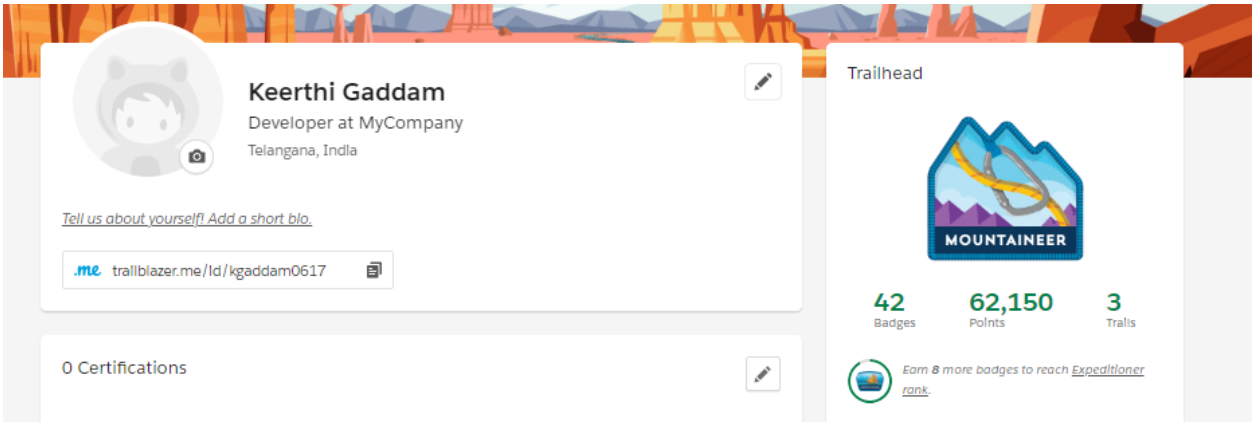


APEX TRIGGERS

1. GETSTARTEDWITHAPEXTRIGGERS:

1.AccountAddressTrigger.apxt

```
triggerAccountAddressTrigger on Account(before insert, before update) {
    for(Account a: Trigger.New){
        if(a.Match_Billing_Address__c==true&&a.BillingPostalCode!=null){
            a.ShippingPostalCode=a.BillingPostalCode;
        }
    }
}
```

2. BULKAPEXTRIGGERS:

1.ClosedOpportunityTrigger.apxt

```
triggerClosedOpportunityTriggeronOpportunity(afterinsert,afterupdate){
    List<Task>taskList=newList<Task>();
    for(Opportunityopp:[SELECTId,StageNameFROMOpportunityWHERE
```

```
StageName='ClosedWon'ANDIdIN:Trigger.New]){
        taskList.add(newTask(Subject='FollowUpTestTask',WhatId=opp.Id));
    }
    if(taskList.size()>0){
        inserttasklist;
    }
```

APEX TESTING

3. GETSTARTEDWITHAPEXUNITTEST:

1. VerifyDate.apxc

```
public class VerifyDate {
  public staticDate 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)) {
              returndate2;

        }else {

        }
        }

return SetEndOfMonthDate(date1);
  private static Boolean DateWithin30Days(Date date1, Date date2) {
        Date date30Days = date1.addDays(30); //createa date 30 days away from date1 if( date2 >
        date30Days ) { returnfalse; }
        else{ return true; }
```

```
  }
  privatestaticDateSetEndOfMonthDate(Datedate1){
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());  Date
        lastDay= Date.newInstance(date1.year(), date1.month(), totalDays); return
        lastDay;
  }
}
```

2. TestVerifyDate.apxc

```
@isTest
privateclassTestVerifyDate{
   @isTeststatic void testCheckDates() { Date
      now=Date.today();
      DatelastOfTheMonth=Date.newInstance(now.year(),now.month(),
Date.daysInMonth(now.year(),now.month()));
      Date plus60 = Date.today().addDays(60);
        Dated1 = VerifyDate.CheckDates(now, now);
      System.assertEquals(now,d1);
      Dated2=VerifyDate.CheckDates(now,plus60);
      System.assertEquals(lastOfTheMonth,d2);
   }
}
```

4. TESTAPEXTRIGGERS:

1.RestrictContactByName.apxt

```
triggerRestrictContactByName on Contact(before insert) { For
   (Contactc:Trigger.New){
        if(c.LastName == 'INVALIDNAME') {          //invalidnameisinvalid
```

```
                c.AddError('TheLastName'"+c.LastName+"'isnotallowedforDML');
        }
    }
```

5. CREATETESTDATAFORAPEXTESTS:

1.RandomContactFactory.apxc

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

ASYNCHRONOUS APEX

6. USEFUTUREMETHODS:

1. AccountProcessor.apxc

```
publicwithoutsharingclassAccountProcessor{
    //Addannotationtodeclareafuturemethod
```

```
    @future(callout=false)
    public static void countContacts(List<Id> accountIds){
    //Query all accountsin the list of Ids passed
        Map<Id,Account>accountMap=newMap<Id,Account>([SELECTId,(SELECTId
FROMContacts)FROMAccountWHEREIdIN:accountIds]);
        List<Account> listName = new List<Account>();
        //Loop through list of accounts
        for(Account a: accountMap.values()){
          //Assign field to numberof contact
           a.Number_of_Contacts__c=accountMap.get(a.Id).Contacts.size();
        }
       //UpdateAccounts
        updateaccountMap.values();

    }
}
```

2. AccountProcessorTest.apxc

```
@isTest
publicclassAccountProcessorTest{ @isTest
    publicstatic void testNoOfContacts(){
        Accounta=newAccount(); a.Name=
        'TestAccount';
        Insert a;

        Contactc=newContact();
        c.FirstName = 'Bob';
        c.LastName= 'Willie';
        c.AccountId = a.Id;

        Contactc2=newContact();
        c2.FirstName='Tom';
        c2.LastName = 'Cruise';
```

```
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);
        Test.startTest(); AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }
}
```

7. USEBATCHAPEX:

1. LeadProcessor.apxc

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    /instancemembertoretainstateacrosstransactions global
    Integer recordsProcessed = 0;

    globalDatabase.QueryLocator start(Database.BatchableContext bc) { return
        Database.getQueryLocator('SELECT Id,LeadSource FROM Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope){
        /processeachbatchofrecords
        List<Lead> leads = new List<Lead>();
        for(Lead lead : scope) {

            lead.LeadSource = 'Dreamforce';
            /incrementtheinstancemembercounter
            recordsProcessed = recordsProcessed + 1;

        }
        updateleads;
    }
```

```
    globalvoidfinish(Database.BatchableContextbc){ System.debug(recordsProcessed
        +'recordsprocessed.Shazam!');

    }
}
```

2. LeadProcessorTest.apxc

```
@isTest
publicclassLeadProcessorTest{
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
         /insert200leads
        for(Integeri=0;i<200;i++){
            leads.add(newLead(LastName='Lead'+i,
                Company='Lead', Status='Open - Not Contacted'));

        }
        insert leads;
    }

    statictestmethodvoidtest(){
        Test.startTest();
        LeadProcessorlp=newLeadProcessor(); Id
        batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

         /afterthetestingstops,assertrecordswereupdatedproperly System.assertEquals(200,
        [selectcount()fromleadwhereLeadSource=
'Dreamforce']);
    }
```

}

8. CONTROLPROCESSESWITHQUEUEABLEAPEX:

1. AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {

    private ContactcontactObj;

    private String state_code;

    publicAddPrimaryContact(Contact c, Strings) {
        this.contactObj=c;
        this.state_code=s;
    }

    publicvoidexecute(QueueableContextcontext){
        List<Account>accounts=[SELECTId
                                FROM Account
                               WHERE BillingState = :this.state_code
                               LIMIT200];

        List<Contact>contacts = new List<Contact>();
        for(Accounta:accounts){
            Contact c=this.contactObj.clone(false, false,false, false);
            c.AccountId=a.Id;
            contacts.add(c);
```

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

2. AddPrimaryContactTest.apxc

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

    INSERTlstOfAcc;
  }

  statictestmethod void testAddPrimaryContact(){ Contact
    con= new Contact(LastName = 'TestCont');
    AddPrimaryContact addPCIns= new AddPrimaryContact(CON ,'CA');

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

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

```
    }
}
```

9. SCHEDULEJOBSUSINGAPEXSCHEDULER:

1. DailyLeadProcessor.apxc

```
publicclassDailyLeadProcessorimplementsSchedulable {
    Public void execute(SchedulableContext SC){
      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200]; for(Lead
       l:LeadObj){
          l.LeadSource='Dreamforc';
          update l;
       }
    }
}
```

2. DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
  statictestMethod void testDailyLeadProcessor() {
          StringCRON_EXP='001**?';
          List<Lead> lList = new List<Lead>();
      for(Integeri=0;i<200;i++){
                  lList.add(newLead(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 SERVICES

1. APEXRESTCALLOUTS:

1. AnimalLocator.apxc

```
public class AnimalLocator {
  public static String getAnimalNameById(Integer animalId){ String
      animalName;
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+animalId);
      request.setMethod('GET');
      HttpResponse response=http.send(request);
       / If the request is successful, parse the JSON response. 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;
   }
}
```

2. AnimalLocatorMock.apxc

```
@isTest
```

```
globalclass AnimalLocatorMock implements HttpCalloutMock { global
    HTTPResponse respond(HTTPRequest request){
        HttpResponse response = newHttpResponse();
        response.setHeader('Content-Type', 'application/json');


   response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        response.setStatusCode(200); return
        response;
    }
}
```

3. AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest { @isTest
staticvoidgetAnimalNameById(){
     /Setmockcalloutclass
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
     /Thiscausesafakeresponsetobesent
     / from the class that implements HttpCalloutMock. String
    response=AnimalLocator.getAnimalNameById(1);
     / Verify thatthe response received contains fake values System.assertEquals('chicken',
    response);
}
}
```

2. APEXSOAPCALLOUTS:

1. ParkLocator.apxc

```
public classParkLocator {
    public static String[] country (Stringx) {
        String parks = x;/ {'Yellowstone','Kanha','Mount Fuji'}; ParkService.ParksImplPort
```

```
        findCountries = new ParkService.ParksImplPort (); return findCountries.byCountry
        (parks)}

}
```

2. ParkLocatorTest.apxc

```
@isTest
publicclass ParkLocatorTest { @isTest
    static void testCallout () {
         /Thiscausesafakeresponsetobegenerated
        Test.setMock (WebServiceMock.class, new ParkServiceMock ());
        Stringx='Yellowstone';
        List<String> result = ParkLocator.country(x);

        string resultstring = string.join (result,',');
        System.assertEquals('USA', resultstring);
    }
}
```

3. ParkServiceMock

```
@isTest
globalclassParkServiceMockimplementsWebServiceMock{ global
  voiddoInvoke(
      Objectstub,
      Objectrequest,
      Map<String,Object>response,
      String endpoint,
      StringsoapAction,
      StringrequestName,
      StringresponseNS,
      StringresponseName,
      StringresponseType){
```

```
        ParkService.byCountryResponse response_x =new ParkService.byCountryResponse();
}
       response_x.return_x=newList<String>{'USA'}; response.put
('response_x', response_x);
}
```

1. APEXWEBSERVICES:

1. AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
globalwithsharingclassAccountManager{
    @HttpGet
    globalstatic Account getAccount(){
        RestRequestreq=RestContext.request;
        StringaccId=req.requestURI.substringBetween('Accounts/','/contacts');
        Accountacc=[SELECTId,Name,(SELECTId,NameFROMContacts)
                    FROMAccountWHEREId=:accId];
        return acc;
    }
}
```

2. AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
    @isTeststaticvoidtestAccountManager(){ Id
        recordId=getTestAccountId();
         /Setupatestrequest
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
```

```
        /Callthemethodtotest
        Account acc=AccountManager.getAccount();
        / Verify results
        System.assert(acc!=null);
    }

    private static Id getTestAccountId(){
        Accountacc=newAccount(Name='TestAcc2'); Insertacc;
        Contactcon=newContact(LastName='TestCont2',AccountId=acc.Id); Insertcon;
        return acc.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE

2. AUTOMATE RECORDCREATION:

1. MaintenanceRequest.apxt

```
triggerMaintenanceRequestonCase(beforeupdate,afterupdate){
    / ToDo: Call MaintenanceRequestHelper.updateWorkOrders if(Trigger.isUpdate
    &&Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }
}
```

2. MaintenanceRequestHelper.apxc

```
publicwithsharingclassMaintenanceRequestHelper{
```

```
publicstatic 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=newMap<Id,Case>([SELECTId,Vehicle__c, Equipment_
c,Equipment_r.Maintenance_Cycle_c,(SELECT
Id,Equipment__c,Quantity__cFROMEquipment_Maintenance_Items__r)
                                        FROMCase 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(Casecc:closedCasesM.values()){ Case nc
```

```
= new Case (
    ParentId = cc.Id,
Status=
'New',Subject='Routine
Maintenance', Type
='RoutineMaintenance',
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__cwp:
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items___r){
        Equipment_Maintenance_Item_cwpClone=wp.clone();
        wpClone.Maintenance_Request_c=nc.Id; ClonedWPs.add(wpClone);

    }
}
insert ClonedWPs;
}
}
```

}

3. SYNCHRONIZATIONSALESFORCEDATAWITHAN EXTERNAL SYSTEM:

1.WarehouseCalloutService.apxc

```
publicwithsharingclassWarehouseCalloutServiceimplementsQueueable{ private
   staticfinalStringWAREHOUSE_URL='https://th-superbadge-
apex.herokuapp.com/equipment';


   //classthatmakesaRESTcallouttoanexternalwarehousesystemtogetalistof equipmentthat
needstobeupdated.
   //Thecallout'sJSONresponsereturnstheequipmentrecordsthatyouupsertin Salesforce.

   @future(callout=true)
   publicstaticvoidrunWarehouseEquipmentSync(){ Http
      http=newHttp();
      HttpRequestrequest=newHttpRequest();


      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());

        //classmapsthefollowingfields:replacementpart(alwaystrue),cost,current inventory,
lifespan,maintenance cycle,andwarehouse SKU
        //warehouseSKUwillbeexternalIDforidentifyingwhichequipmentrecordsto update
withinSalesforce
        for(Objecteq:jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)eq;
            Product2myEq=newProduct2();
            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('Yourequipmentwassyncedwiththewarehouseone');
        }
    }
  }

  publicstatic void execute(QueueableContext context){
    runWarehouseEquipmentSync();
  }

}
```

4. SCHEDULESYNCHRONIZATIONUSINGAPEXCODE:

1.WarehouseSyncSchedule.apxc

```
globalclass WarehouseSyncSchedule implements Schedulable { global
   voidexecute(SchedulableContextctx){

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

5. TESTAUTOMATIONLOGIC:

1. MaintenanceRequestHelperTest.apxc

```
publicwithsharingclassMaintenanceRequestHelper{
   publicstatic voidupdateworkOrders(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=newMap<Id,Case>([SELECTId,Vehicle__c, Equipment_
c,Equipment_r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__cFROMEquipment_Maintenance_Items__r)
                                    FROMCaseWHERE IdIN: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__cWHEREMaintenance_Request__cIN:ValidIdsGROUP BY
Maintenance_Request_c];

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

        for(Casecc:closedCasesM.values()){ Case nc
            = new Case (
                ParentId = cc.Id,
            Status='New',
                Subject='RoutineMaintenance', Type
                ='RoutineMaintenance', 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;
      }
   }
}
```

2. MaintenanceRequestHelper.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

   private static final string STATUS_NEW = 'New';


   private static final string WORKING = 'Working';
   private static final string CLOSED = 'Closed'; private
   static final string REPAIR = 'Repair';
   private static final string REQUEST_ORIGIN = 'Web';
   private static final string REQUEST_TYPE = 'Routine Maintenance';
```

```
privatestaticfinalstringREQUEST_SUBJECT='Testingsubject';

PRIVATESTATICVehicle__ccreateVehicle(){
    Vehicle__cVehicle=newVehicle__C(name='SuperTruck'); return
    Vehicle;
}

PRIVATESTATICProduct2createEq(){
    product2 equipment = new product2(name = 'SuperEquipment', lifespan_months_C=
                        10,
                        maintenance_cycle_C=10,
                        replacement_part__c=true);
    return equipment;
}

PRIVATESTATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){ casecs=
    newcase(Type=REPAIR,
                Status=STATUS_NEW,
                Origin=REQUEST_ORIGIN,
                Subject=REQUEST_SUBJECT,
                Equipment_c=equipmentId,
                Vehicle_c=vehicleId);
    return cs;
}

PRIVATESTATICEquipment_Maintenance_Item__ccreateWorkPart(idequipmentId,id
requestId){
    Equipment_Maintenance_Item_cwp=new Equipment_Maintenance_Item_
c(Equipment_c=equipmentId,
                                        Maintenance_Request__c=requestId);
    return wp;
}
 @istest
privatestatic void testMaintenanceRequestPositive(){ Vehicle_
```

```
        cvehicle=createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
Product2equipment=createEq();
        insertequipment;
        idequipmentId=equipment.Id;

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

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

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

        CasenewReq=[Selectid,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
                                fromEquipment_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
```

```
    privatestatic void testMaintenanceRequestNegative(){ Vehicle_Cvehicle=createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
product2equipment=createEq(); insertequipment;
        idequipmentId=equipment.Id;


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

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


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

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


        Equipment_Maintenance_Item__c workPart=[select id
                                                fromEquipment_Maintenance_Item__c
                                                whereMaintenance_Request__c=:emptyReq.Id];


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


    @istest
    privatestaticvoidtestMaintenanceRequestBulk(){ 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=newlist<case>();
list<id> oldRequestIds = new list<id>();

for(integeri=0;i<300;i++){
  vehicleList.add(createVehicle());
  equipmentList.add(createEq());
}
insertvehicleList;
insertequipmentList;

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

for(integeri=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);
}
updaterequestList;
test.stopTest();
list<case> allRequests= [selectid
                fromcase
                wherestatus =: STATUS_NEW];
list<Equipment_Maintenance_Item__c>workParts=[selectid
                        fromEquipment_Maintenance_Item__c
                        whereMaintenance_Request_cin:oldRequestIds];
```

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

3. MaintenanceRequest.apxt

```
triggerMaintenanceRequestonCase(beforeupdate,afterupdate){
    / ToDo: Call MaintenanceRequestHelper.updateWorkOrders if(Trigger.isUpdate
    &&Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }
}
```

6. TESTCALLOUTLOGIC:

1. WarehouseCalloutService.apxc

```
publicwithsharingclassWarehouseCalloutServiceimplementsQueueable{ private
    staticfinalStringWAREHOUSE_URL='https://th-superbadge-
apex.herokuapp.com/equipment';

    //classthatmakesaRESTcallouttoanexternalwarehousesystemtogetalistof equipmentthat
needstobeupdated.
    //Thecallout'sJSONresponsereturnstheequipmentrecordsthatyouupsertin Salesforce.

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

        Httphttp=newHttp();
```

```
        HttpRequestrequest=newHttpRequest();


        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());
            //classmapsthefollowingfields:replacementpart(alwaystrue),cost,current inventory,
lifespan, maintenance cycle, andwarehouse SKU
            //warehouseSKUwillbeexternalIDforidentifyingwhichequipmentrecordsto update
withinSalesforce
            for(Objecteq:jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2myEq=newProduct2();
                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('Yourequipmentwassyncedwiththewarehouseone');
            }
        }
    }
    publicstatic void execute(QueueableContext context){
```

```
        runWarehouseEquipmentSync();
    }
}
```

2. WarehouseCalloutServiceTest.apxc

```
@isTest
privateclass WarehouseCalloutServiceTest {
    @isTest
    staticvoid testWareHouseCallout(){
        Test.startTest();
         /implementmockcallouttesthere
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count()FROM Product2]);
    }
}
```

3. WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
     /implementhttpmockcallout
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
         /Createafakeresponse
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
,"name":"Generator1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
```

```
        response.setStatusCode(200); return
        response;
    }
    }
```

7. TEST SCHEDULING LOGIC:

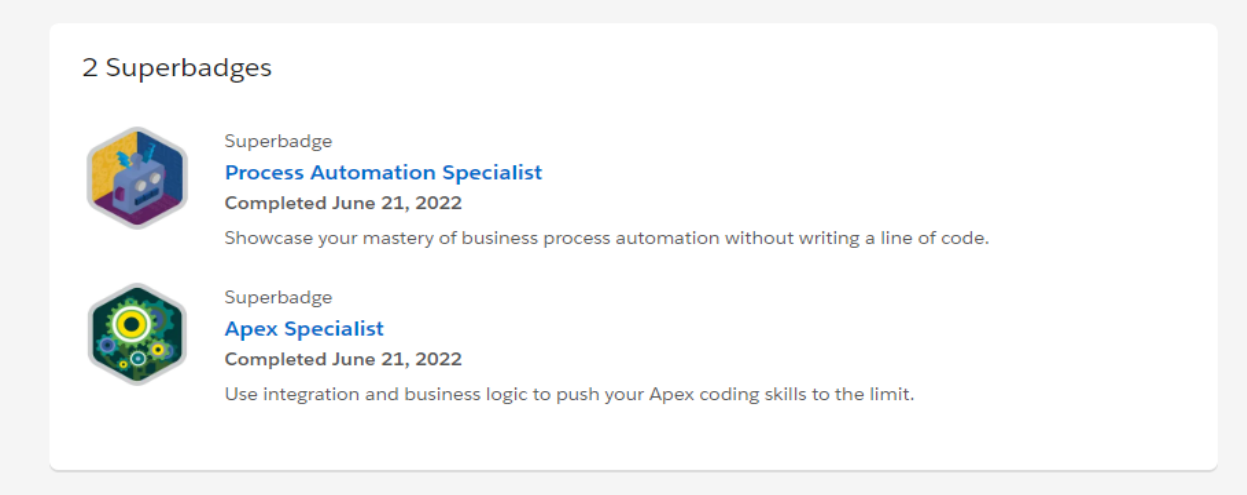
1. WarehouseSyncSchedule.apxc

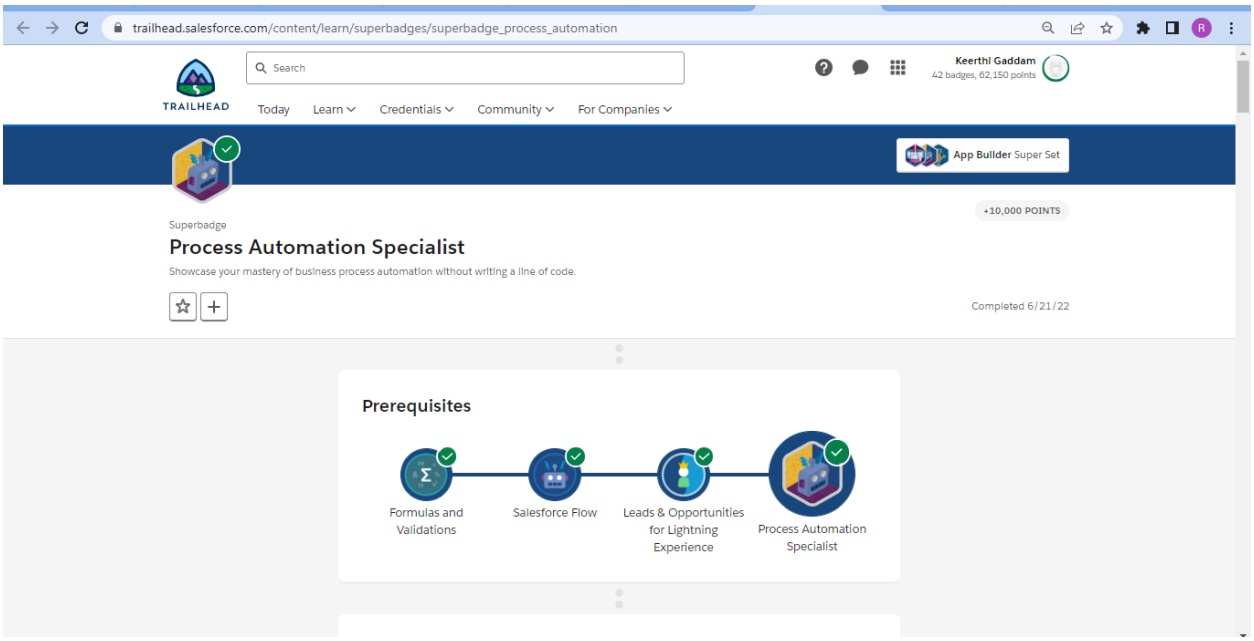
```
globalclass WarehouseSyncSchedule implements Schedulable { global
    voidexecute(SchedulableContextctx){

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

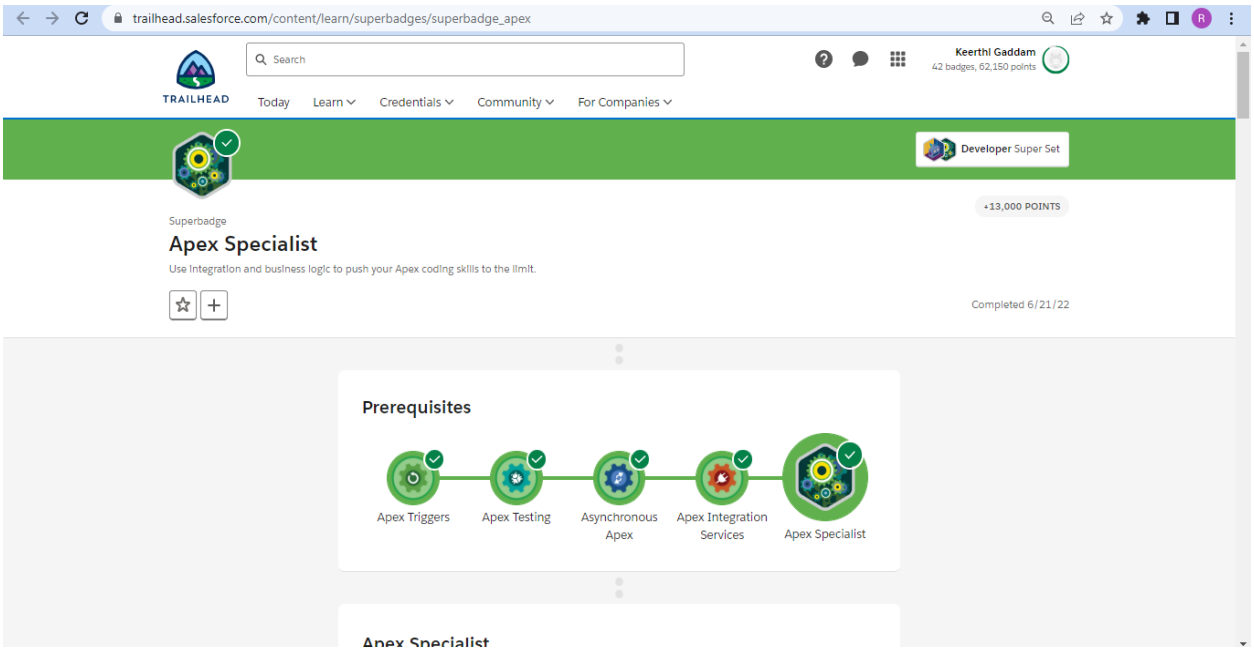
2. WarehouseSyncScheduleTest.apxc

```
@isTest
public class WarehouseSyncScheduleTest {
    @isTeststatic voidWarehousescheduleTest(){ String
        scheduleTime='000001**?'; Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock()); String
        jobID=System.schedule('Warehouse Time To Scheduleto Test',
scheduleTime, new WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cronjobon
UNIXsystems.
         /Thisobjectisavailable inAPIversion17.0andlater.

        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');

    }
}
```

Superbadges Completion:






trailhead.salesforce.com/content/learn/superbadges/superbadge_apex
TRAILHEAD
Search
Keerthi Gaddam
42 badges, 62,150 points
Today
Learn
Credentials
Community
For Companies
Developer Super Set
+13,000 POINTS
Superbadge
Apex Specialist
Use integration and business logic to push your Apex coding skills to the limit.
Completed 6/21/22
Prerequisites
Apex Triggers
Apex Testing
Asynchronous Apex
Apex Integration Services
Apex Specialist
Apex Specialist