wp;
}


@istest
private static void testMaintenanceRequestPositive(){
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;

Product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;

case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
insert somethingToUpdate;

Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
insert workP;

test.startTest();
somethingToUpdate.status = CLOSED;
update somethingToUpdate;
test.stopTest();

Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
from case
```

```
where status =:STATUS_NEW];

Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newReq.Id];

system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

@istest
private static void testMaintenanceRequestNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;

product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;

case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
insert emptyReq;

Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
insert workP;

test.startTest();
emptyReq.Status = WORKING;
update emptyReq;
test.stopTest();

list<case> allRequest = [select id
from case];

Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :emptyReq.Id];
```

```
system.assert(workPart != null);
system.assert(allRequest.size() == 1);
}

@istest
private static void testMaintenanceRequestBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>();
list<Product2> equipmentList = new list<Product2>();
list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
list<case> requestList = new list<case>();
list<id> oldRequestIds = new list<id>();

for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle());
equipmentList.add(createEq());
}
insert vehicleList;
insert equipmentList;

for(integer i = 0; i < 300; i++){
requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
}
insert requestList;

for(integer i = 0; i < 300; i++){
workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
}
insert workPartList;

test.startTest();
for(case req : requestList){
req.Status = CLOSED;
oldRequestIds.add(req.Id);
}
update requestList;
test.stopTest();

list<case> allRequests = [select id
from case
```

```
where status =: STATUS_NEW];

list<Equipment_Maintenance_Item__c> workParts = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in: oldRequestIds];

system.assert(allRequests.size() == 300);
}
}
```

MaintenanceRequestHelper.apxc :-

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
Set<Id> validIds = new Set<Id>();


For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
validIds.add(c.Id);

}
}
}

if (!validIds.isEmpty()){
List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE
Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

for (AggregateResult ar : results){
maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
}
```

```
for(Case cc : closedCasesM.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = 'New',
Subject = 'Routine Maintenance',
Type = 'Routine Maintenance',
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = 'Web',
Date_Reported__c = Date.Today()

);

If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
}

newCases.add(nc);
}

insert newCases;

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

}
}
insert ClonedWPs;
}
}
}
MaintenanceRequest.apxt :-
trigger MaintenanceRequest on Case (before update, after update)
{
if(Trigger.isUpdate && Trigger.isAfter)
```

```
{
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
}
}
```

WarehouseCalloutService.apxc :-

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

//@future(callout=true)
public static void runWarehouseEquipmentSync(){

Http http = new Http();
HttpRequest request = new HttpRequest();

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


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

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

for (Object eq : jsonResponse){
Map<String,Object> mapJson = (Map<String,Object>)eq;
Product2 myEq = new Product2();
myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
myEq.Name = (String) mapJson.get('name');
myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
myEq.Cost__c = (Decimal) mapJson.get('lifespan');
myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
warehouseEq.add(myEq);
}

if (warehouseEq.size() > 0){
upsert warehouseEq;
```

```
System.debug('Your equipment was synced with the warehouse one');
System.debug(warehouseEq);
}

}
}
}
```

WarehouseCalloutServiceTest.apxc :-

```
@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
// implement mock callout test here
Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
System.assertEquals('GET', request.getMethod());
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"nam
e":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
response.setStatusCode(200);
return response;
}
}
```

WarehouseSyncSchedule.apxc :-

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

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
String scheduleTime = '00 00 01 * * ?';
Test.startTest();
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
Test.stopTest();
//Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on
UNIX
systems.
// This object is available in API version 17.0 and later.
CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
System.assertEquals(jobID, a.Id,'Schedule ');
}
}
```

APEX TRIGGERS

AccountAddressTrigger

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

ClosedOpportunityTrigger

```
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 TRIGGERS

RestrictContactBYName

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