APEX SPECIALIST SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.axpt:

```
trigger AccountAddressTriggeron Account (before insert,before
update) { for(Account account:Trigger.New){
   if(account.Match_Billing_Address c == True){
       account.ShippingPostalCode = account.BillingPostalCode;
     }

  }

}
```

ClosedOpportunityTrigger.axpt:

```
  trigger ClosedOpportunityTrigger on Opportunity (afterinsert,after
update) { List<Task> tasklist= new List<Task>();
for(Opportunity opp: Trigger.New){
    if(opp.StageName == 'ClosedWon'){
        tasklist.add(newTask(Subject = 'Follow Up Test Task',WhatId =opp.Id));
    }
}
if(tasklist.
    size() >
    0){
    insert
    taskli
    st;

}
```

```
}



public class VerifyDate {
```

APEX TESTING

VerifyData.apxc:

```
        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;

                } else {

                }
                                                        }

return SetEndOfMonthDate(date1);

        @TestVisible privatestatic Boolean DateWithin30Days(Datedate1, Date date2){
                /check for date2 being
        inthe past if( date2< date1) {
        returnfalse; }
```

```
 /check that date2 is within (>=)30 days of date1

Date date30Days = date1.addDays(30);  /create a date 30 days away from
          date1 if( date2 >= date30Days ) { return false; }
          else { return true; }
}


 /method to returnthe end of the monthof a given date






@TestVisible private staticDate
SetEndOfMonthDate(Datedate1){
          IntegertotalDays =Date.daysInMonth(date1.year(), date1.month());

          Date lastDay = Date.newInstance(date1.year(),
          date1.month(),totalDays); return lastDay;
}


}
```

TestVerifyData.apxc:

```
@isTest

private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){

      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('01/05/2022'));
      System.assertEquals(date.parse('01/05/2022'), D);
}

   @isTest static void Test_CheckDates_case2(){

      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('05/05/2022'));
      System.assertEquals(date.parse('01/31/2022'), D);
   }
   @isTest static void Test_Within30Days_case1(){

      Boolean flag =

VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('12/30/2021'));
      System.assertEquals(false, flag);

   }
@isTest static void Test_Within30Days_case2(){

      Boolean flag =

VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('02/02/2021'));
      System.assertEquals(false, flag);

   }

@isTest static void Test_Within30Days_case3(){
        Boolean flag =
```

APEXSPECIALIST SUPER BADGE CODES

```
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('01/15/2022'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Datereturndate =VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
}
}
```

RestrictContactByName.apxt:

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


                    /check contacts prior to insertor update for
                   invalid data For (Contactc : Trigger.New) {
                            if(c.LastName == 'INVALIDNAME') {    /invalidname is invalid
                                        c.AddError('The Last Name "'+c.LastName+'" is not allowedfor
                                        DML');
                            }
                    }
}
```

TestRestrictContactByName.apxc:

```
@isTest
private class TestRestrictContactByName
    { @isTeststatic void
    Test_insertupdateContact(){
        Contact cnt = new
        Contact();cnt.LastName =
        'INVALIDNAME';






        Test.startTest();
        Database.SaveResult result=
        Database.insert(cnt,false);Test.stopTes
        t();System.assert(!result.isSuccess());
        System.assert(result.getErrors().size()
        > 0);
        System.assertEquals('The Last Name"INVALIDNAME" is notallowed for  DML',
result.getErrors()[0].getMessage());
    }
}
```

APEX SPECIALIST SUPER BADGE CODES

RandomContactFactory.apxc:

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num_cnts, string lastname) {
      List<Contact> contacts= new List<Contact>();
        for(Integer i = 0; i < num_cnts; i++) {
            Contact cnt = new Contact(FirstName = 'Test' +i,LastName =
            lastname); contacts.add(cnt);

        }
        return contacts;
                    }
}
```

ASYNCHRONOUS APEX

AccountProcessor.apxc:

```
public class AccountProcessor
                    {@future
```

```
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new
        List<Account>();


        List<Account> accounts = [Select Id, Name, (Select Id from Contacts)from Account Where Id
        in

:accountIds];

        For(Account acc: accounts) {

                                        List<Contact> contactList = acc.contacts;

            acc.Number_Of_Contacts c = contactList.size();

            accountsToUpdate.add(acc);

        }

        update accountsToUpdate;

    }

}
```

AccountProcessorTest.apxc:

```
@isTest

public class AccountProcessorTest {

                    @isTest

    private static void testCountContacts() {

        Account newAccount = new Account(Name =
        'TestAccount'); insert newAccount;
        ContactnewContact1 = new Contact(FirstName ='John',LastName = 'Doe',AccountId =









newAccount.Id);
```

```
        insert newContact1;


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

LeadProcessor.apxc:

```
global class LeadProcessor implements
                    Database.Batchable<sObject>{ globalInteger count =
                    0;


    global Database.QueryLocator start(Database.BatchableContext
    bc) { return Database.getQueryLocator('SELECT ID,LeadSource
    FROM Lead');
    }
    global void execute(Database.BatchableContext bc, List<Lead>
        L_list){ List<lead> L_list_new = new List<lead>();
        for(lead L: L_list){
            L.leadSource =
            'Dreamforce';
            L_list_new.add(
            L);count += 1;
        }
        update L_list_new;
    }
    global void
       finish(Database.BatchableContext bc){
```

```
       system.debug('count= ' + count);
    }
}
```

LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {
          @isTest
   publicstatic void
          testit(){
```

APEX SPECIALIST SUPER BADGE CODES

```
      List<lead> L_list = new
      List<lead>();for(Integer i = 0; i <
      200; i++) {
         Lead L = new
         Lead();L.LastName
         = 'name'
         + i; L.Company =
         'Company';
```

```
            L.Status
            = 'Random
            Status';
            L_list.add(L);
        }
        insert
        L_list;
        Test.start
        Test();
        LeadProcessor lp = new
        LeadProcessor(); Id batchId =
        Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

AddPrimaryContact.apxc:

```
public class AddPrimaryContact
                implementsQueueable{ private
                Contact con;
    private String state;
    public AddPrimaryContact(Contact con, String
        state) { this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id,Name,(Select FirstName,LastName, Id from
                          contacts) from Accountwhere BillingState = :state Limit 200];
        List<Contact> primaryContacts = new
        List<Contact>();for(Account acc : accounts) {
            Contact c =
            con.clone();
            c.AccountId =
            acc.Id;
            primaryContacts.a
            dd(c);
        }
        if(primaryContacts.si
            ze() > 0) {
            insertprimaryCont
            acts;
        }
    }
}
```

```
@isTest public class
```

APEX SPECIALIST SUPER BADGE CODES

AddPrimaryContactTest.apxc:

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

DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor
    implementsSchedulable{ global void
    execute(SchedulableContext ctx) {
        List<Lead> leadstoupdate = new List<Lead>();
        List<Lead> leads = [Select id From LeadWhere LeadSource = NULL
        Limit200]; for(Lead l: leads) {
            l.LeadSource = 'Dreamforce';
            leadstoupdate.add(l);
        }
        update leadstoupdate;
    }
}
```

APEX SPECIALIST SUPER BADGE CODES

DailyLeadProcessorTest.apxc:

```
@
i

s

T

e s
t

private class DailyLeadProcessorTest {
                    public static String CRON_EXP= '0 0 0 15 3 ?
    2024'; static testmethod void testScheduledJob() {
        List<Lead> leads= new
        List<Lead>(); for(Integer i
        =0; i < 200; i++) {
            Lead l = new
                Lead(
                FirstName =
            'First'
            + i, LastName
            ='LastName',
            Company =
            'TheInc'
          );
          leads.add(l);
        }
        insert
        leads;
        Test.start
        Test();
        String jobId =System.schedule('ScheduledApexTest',CRON_EXP,new
              DailyLeadProcessor()); Test.stopTest();
        List<Lead> checkleads = new List<Lead>();
        checkleads = [SelectIdFrom Lead Where LeadSource = 'Dreamforce' and Company = 'The
        Inc']; System.assertEquals(200,checkleads.size(),'Leads were not created');

      }
    }

    public class AnimalLocator{
```

APEX INTEGRATION SERVICES

AnimalLocator.apxc:

```
public static String
  getAnimalNameById(Integer x){
  Httphttp = new Http();
  HttpRequest req =new HttpRequest();
  req.setEndpoint('https: /th-apex-http-callout.herokuapp.com/animals/'
  +x); req.setMethod('GET');
  Map<String, Object> animal= new Map<String,
  Object>(); HttpResponse res = http.send(req);
    if (res.getStatusCode() == 200) {
```

APEX SPECIALIST SUPER BADGE CODES

```
    Map<String, Object> results = (Map<String,
  Object>)JSON.deserializeUntyped(res.getBody()); animal= (Map<String, Object>)
  results.get('animal');
    }
return (String)animal.get('name');

  }
```

```
}


@isTest
private class AnimalLocatorTest{
```

AnimalLocatorTest.apxc:

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

AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
      / Implementthis interface method
    global HTTPResponse respond(HTTPRequest request) {
         / Create a fake response
        HttpResponse response = new
        HttpResponse();
        response.setHeader('Content-
        Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
        response.setStatusC
        ode(200); return
        response;
    }
}
```

ParkLocator.apxc:

```
        public class
       ParkLocator {
    public staticstring[] country(string theCountry) {
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort();/
        removespace return parkSvc.byCountry(theCountry);
    }
}
```

```
@isTest private class
```

ParkLocatorTest.apxc:

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

ParkServiceMock.apxc:

```
@isTest
global class ParkServiceMock implements
  WebServiceMock { global void doInvoke(
        Obje
        ct
        stub,
        Obje
        ct
        reque
        st,
        Map<String, Object>




        response, String endpoint,
        String
        soapAction,
        String
        requestNam
        e, String
        responseNS,
        String
        responseNam
        e,
        Stringrespons
        eType){
       /start -specifythe response you want to send

      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac NationalPark', 'Yosemite'};
       / end

      response.put('response_x',response_x);
  }

}
```

AccountManager.apxc:

```
@RestResource(urlMapping='/Accounts/*/
contacts') global classAccountManager {
    @HttpGet

    global static Account getAccount() {
        RestRequest req =
        RestContext.request;
        String accId =req.requestURI.substringBetween('Accounts/', '/contacts');
```

APEX SPECIALIST SUPER BADGE CODES

```
        Account acc = [SELECTId, Name, (SELECTId, Name FROM
                    Contacts) FROM AccountWHERE Id = :accId];
        return acc;
    }
}
```

AccountManagerTest.apxc:

```
@isTest
private class AccountManagerTest {


    private static testMethod
        voidgetAccountTest1() { Id recordId =
        createTestRecord();
         / Set up a test request
        RestRequest request= new RestRequest();
        request.requestUri= 'https: /na1.salesforce.com/services/apexrest/Accounts/'+
        recordId
+'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request= request;
         / Call the method to test
        Account thisAccount = AccountManager.getAccount();
         / Verify results
        System.assert(thisAccount
        !=null);
        System.assertEquals('Test record',thisAccount.Name);


    }
    / Helper method
     static Id createTestRecord() {
       / Create test record
     Account TestAcc = new Account(
       Name='Test record');
     insert TestAcc;
     Contact TestCon= new Contact(
     LastName='Test',



     AccountId =
     TestAcc.id);
     return
     TestAcc.Id;
  }
}
```

APEX SPECIALIST SUPER BADGE CODES

APEX SPECIALIST SUPER BADGE

Challeng e-1

MaintenanceRequestHelper.apxc:

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



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




        }
      }
    }


    if (!validIds.isEmpty()){








      List<Case> newCases = new List<Case>();
      Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle c, Equipment c,
Equipment r.Maintenance_Cycle c,(SELECT Id,Equipment c,Quantity c FROM
Equipment_Maintenance_Items r)
                              FROM Case WHERE Id IN :validIds]);
      Map<Id,Decimal> maintenanceCycles = new
      Map<ID,Decimal>();AggregateResult[] results = [SELECT
```

```
          Maintenance_Request  c,
MIN(Equipmentr.Maintenance_Cyclec)cycle FROM Equipment_Maintenance_Item c
WHEREMaintenance_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
             (
                Parent
             Id =cc.Id,
             Status
             ='New',
```

APEX SPECIALIST SUPER BADGE CODES

```
                Subject =
                'RoutineMaintenance',
                Type = 'Routine
                Maintenance', Vehicle c =
                cc.Vehicle  c, Equipment
                c
                =cc.Equipment  c, Origin
                ='Web',
                Date_Reportedc = 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 (Casenc : 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;
        }
    }
}
```

APEX SPECIALIST SUPER BADGE CODES

MaintenanceRequest.apxt:

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

MaintenanceRequestHelperTest.apxc:

```
@
i

s

t

e

s

t

public with sharing class MaintenanceRequestHelperTest {


    private static final string STATUS_NEW
    ='New'; private staticfinal string
    WORKING= 'Working'; private static final
```

```
string CLOSED = 'Closed'; private static
final string REPAIR = 'Repair';
private staticfinal string REQUEST_ORIGIN = 'Web';

private static final string REQUEST_TYPE =
'RoutineMaintenance'; private static final string
REQUEST_SUBJECT = 'Testing subject';


PRIVATE STATICVehicle_c createVehicle(){

   Vehicle  c Vehicle= new VehicleC(name
   ='SuperTruck'); return Vehicle;
}


PRIVATE STATIC Product2 createEq(){

   product2equipment = 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(){ Vehiclec
        vehicle= createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
```

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


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


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


test.startTest();
somethingToUpdate.stat
us =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];
```

APEX SPECIALIST SUPER BADGE CODES

```
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);
    insertemptyReq;
    Equipment_Maintenance_Item c workP
    =createWorkPart(equipmentId,emptyReq.Id);  insertworkP;


    test.startTest();
    emptyReq.Stat
    us =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 =
        newlist<case>(); list<id>
        oldRequestIds =new list<id>();


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

        insert








        vehicleList;
        insert
        equipmentLi
        st;

        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.ad
    d(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);
   }
}
```

Challenge-2

WarehouseCalloutService.apxc:

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


    /class that makesaREST callout to an externalwarehouse system to get a list of equipment
that needs to be updated.
    /The callout's JSON response returns the equipmentrecords that you upsert in
    Salesforce.


   @future(callout=tru
   e)public staticvoid
     runWarehouseEquipmentSync(){ Http
     http= new Http();
     HttpRequest request=new HttpRequest();


     request.setEndpoint(WAREHOUSE_UR
     L);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 (alwaystrue), cost,
    currentinventory, lifespan, maintenance cycle, and warehouse SKU
           /warehouse SKU will be external ID for identifying which equipment records
    toupdate withinSalesforce
          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.si
            ze()> 0){
            upsertwarehous
            eEq;
            System.debug('Your equipmentwas synced with the warehouse one');
          }
        }
      }


      public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
      }



  }
```

@isTest

WarehouseCalloutServiceMock.apxc:

```
global classWarehouseCalloutServiceMock implements HttpCalloutMock {
     / implement http mock callout
   global staticHttpResponse respond(HttpRequest request){
```

```
        HttpResponse response = new
        HttpResponse();
        response.setHeader('Content-
        Type', 'application/json');
```

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Gene rator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611100a af742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf743 ","replacement":true,"quantity":143,"name":"Fuse

```
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
    response.setStatusCode(200);


    return response;
  }
}
```

WarehouseCalloutServiceTest.apxc:

```
@IsTest
private class WarehouseCalloutServiceTest {
   / implement your mock callout
       testhere @isTest
  static void testWarehouseCallout() {
    test.startTest();
    test.setMock(HttpCalloutMock.class,new WarehouseCalloutServiceMock());
    WarehouseCalloutService.execute(null);
    test.stopTest();


    List<Product2> product2List = new
    List<Product2>();product2List = [SELECTProductCode
    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);
  }
}
```

Challenge-3

WarehouseSyncSchedule.apxc:

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
```

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

}
```

WarehouseSyncScheduuleTest.apxc:

```
@isTest

public class WarehouseSyncScheduleTest {


    @isTest static void
        WarehousescheduleTest(){
        StringscheduleTime = '00 00 01
        * * ?'; 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 scheduledjob. CronTrigger is similarto 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 ');



    }
}
```

Challenge-4

MaintenanceRequestHelperTest.apxc:

```
@istest
public with sharing class MaintenanceRequestHelperTest {


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


    PRIVATE STATICVehicle__c createVehicle(){
```

```
    Vehicle  c Vehicle= new VehicleC(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(){ Vehiclec
      vehicle= createVehicle();
      insert vehicle;

      id vehicleId = vehicle.Id;


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

APEX SPECIALIST SUPER BADGE CODES

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


      Equipment_Maintenance_Item c workP
```

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


        test.startTest();
        somethingToUpdate.stat
        us =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;
```

APEXSPECIALIST SUPER BADGE CODES

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

Equipment_Maintenance_Item c workP
=createWorkPart(equipmentId,emptyReq.Id);  insertworkP;

test.startTest();
emptyReq.Stat
us =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 =
        newlist<case>(); list<id>
        oldRequestIds =new list<id>();


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

```
insert
equipmentLi
st;
```

APEX SPECIALIST SUPER BADGE CODES

```
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 :
    requestLis
```

```
          t){
          req.Status
          =CLOSED;
          oldRequestIds.ad
          d(req.Id);
        }

        updatere
        questList;
        test.stopT
        est();


        list<case> allRequests = [select id

                              from case

                              where status=:STATUS_NEW];


        list<Equipment_Maintenance_Item_c> workParts = [select id

                                              fromEquipment_Maintenance_Item c

                                              where Maintenance_Request  c in: oldRequestIds];


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

      }

    }
```

MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {








    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
        nonUpdCaseMap) { Set<Id> validIds= new Set<Id>();
        For (Case c : updWorkOrders){
           if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
```

```
              if (c.Type == 'Repair'||c.Type== 'Routine Maintenance'){
                 validIds.add(c.Id);


              }
           }
        }


        if (!validIds.isEmpty()){
           List<Case> newCases = new List<Case>();
           Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle c, Equipment c,
  Equipment r.Maintenance_Cycle c,(SELECT Id,Equipment c,Quantity c FROM
  Equipment_Maintenance_Items r)
                                                    FROM Case WHERE Id IN :validIds]);
           Map<Id,Decimal> maintenanceCycles = new
           Map<ID,Decimal>();AggregateResult[] results= [SELECT
           Maintenance_Request  c,
  MIN(Equipmentr.Maintenance_Cyclec)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
        (
            Parent
        Id =cc.Id,
        Status
        ='New',
            Subject =
            'RoutineMaintenance',
            Type = 'Routine
            Maintenance', Vehicle c =
            cc.Vehicle  c, Equipment
            c
            =cc.Equipment  c, Origin
            ='Web',
            Date_Reportedc = 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 (Casenc : 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;
        }
      }
```

}

Challenge-5

WarehouseCalloutService.apxc:

public with sharing classWarehouseCalloutService implements
  Queueable { private static final String WAREHOUSE_URL = 'https:
  /th-superbadge-
apex.herokuapp.com/equipment';

   /class that makesaREST callout to an externalwarehouse system to get a list of equipment that needs to be updated.
   /The callout’s JSON response returns the equipmentrecords that you upsert in
   Salesforce.

  @future(callout=tru
  e)public staticvoid

    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 (alwaystrue), cost,
currentinventory, lifespan, maintenance cycle, and warehouse SKU
             /warehouse SKU will be external ID for identifying which equipment records
toupdate withinSalesforce
            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.si
                ze()> 0){
                upsertwarehous
                eEq;
                System.debug('Your equipmentwas synced with the warehouse one');
            }
}
}


public static void execute (QueueableContext context){
runWarehouseEquipmentSync();
}
```

```
}

@isTest
```

APEXSPECIALIST SUPER BADGE CODES

WarehouseCalloutServiceMock.apxc:

```
global classWarehouseCalloutServiceMock implements HttpCalloutMock {
     / implement http mock callout
   global staticHttpResponse respond(HttpRequest request){









        HttpResponse response = new
        HttpResponse();
        response.setHeader('Content-
        Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"
na me":"Gene                 rator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d6622672
6b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611
100aaf743 ","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);


        return response;
    }
}
```

WarehouseCalloutServiceTest.apxc:

```
@isTest
global classWarehouseCalloutServiceMock implements HttpCalloutMock {
      / implement http mock callout
    global staticHttpResponse respond(HttpRequest request){


        HttpResponse response = new
        HttpResponse();
        response.setHeader('Content-
        Type', 'application/json');


response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"
na me":"Gene                 rator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d6622672
6b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611
100aaf743  ","replacement":true,"quantity":143,"name":"Fuse






20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
```

```
        response.setStatusCode(200);



        return response;
    }
}
```

Challenge-6

WarehouseSyncSchedule.apxc:

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

WarehouseSyncScheduleTest.apxc:

```
@isTest
public class WarehouseSyncScheduleTest {
    @isTest static void
        WarehousescheduleTest(){
        StringscheduleTime = '00 00 01
        * * ?';








        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 scheduledjob. CronTrigger is similarto 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 ');



    }
}
```