Apex Specialist Super Badge

APEX TRIGGERS

AccountAddressTrigger.apxt

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

ClosedOpportunityTrigger.apxt

```
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;
    }
}
```

APEX TESTING

VerifyDate.apxc

```
public class VerifyDate {
     //method to handle potential checks against two dates
     public static Date 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)) {
               return date2;
          } else {
               return SetEndOfMonthDate(date1);
          }
     }

     //method to check if date2 is within the next 30 days of
date1
     @TestVisible private static Boolean DateWithin30Days(Date
date1, Date date2) {
          //check for date2 being in the past
     if( date2 < date1) { return false; }

     //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 return the end of the month of a given date
     @TestVisible private static Date SetEndOfMonthDate(Date
date1) {
          Integer totalDays = Date.daysInMonth(date1.year(),
date1.month());
```

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

TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
    @isTest static void Test_CheckDates_case1(){
        Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/03
/2020'));
        System.assertEquals(date.parse('01/03/2020'),D);

    }
     @isTest static void Test_CheckDates_case2(){
        Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('03/03
/2020'));
        System.assertEquals(date.parse('01/31/2020'),D);

    }
}
```

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (before insert,before
update) {
     For (Contact c : Trigger.New) {
          if(c.LastName == 'INVALIDNAME') {  //invalidname is
invalid
               c.AddError('The Last Name "'+c.LastName+'" is not
allowed for DML');
          }
```

```
    }
}
```

TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {
    @isTest public static void testContact(){
        Contact c = new Contact();
        c.LastName = 'INVALIDNAME';
        Database.SaveResult res = Database.insert(c,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not
allowed for DML',res.getErrors()[0].getMessage());
    }
}
```

RandomContactFactory.apxc

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

AsynchronousApex

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
private 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

```
public without sharing class LeadProcessor implements
Database.Batchable<sObject>{
    public Database.QueryLocator start(Database.BatchableContext
dbc){
        return database.getQueryLocator([SELECT Id,Name FROM
Lead]);
    }
    public void execute(Database.BatchableContext dbc,List<Lead>
leads){
        for(Lead l: leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext dbc){
        System.debug('Done');
    }
}
```

LeadProcesorTest.apxc

```
@isTest
private class LeadProcessorTest{
    @isTest
    private static void testBatchClass(){
        List<Lead> leads = new List<Lead>();
      for(Integer i=0; i<200; i++){
            leads.add(new Lead(LastName = 'Connock',Company =
'Salesforce'));
        }
        insert leads;
```

```
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp,200);
        Test.stopTest();

        List<lead> updateLeads = [SELECT Id FROM Lead WHERE
LeadSource = 'Dreamforce'];
        System.assertEquals(200,updateLeads.size(),'ERROR: At
least 1 Lead record not updated correctly');
    }
}
```

AddPrimaryContact.apxc

```
public without sharing class AddPrimaryContact implements
Queueable{
     private Contact contact;
    private String state;

    public AddPrimaryContact(Contact inputcontact, String
inputState){
        this.contact = inputContact;
        this.state = inputState;
    }
    public void execute(QueueableContext context){
        List<Account> accounts = [SELECT Id FROM Account WHERE
BillingState = :state LIMIT 200];
        List<Contact> contacts = new List<Contact>();
        for(Account acc : accounts){
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }
```

```
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContacttest {
     @isTest
    private static void testQueueableClass(){
        List<Account>accounts = new List<Account>();
        for(Integer i=0; i<500; i++){
            Account acc = new Account(Name = 'Test Account');
            if(i<250){
                acc.BillingState = 'NY';
            }else{
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;
        Contact contact = new Contact(FirstName =
'Simon',LastName = 'Connock');
        insert contact;

        Test.startTest();
        Id jobId = System.enqueueJob(new
AddPrimaryContact(contact,'CA'));
        Test.stopTest();
        List<Contact> contacts = [SELECT Id FROM Contact WHERE
Contact.Account.BillingState = 'CA'];
        System.assertEquals(200,contacts.size(),'ERROR:
Incorrect number of Contact records found');
    }
}
```

DailyLeadProcessor.apxc

```
public Without sharing class DailyLeadProcessor implements
Schedulable{
    public void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id,LeadSource FROM Lead WHERE
LeadSource = null LIMIT 200];
        for(Lead l: leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }

}
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
     private static String CRON_EXP = '0 0 0 ? * * *';
    @isTest
    private static void testSchedulableClass(){
        List<Lead> leads = new List<Lead>();
        for(Integer i=0; i<500; i++){
            if(i<250){
                leads.add(new Lead(lastName='Connock',Company =
'Salesforce'));

            }else{
                leads.add(new Lead(LastName = 'Connock',Company
= 'Salesforce',LeadSource = 'Other'));
            }
        }insert leads;
        Test.startTest();
        String jobId = System.schedule('Process Leads',CRON_EXP,
new DailyLeadProcessor());
```

```
        Test.stopTest();

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM
Lead WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200,updatedLeads.size(),'ERROR: At
Least 1 record not updates correctly');

        List<CronTrigger> cts = [SELECT Id, TimesTriggered,
NextFireTime FROM CronTrigger WHERE Id =: jobId];
        System.debug('Next Fire Time '+cts[0].NextFireTime);
    }
}
```

Apex Integration Services

AnimalLocator.apxc

```
public class AnimalLocator{
    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');
    }
}
```

AnimalLocatorTest.apxc

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

AnimalLocatorMock.apxc

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

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

ParkLocator.apxc

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

```
}
```

ParkSerice.apxc

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-
1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new
String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
```

```
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new
ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse>
response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new
ParkServiceMock());
```

```
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

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) {
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String>
{'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

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

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
    @HttpGet
    global static Account getAccount(){
```

```
        RestRequest req = RestContext.request;
        String accId =
req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM
Contacts)
                       FROM Account WHERE Id = :accId];

        return acc;
    }
}
```

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        RestRequest request = new RestRequest();
        request.requestUri =

'https://ap5.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        Account  acc = AccountManager.getAccount();

        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;
```

```
        Contact con = new Contact(LastName = 'TestCont2',
AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

ApexSpecialist

Challenge-1

MaintenanceRequestHelper.apxc

```
public class MaintenanceRequestHelper {
     public static void updateWorkOrders(){
     Map<Id, Case> mantnceReqToEvaluate = new Map<Id, Case>();
     for(Case mantnceReq : (List<Case>)Trigger.new){
       if((mantnceReq.Type.contains('Repair') ||
mantnceReq.Type.contains('Routine Maintenance')) &&
mantnceReq.Status == 'Closed'){
         mantnceReqToEvaluate.put(mantnceReq.Id,mantnceReq);
       }
     }
         Map<Id, decimal> mapOfProdIdWithMaintenanceCycle =
getMapOfProdIdWithMaintenanceCycle();
         List<Case> lstOfMaintenanceRoutines =
getListOfMaintenanceRoutineList(mantnceReqToEvaluate,
mapOfProdIdWithMaintenanceCycle);
     System.debug('lstOfMaintenanceRoutines ::::::::
'+lstOfMaintenanceRoutines);
     if(lstOfMaintenanceRoutines != null &&
lstOfMaintenanceRoutines.size() > 0)
       INSERT lstOfMaintenanceRoutines;
   }
     private static Map<Id, decimal>
getMapOfProdIdWithMaintenanceCycle(){
```

```
     Map<Id,decimal> mapOfProdIdWithMaintenanceCycle = new
Map<Id, decimal>();
     for(Product2 prod : [SELECT Id, Maintenance_Cycle__c from
Product2]){
       mapOfProdIdWithMaintenanceCycle.put(prod.Id,
prod.Maintenance_Cycle__c);
     }
         return mapOfProdIdWithMaintenanceCycle;
   }
   private static List<Case>
getListOfMaintenanceRoutineList(Map<Id, Case>
mantnceReqToEvaluate, Map<Id, decimal>
mapOfProdIdWithMaintenanceCycle){
     List<Case> lstOfMaintenanceRoutines = new List<Case>();
       for(Case maintenance : mantnceReqToEvaluate.values()){
       Case maintenanceNewIns = new Case();
       maintenanceNewIns.Vehicle__c = maintenance.Vehicle__c;
       maintenanceNewIns.Equipment__c =
maintenance.Equipment__c;
       maintenanceNewIns.Type = 'Routine Maintenance';
       maintenanceNewIns.Subject = 'Your Routine Maintenance
Schedule';
       maintenanceNewIns.Date_Reported__c = Date.today();
       maintenanceNewIns.Date_Due__c = getDueDate(maintenance,
mapOfProdIdWithMaintenanceCycle);
       maintenanceNewIns.Status = 'New';
       maintenanceNewIns.Origin = 'Phone';
       lstOfMaintenanceRoutines.add(maintenanceNewIns);
     }
     return lstOfMaintenanceRoutines;
   }
   private static Date getDueDate(Case maintenance, Map<Id,
decimal> mapOfProdIdWithMaintenanceCycle){
     Date dt = null;
     if
```

```
(mapOfProdIdWithMaintenanceCycle.get(maintenance.Equipment__c)
!= null) {
       dt =
Date.today().addDays(Integer.valueOf(mapOfProdIdWithMaintenanceC
ycle.get(maintenance.Equipment__c)));
     }
     return dt;
   }
 }
```

MaintetanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update)
{
// ToDo: Call MaintenanceRequestHelper.updateWorkOrders
     MaintenanceRequestHelper.updateWorkOrders();
}
```

Challenge2

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-
superbadge-apex.herokuapp.com/equipment';
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
          HttpRequest request = new HttpRequest();
          request.setEndpoint(WAREHOUSE_URL);
          request.setMethod('GET');
          HttpResponse response = http.send(request);
          // If the request is successful, parse the JSON
response.
```

```
            if (response.getStatusCode() == 200) {
            // Deserialize the JSON string into collections of
primitive data types.
            List<Object> equipments = (List<Object>)
JSON.deserializeUntyped(response.getBody());
              List<Product2> products = new List<Product2>();
              for(Object o :  equipments){
                  Map<String, Object> mapProduct = (Map<String,
Object>)o;
                  Product2 product = new Product2();
                  product.Name = (String)mapProduct.get('name');
                  product.Cost__c =
(Integer)mapProduct.get('cost');
                  product.Current_Inventory__c =
(Integer)mapProduct.get('quantity');
                  product.Maintenance_Cycle__c =
(Integer)mapProduct.get('maintenanceperiod');
                  product.Replacement_Part__c =
(Boolean)mapProduct.get('replacement');
                  product.Lifespan_Months__c =
(Integer)mapProduct.get('lifespan');
                  product.Warehouse_SKU__c =
(String)mapProduct.get('sku');
                  product.ProductCode =
(String)mapProduct.get('_id');
                  products.add(product);
              }
              if(products.size() > 0){
                  System.debug(products);
                  upsert products;
              }
            }
    }
}
```

WarehouseCallutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutSeviceTest.apxc

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new
WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

Challenge3

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 WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule
to Test', scheduleTime, new WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job.
CronTrigger is similar to a cron job on UNIX systems.
        // This object is available in API version 17.0 and
later.
        CronTrigger a=[SELECT Id FROM CronTrigger where
NextFireTime > today];
```

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


    }
}
```

Challenge4

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 = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()

                );

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

                newCases.add(nc);
            }
```

```
            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c wpClone =
wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);

                }
            }
            insert ClonedWPs;
        }
    }
}
```

MaintenanceRequestHelperTest.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';
    private static final string REQUEST_SUBJECT = 'Testing
subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
```

```
        Vehicle__c Vehicle = new Vehicle__C(name =
'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name =
'SuperEquipment',
                                          lifespan_months__C =
10,
                                          maintenance_cycle__C =
10,
                                          replacement_part__c =
true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId,
id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c
createWorkPart(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,

Maintenance_Request__c = requestId);
        return wp;
```

```
    }


    @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

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

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

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

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

        Case newReq = [Select id, subject, type, Equipment__c,
Date_Reported__c, Vehicle__c, Date_Due__c
                       from case
                       where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from
Equipment_Maintenance_Item__c
                                                  where
```

```
Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c,
system.today());
    }

    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

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

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

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

        list<case> allRequest = [select id
```

```
                                    from case];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from
Equipment_Maintenance_Item__c
                                                  where
Maintenance_Request__c = :emptyReq.Id];

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

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

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

        for(integer i = 0; i < 300; i++){

requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert requestList;
```

```
        for(integer i = 0; i < 300; i++){

workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select
id
                                                         from
Equipment_Maintenance_Item__c
                                                         where
Maintenance_Request__c in: oldRequestIds];

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

Challenge5

WarehouseCalloutServiceTest.apxc

```
@isTest
private class WarehouseCalloutServiceTest {
```

```
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new
WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Challenge6

WarehouseSuncScheduleTest.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 Schedule
to Test', scheduleTime, new WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job.
CronTrigger is similar to a cron job on UNIX systems.
        // This object is available in API version 17.0 and
later.
        CronTrigger a=[SELECT Id FROM CronTrigger where
NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```