TRIGGERS:
Get Started With Apex Triggers:

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

}
```

Bulk Apex Triggers:

```
  trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> newtsk = new List<Task>();
    if(trigger.IsAfter && (trigger.IsInsert || trigger.IsUpdate)){
    for(Opportunity op:Trigger.New){
        if(op.StageName == 'Closed Won'){
            Task tsk = new Task();
            tsk.Subject = 'Follow Up Test Task';
            tsk.WhatId = op.id;
            newtsk.add(tsk);
            }
            }
    }
    if(newtsk.size()>0){
        insert newtsk;
    }
}
```

TESTING:
Get Started with Apex Unit Tests:
apex class:

```
public class VerifyDate {


        public static Date CheckDates(Date date1, Date date2) {

                if(DateWithin30Days(date1,date2)) {
                        return date2;
```

```
            } else {
                return SetEndOfMonthDate(date1);
            }
        }

        private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }

        Date date30Days = date1.addDays(30);
            if( date2 >= date30Days ) { return false; }
            else { return true; }
        }

        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;
        }

}
unit test:
   @isTest
public class TestVerifyDate {
   @isTest static void testOldDate(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));
      System.assertEquals(date.newInstance(2016, 4, 30), dateTest);
   }

   @isTest static void testLessThan30Days(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(20));
      System.assertEquals(date.today().addDays(20), dateTest);
   }

   @isTest static void testMoreThan30Days(){
      Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(31));
```

```
        System.assertEquals(date.newInstance(2016, 4, 30), dateTest);
    }

}
```

Test Apex Triggers:
trigger:

```
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');
                }

        }



}
```

test:

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {

        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;


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

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
```

```
                    result.getErrors()[0].getMessage());

    }
}
```

Create Test Data for Apex Tests
Apex Class:

```
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;
    }

}
```

Use Future Methods:

```
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;
 }
}
test:
@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);
 }

}



Use Batch Apex:
class:
global class LeadProcessor implements
```

```
Database.Batchable<sObject>, Database.Stateful {


   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){

      List<Lead> leads = new List<Lead>();
      for (Lead lead : scope) {

            lead.LeadSource = 'Dreamforce';

            recordsProcessed = recordsProcessed + 1;

      }
      update leads;
   }

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

   }
test:
@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']);
    }
}
}

Control Processes with Queueable Apex:
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;
    }
```

```
}
TEST
@IsTest
public class AddPrimaryContactTest {

   @IsTest
   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);
   }

}
```

Schedule Jobs Using the Apex Scheduler:

class:

```
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;
        }
    }
}
```

TEST:

```
@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();
```

```
    }
}
```

Apex REST Callouts:
apex class:

```
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 ;
   }
test:
```

```
@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);
   }
}
```

Apex SOAP Callouts

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

test:

```
@isTest
private class ParkLocatorTest {

   @isTest static void testCallout(){
      Test.setMock(WebServiceMock.class,new ParkServiceMock());
      string country ='United States';

      List<string> expectedParks = new List<String>{'Yosemite', 'Sequoia', 'Crater Lake'};
      System.assertEquals(expectedParks,ParkLocator.country(country));

   }

}
```

Apex Web Services

```
@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, FirstName,LastName FROM
Contacts)
                  FROM Account WHERE Id = :accountId];

      return result;
   }
test:
@isTest
private class AccountManagerTest{
   @isTest
   static void testGetAccount(){
      Account a = new Account(Name='TestAccount');
      insert a;
      Contact c= new Contact(AccountId=a.Id,FirstName='Test',LastName='Test');
      insert c;
            // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+ a.id
+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

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

      // Verify results
      System.assert(myAcct!=null);
      System.assertEquals('TestAccount',myAcct.Name);

   }
}
}
```

```
}


}
```

APEX SUPERBADGES CODING :

class:

```
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()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();


         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()
          );


SYNCHRONIZATION SALESFORCE:
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {

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

  public void execute(QueueableContext context) {
    Http http = new Http();
  HttpRequest req = new HttpRequest();
  req.setEndpoint(WAREHOUSE_URL);
  req.setMethod('GET');
  HttpResponse res = http.send(req);

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

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

      List<Object> results = (List<Object>) JSON.deserializeUntyped(res.getBody());
```

```
        for(Object obj: results){

            Map<String, Object> mapObj = (Map<String, Object>) obj;

          Product2 product = new Product2();

            product.ExternalId = (String) mapObj.get('_id'); // _id
            product.Replacement_Part__c =(boolean) mapObj.get('replacement');
//replacement
            product.Current_Inventory__c = (Decimal) mapObj.get('quantity'); //quantity
            product.Name = (String) mapObj.get('name'); //name
            product.Maintenance_Cycle__c = (decimal) mapObj.get('maintenanceperiod');
//maintenanceperiod
            product.Lifespan_Months__c = (decimal) mapObj.get('lifespan'); //lifespan
            product.Cost__c = (decimal) mapObj.get('cost'); //cost
            product.StockKeepingUnit = (String) mapObj.get('sku'); //sku

            lstProducts.add(product);
          }

          Database.upsert (lstProducts, false);
       }

    }

}


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

```
            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}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()){
      Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                  (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                  FROM Case WHERE Id IN :validIds]);
```

```
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();


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

           newCases.add(nc);
        }
```

```
        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
          for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c item = clonedListItem.clone();
            item.Maintenance_Request__c = nc.Id;
            clonedList.add(item);
          }
        }
        insert clonedList;
      }
    }
}
```

triggger:

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


}
```

Queable job:

```
System.enqueueJob(new WarehouseCalloutService());
```

apex 4 :

WAREHOUSESYNCSCHEDULE.APX:

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

WAREHOUSECALLOUTSERVICES.APX:

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setMethod('GET');
    request.setEndpoint(WAREHOUSE_URL);
    HttpResponse response = http.send(request);
    if(response.getStatusCode() == 200) {
      List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
      system.debug('~~ '+jsonResponse);
      List<Product2> productList = new List<Product2>();
      for(Object ob : jsonResponse) {
        Map<String,Object> mapJson = (Map<String,Object>)ob;
        Product2 pr = new Product2();
        pr.Replacement_Part__c = (Boolean)mapJson.get('replacement');
        pr.Name = (String)mapJson.get('name');
        pr.Maintenance_Cycle__c = (Integer)mapJson.get('maintenanceperiod');
        pr.Lifespan_Months__c = (Integer)mapJson.get('lifespan');
        pr.Cost__c = (Decimal) mapJson.get('lifespan');
        pr.Warehouse_SKU__c = (String)mapJson.get('sku');
        pr.Current_Inventory__c = (Double) mapJson.get('quantity');
        productList.add(pr);
      }
      if(productList.size()>0)
        upsert productList;
    }
  }
  public static void execute(QueueableContext context){
    runWarehouseEquipmentSync();
  }
}
```

APEX-5 badge:

trigger:

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

test:

```
@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);
   }
}
Maintancrequest
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
```

```
            validIds.add(c.Id);
          }
        }
      }

      //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
        Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

        //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
        AggregateResult[] results = [SELECT Maintenance_Request__c,
                        MIN(Equipment__r.Maintenance_Cycle__c)cycle
                        FROM Equipment_Maintenance_Item__c
                        WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.values()){
          Case nc = new Case (
            ParentId = cc.Id,
            Status = 'New',
            Subject = 'Routine Maintenance',
            Type = 'Routine Maintenance',
            Vehicle__c = cc.Vehicle__c,
```

```
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );

            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today's
date.
            //If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            //} else {
            //    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            //}

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Apex 6 badges:

TEST:

```
@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);
  }
}
MOCK:
@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;
    }
}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');
  }

}
apex:6
test:
@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();
    }
}
mock:
@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;
    }
}
apex class:
global with sharing class WarehouseSyncSchedule implements Schedulable{
```

```
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```