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 TESTING

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;
 }

}
```

TestVerifyDate:

```
@isTest
public class TestVerifyDate
{
   static testMethod void testMethod1()
   {
      Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
      Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
   }
}
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');
                }

        }
}
```

TestRestrictContactByName:

```
@isTest
public 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());
   }
}
```

RandomContactFactory

```
public class RandomContactFactory {
   public static List<contact> generateRandomContacts(Integer n, string m) {
      List<Contact> con = new List<contact>();
      for(Integer i=1; i<n+1; i++) {
```

```
        Contact c = new Contact(Firstname='test'+i,Lastname=m);
        con.add(c);
      }
      return con;
   }
}
```

Asynchronous Apex

AccountProcessor:

```
public class AccountProcessor{
   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accountsToUpdate = new List<Account>();
      List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account
Where Id in :accountIds];
      For(Account acc:accounts){
         List<Contact> contactList = acc.Contacts;
         acc.Number_Of_Contacts__c = contactList.size();
         accountsToUpdate.add(acc);
      }
      update accountsToUpdate;
   }
}
```

AccountProcessorTest:

```
@IsTest
private class AccountProcessorTest {
   @IsTest
   private static void testCountContacts(){
      Account newAccount = new Account(Name='Test Account');
    insert newAccount;
   Contact newContact1 = new Contact (FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
      insert newContact1;
```

```
    Contact newContact2 = new Contact (FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;
        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
}
```

LeadProcessor:

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 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);
            count += 1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count);
    }
}
```

LeadProcessorTest:

```
@isTest
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();
  }
}
```

AddPrimaryContact :

```
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:

```
@isTest
public class AddPrimaryContactTest {
   static testmethod void testQueueable(){
      List<Account> testAccounts = new List<Account>();
      for(Integer i=0;i<50;i++){
         testAccounts.add(new Account(Name='Account ' +i,BillingState='CA'));
      }
      for(Integer j=0;j<50;j++){
         testAccounts.add(new Account(Name='Account ' +j,BillingState='NY'));
      }
      insert testAccounts;
      Contact testContact = new Contact(FirstName = 'John', LastName = 'Doe');
      insert testContact;
      AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');
      Test.startTest();
      system.enqueueJob(addit);
      Test.stopTest();
      System.assertEquals(50,[Select count() from Contact where accountId in (Select Id
from Account where BillingState='CA')]);
   }

}
```

DailyLeadProcessor:

```
public class DailyLeadProcessor implements Schedulable {
   Public void execute (SchedulableContext SC) {
      List<Lead> LeadObj = [SELECT Id from Lead Where LeadSource = null limit 200 ] ;
      for(Lead l : LeadObj ) {
         l.LeadSource = 'Dreamforce' ;
         update l ;
      }
   }
}
```

DailyLeadProcessorTest:

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

APEX INTEGRATION

AnimalLocator:

```
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:

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}
```

AnimalLocatorMock:

```
@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('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

ParkLocator:

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove
space
        return parkSvc.byCountry(theCountry);
```

```
    }
}
```

ParkLocatorTest:

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

AccountManager:

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where Id=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest:

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
```

```
        request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
            + recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }
    static Id createTestRecord(){
        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;
    }
}
```

APEX SPECIALIST

CreateDefaultData:

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default
data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings_c  customSetting =
How_We_Roll_Settings_c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
```

```
    }
    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords =
createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }


    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings_c  customSetting =
How_We_Roll_Settings_c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle_c> vehicles = new List<Vehicle_c>();
        vehicles.add(new Vehicle_c(Name = 'Toy Hauler RV', Air_Conditionerc = true,
Bathroomsc = 1, Bedroomsc = 1, Model_c = 'Toy Hauler RV'));
        vehicles.add(new Vehicle_c(Name = 'Travel Trailer RV', Air_Conditionerc = true,
Bathroomsc = 2, Bedroomsc = 2, Model_c = 'Travel Trailer RV'));
        vehicles.add(new Vehicle_c(Name = 'Teardrop Camper', Air_Conditionerc = true,
Bathroomsc = 1, Bedroomsc = 1, Model_c = 'Teardrop Camper'));
        vehicles.add(new Vehicle_c(Name = 'Pop-Up Camper', Air_Conditionerc = true,
Bathroomsc = 1, Bedroomsc = 1, Model_c = 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU_c =
```

```
'55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement_Partc =
true,Costc = 100 ,Maintenance_Cycle_c = 100));
        equipments.add(new Product2(name = 'Fuse 20B',Replacement_Part_c = true,Costc
= 1000, Maintenance_Cycle_c = 30  ));
        equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part_c =
true,Costc = 100  , Maintenance_Cycle_c = 15));
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part_c =
true,Costc = 200  , Maintenance_Cycle_c = 60));
        insert equipments;
        return equipments;
    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle_c = vehicles.get(1).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported_c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle_c = vehicles.get(2).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported_c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c>
createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){
        List<Equipment_Maintenance_Item_c> joinRecords = new
List<Equipment_Maintenance_Item_c>();
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
equipment.get(0).Id, Maintenance_Request_c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
equipment.get(1).Id, Maintenance_Request_c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
equipment.get(2).Id, Maintenance_Request_c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
equipment.get(0).Id, Maintenance_Request_c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
equipment.get(1).Id, Maintenance_Request_c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item_c(Equipmentc =
```

```
equipment.get(2).Id, Maintenance_Request_c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest:

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle_c> vehicles = [SELECT Id FROM Vehicle_c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item_c> joinRecords = [SELECT Id FROM
Equipment_Maintenance_Item_c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles
created');
        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment
created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2
maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment
maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings_c  customSetting =
How_We_Roll_Settings_c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
```

```
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings_c.Is_Data_Created_c should be false');

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;
        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings_c.Is_Data_Created_c should be true');

    }
}
```

MaintenanceRequest:

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

MaintenanceRequestHelper:

```
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,
Equipmentc, Equipmentr.Maintenance_Cycle_c,
                                        (SELECT Id,Equipment_c,Quantityc 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.Equipmentr.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:

```
@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);
    }
}
```

WarehouseCalloutService:

```
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:

```
@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":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutServiceTest :

```
@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);
```

```
    }
}
```

WarehouseSyncSchedule:

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

WarehouseSyncScheduleTest:

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