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 (after insert,after update) {
List<Task> tasklist = new List<Task>();
for(Opportunity opp: Trigger.New){
  if(opp.StageName == 'Closed Won'){
    tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId = opp.Id));
  }
}
if(tasklist.size() > 0){
  insert tasklist;

}
}
```

```
 public class VerifyDate {
```

APEX SPECIALIST SUPER BADGE CODES

APEX TESTING

VerifyData.apxc:

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

        } else {


        }
                                                        }
return SetEndOfMonthDate(date1);

        @TestVisible private static Boolean DateWithin30Days(Datedate1, Date date2) {
                /check for date2 being in the past if(
        date2 < date1) { return false; }
```

APEX SPECIALIST SUPER BADGE CODES

```
    /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(Date date1){
        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_ca
```

APEX SPECIALIST SUPER BADGE CODES

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

RestrictContactByName.apxt:

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

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

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;
  }
}
```

APEX SPECIALIST SUPER BADGE CODES

AccountProcessorTest.apxc:

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

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

LeadProcessor.apxc:

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

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

```
}
```

APEX SPECIALIST SUPER BADGE CODES

LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {
                @isTest
      publicstatic void testit(){
      List<lead> L_list = new
      List<lead>();for(Integer i = 0; i < 200; i++) {
         Lead L = new Lead();
         L.LastName = 'name' + i;
         L.Company = 'Company';
         L.Status = 'Random Status';
         L_list.add(L);
      }
      insert L_list;
      Test.startTest();
      LeadProcessor lp = new LeadProcessor(); Id
      batchId = Database.executeBatch(lp);
      Test.stopTest();
   }
}
```

AddPrimaryContact.apxc:

```
public class AddPrimaryContact implements Queueable{
            private Contact con;
   private String state;
   public AddPrimaryContact(Contact con, String state) {
      this.con = con;
      this.state = state;
   }
   public void execute(QueueableContext context) {
      List<Account> accounts = [Select Id,Name,(Select FirstName,LastName, Id from contacts) from
                  Account where BillingState = :state Limit 200];
      List<Contact> primaryContacts = new List<Contact>(); for(Account
      acc : accounts) {
         Contact c = con.clone();
         c.AccountId = acc.Id;
         primaryContacts.add(c);
      }
```

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

APEX SPECIALIST SUPER BADGE CODES

AddPrimaryContactTest.apxc:

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

DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx) {
        List<Lead> leadstoupdate = new List<Lead>();
        List<Lead> leads = [Select id From Lead Where LeadSource = NULL Limit 200];
        for(Lead l: leads) {
            l.LeadSource = 'Dreamforce'; leadstoupdate.add(l);
        }
        update leadstoupdate;
```

```
    }
}
```

APEX SPECIALIST SUPER BADGE CODES

DailyLeadProcessorTest.apxc:

```
  @isTest
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 = 'The
            Inc'
         );
         leads.add(l);
      }
      insert leads;
      Test.startTest();
      String jobId = System.schedule('ScheduledApexTest',CRON_EXP,new DailyLeadProcessor());
            Test.stopTest();
      List<Lead> checkleads = new List<Lead>();
      checkleads = [SelectId From 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:

APEX SPECIALIST SUPER BADGE CODES

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

}
```

```
@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 {
    / Implement this interface method
   global HTTPResponse respond(HTTPRequest request) {
```

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

APEX SPECIALIST SUPER BADGE CODES

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

ParkLocator.apxc:

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

ParkLocatorTest.apxc:

```
@isTest
private class 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', 'MackinacNational Park', 'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

APEX SPECIALIST SUPER BADGE CODES

ParkServiceMock.apxc:

```
@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) {
       /start -specify the 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 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;
   }
}
```

AccountManagerTest.apxc:

```
@isTest
```

```
private class AccountManagerTest {

    private static testMethod void getAccountTest1() { 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

Challenge-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(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 = 'RoutineMaintenance', 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 (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:

```
  @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 static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATICVehicle__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;
              }
```

APEX SPECIALIST SUPER BADGE CODES

```
  @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];
```

APEX SPECIALIST SUPER BADGE CODES

```
    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
```

APEX SPECIALIST SUPER BADGE CODES

```
                                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;
```

APEX SPECIALIST SUPER BADGE CODES

```
        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 makesa 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>();
```

APEX SPECIALIST

SUPER BADGE CODES

```
        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();
    }


}

@isTest
```

WarehouseCalloutServiceMock.apxc:

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

APEX SPECIALIST SUPER BADGE CODES

```
        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 test here
          @isTest
   static void testWarehouseCallout() { test.startTest();
      test.setMock(HttpCalloutMock.class,new WarehouseCalloutServiceMock());
      WarehouseCalloutService.execute(null);
      test.stopTest();


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

APEX SPECIALIST SUPER BADGE CODES

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

WarehouseSyncSchedulueTest.apxc:

```
@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 Scheduleto Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
         /Contains schedule information for a scheduledjob. 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 ');


    }
}
```

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 static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';

PRIVATE STATICVehicle__c createVehicle(){
```

APEX SPECIALIST SUPER BADGE CODES

```
  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;
```

APEX SPECIALIST SUPER BADGE CODES

```
      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;
```

APEX SPECIALIST SUPER BADGE CODES

```
    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;
```

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 : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        updaterequestList;
        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);
    }
}
```

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(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 = 'RoutineMaintenance', 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));
```

APEX SPECIALIST SUPER BADGE CODES

```
            }

            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 makesa 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);
```

APEX SPECIALIST SUPER BADGE CODES

```
  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();
}
```

APEX SPECIALIST SUPER BADGE CODES

```
}
```

WarehouseCalloutServiceMock.apxc:

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

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"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
global classWarehouseCalloutServiceMock implements HttpCalloutMock {
    / implement http mock callout
   global static HttpResponse respond(HttpRequest request) {


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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"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;
    }
}
```

Challenge-6

WarehouseSyncSchedule.apxc:

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

WarehouseSyncScheduleTest.apxc:

```
@isTest
public class WarehouseSyncScheduleTest {


   @isTest static void WarehouseseheduleTest(){ String
      scheduleTime = '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 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 ');


   }
}
```