Salesforce developer catalyst.

1.Apex Triggers.

A. Account Address Trigger.

```
trigger AccountAddressTrigger on Account (before insert,before
update) {

List<Account> acclist =  new List<Account>();

 for(Account a : trigger.new){

 if((a.Match_Billing_Address__c
==true)&&(account.BillingPostalCode != NULL))
a.ShippingPostalCode = a.BillingPostalCode;

 }

 }
```

B.Closed Opportunity Trigger.

```
trigger ClosedOpportunityTrigger on Opportunity(after insert,
after update) {
    List<Task> oppList = new List<Task>();


    for (Opportunity a : [SELECT Id,StageName,(SELECT
WhatId,Subject FROM Tasks) FROM Opportunity
                     WHERE Id IN :Trigger.New AND StageName LIKE
'%Closed Won%']) {
        oppList.add(new Task( WhatId=a.Id, Subject='Follow Up Test
Task'));


    }


    if (oppList.size() > 0) {
        insert oppList;
    }
}
```

2. Apex Testing.

A. Apex Class-VerifyDate.

```
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;
        }

}
```

B.Apex Class-Test Verify Date.

```
@isTest
private class TestVerifyDate {
    static testMethod void TestVerifyDate() {


        Date date1=system.today();
        Date date2=system.today().addDays(5);
        String
returnValue=String.valueOf(VerifyDate.CheckDates(date1,date2)
);




        Date date3=system.today();
        Date date4=system.today().addDays(35);
        String
returnValue2=String.valueOf(VerifyDate.CheckDates(date3,date
4));


        Date date33=system.today().addDays(35);
        Date date43=system.today();
        String
returnValue3=String.valueOf(VerifyDate.CheckDates(date33,dat
e43));
   }
}
B.@isTest
private class TestVerifyDate {
   static testMethod void TestVerifyDate() {

      Date date1=system.today();
      Date date2=system.today().addDays(5);
      String
returnValue=String.valueOf(VerifyDate.CheckDates(date1,date2)
);


      Date date3=system.today();
      Date date4=system.today().addDays(35);
      String
returnValue2=String.valueOf(VerifyDate.CheckDates(date3,date
4));

      Date date33=system.today().addDays(35);
```

```
        Date date43=system.today();

        String
returnValue3=String.valueOf(VerifyDate.CheckDates(date33,dat
e43));

}
```

B.Appex Trigger-RestrictContactByName.

```
trigger RestrictContactByName on Contact (before insert, before
update) {



          For (Contact c : Trigger.New) {

                 if(c.LastName == 'INVALIDNAME') {

                         c.AddError('The Last Name "'+c.LastName+'" is
not allowed for DML');

                 }


          }




}
```

C.Appex Class-TestRestrictContactByName.

```
@isTest
public class TestRestrictContactByName {
@isTest static void Test_insertupdateContact()
{
   Contact cnt= new Contact();
   cnt.LastName ='INVALIDNAME';
   Test.startTest();
   Database.SaveResult result =Database.insert(cnt,false);
  Test.stopTest();
   System.assert(!result.isSuccess());
   System.assert(result.getErrors().size()>0);
   System.assertEquals('The Last Name "INVALIDNAME" is not
allowed for DML',result.getErrors()[0].getMessage());


}
}
```

D.Appex Class-RandomContactFactory

```
public class RandomContactFactory {


   public static List<Contact> generateRandomContacts(Integer noOfContacts, String lastName){


       List<Contact> conList = new List<Contact>();

       for(Integer i=0; i<noOfContacts; i++){

          Contact c = new Contact(LastName=lastName, FirstName = 'Test ' + i);

          conList.add(c);


       }

       return conList;

   }

}
```

3.Asynchronous Apex

A.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;
  }
}
```

B.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);

  }


}
```

C.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_lst) {
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
            count+=1;
        }
        update l_lst_new;
    }

    global void finish (Database.BatchableContext bc) {
        system.debug('count = '+count);
    }
}
```

D. LeadProcessorTest.

```
@isTest
private class LeadProcessorTest {



    @TestSetup
    static void setup(){
        List<Lead> leads = new List<Lead>();

        for (Integer i = 0; i < 200; i++) {
            //Adding a new lead to the lead list
            leads.add(new Lead(LastName='Lead ' + i,
Company='Company Number ' + i, Status='Open - Not
Contacted'));
        }

        //Inserting the lead list
        insert leads;
    }

    static testMethod void test() {
        Test.startTest();

        LeadProcessor lp = new LeadProcessor();

        Id batchId = Database.executeBatch(lp);

        Test.stopTest();


        properly

        System.assertEquals(200, [select count() from lead where
LeadSource = 'Dreamforce']);


    }

}
```

E.AddPrimaryContact.

```
public class AddPrimaryContact implements Queueable
{
   private Contact c;
   private String state;
   public  AddPrimaryContact(Contact c, String state)
   {
      this.c = c;
      this.state = state;
   }
   public void execute(QueueableContext context)
   {
       List<Account> ListAccount = [SELECT ID, Name ,(Select
id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE
BillingState = :state LIMIT 200];
       List<Contact> lstContact = new List<Contact>();
       for (Account acc:ListAccount)
       {
             Contact cont = c.clone(false,false,false,false);
             cont.AccountId =  acc.id;
             lstContact.add( cont );
```

```
        }


        if(lstContact.size() >0 )
        {
            insert lstContact;
        }


    }


}
```

F.AddPrimaryContactTest.

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
       List<Account> Teste = new List <Account>();
       for(Integer i=0;i<50;i++)
       {
          Teste.add(new Account(BillingState = 'CA', name =
'Test'+i));
       }
       for(Integer j=0;j<50;j++)
       {
          Teste.add(new Account(BillingState = 'NY', name =
'Test'+j));
       }
       insert Teste;

       Contact co = new Contact();
       co.FirstName='demo';
```

```
        co.LastName ='demo';

        insert co;

        String state = 'CA';


         AddPrimaryContact apc = new AddPrimaryContact(co,
state);

         Test.startTest();

           System.enqueueJob(apc);

         Test.stopTest();

     }

 }
```

I.DailyLeadProcessor.

```
global class DailyLeadProcessor implements Schedulable {


   global void execute(SchedulableContext ctx) {

      List<Lead> lList = [Select Id, LeadSource from Lead where
LeadSource = null];


      if(!lList.isEmpty()) {

                  for(Lead l: lList) {

                      l.LeadSource = 'Dreamforce';

                  }

                  update lList;

              }

   }

}
```

J.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());
        }
}
```

4.Apex Integration Services.

A.AnimalLocator.

```
public class AnimalLocator {
   public static String getAnimalNameById(Integer id) {
      String animalName;
      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);
      if(response.getStatusCode()==200){
         Map<String, Object> r = (Map<String, Object>)
            JSON.deserializeUntyped(response.getBody());
         Map<String, Object> animal = (Map<String,
Object>)r.get('animal');
         animalName = string.valueOf(animal.get('name'));
      }
      return animalName;
  }
```

}

B.AnimalLocatorTest.

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());

        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

C.AnimalLocatorMock.

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);

response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chick
en food","says":"cluck cluck"}}');
        return response;
    }


}
```

D.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;
        }
     }
}
```

E.ParkLocator.

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

F.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);
   }
}
```

I.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]);
   }
}
```

J.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;
   }
}
```

I.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;
    }
}
```

5. APEX SPECIALIST SUPERBADGE.

A.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);


               }
           }
       }

       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));
             } 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> 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;
        }
    }
}
```

B.MaintainRequestHelperTrigger.

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

C.WarehouseCalloutService.

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


}
```

D.WareHouseSyncSchedule.

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

E.MaintenanceRequestHeperTest.

```
@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);

    }

}
```

F.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);



                }

            }

        }
        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;
      }
   }
}
```

G.MaintenanceRequestTrigger.

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

H.Warehouse Callout Service.

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

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

      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);
         }

      }
   }
}
```

i.WarehouseCalloutServiceTest.

```
@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]);
    }
}
```

J.WarehouseCalloutServiceMock

```
@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","replac
ement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"}]');
      response.setStatusCode(200);
      return response;
   }
}
```

K.WarehouseSyncSchedule.

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

L.WarehouseSyncScheduleTest.

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To
Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job.
CronTrigger is similar to a cron job on UNIX systems.

        // This object is available in API version 17.0 and later.

        CronTrigger a=[SELECT Id FROM CronTrigger where
NextFireTime > today];

        System.assertEquals(jobID, a.Id,'Schedule ');




    }

}
```