APEX CLASSES

Account manager :

```
@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;
    }
}
```

Account Manager Test :

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            
'https://ap5.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        
        // Call the method to test
        Account  acc = AccountManager.getAccount();
        
        // Verify results
        System.assert(acc != null);
    }
    
    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;
        
        Contact con = new Contact(LastName = 'TestCont2',
AccountId = acc.Id);
         Insert con;
        
         return acc.Id;
    }
}
```

Account Processor :

```
public class AccountProcessor {
    
    @future
    public static void countContacts(List<Id> accountId_lst) {
        
        Map<Id,Integer> account_cno = new Map<Id,Integer>();
        List<account> account_lst_all = new List<account>([select
id, (select id from contacts) from account]);
        for(account a:account_lst_all) {
            account_cno.put(a.id,a.contacts.size()); //populate the
map
            
        }
        
        List<account> account_lst = new List<account>(); // list of
account that we will upsert
        
        for(Id accountId : accountId_lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c =
account_cno.get(accountId);
                account_lst.add(acc);
            }
        }
        upsert account_lst;
    }

}
```

Account Processor Test :

```
@isTest
public class AccountProcessorTest {
    
    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;
        
        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;
        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
          AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts
from account where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }

}
```

Add Primary Contact :

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }
    
    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, 
BillingState from account where account.BillingState = :this.state 
limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}
```

Add Primary Contact Test :

```
@IsTest
public class AddPrimaryContactTest {
    
    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new
account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new
account(name=string.valueOf(50+i),billingstate='CA'
);
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new
AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select
id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }

}
```

Animal Locator :

```
public class AnimalLocator {
  public class cls_animal {
    public Integer id;
    public String name;
    public String eats;
    public String says;
  }
public class JSONOutput{
  public cls_animal animal;

     //public JSONOutput parse(String json){
  //return (JSONOutput) System.JSON.deserialize(json,
JSONOutput.class);
  //}
}

    public static String getAnimalNameById (Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/' + id);
        //request.setHeader('id', String.valueof(id)); -- cannot be
used in this challenge :)
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        system.debug('response: ' + response.getBody());
        //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
        jsonOutput results = (jsonOutput)
JSON.deserialize(response.getBody(), jsonOutput.class);
        //Object results = (Object) map_results.get('animal');
    system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }

}
```

Animal Locator Test :

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
        //Httpresponse response =
AnimalLocator.getAnimalNameById(1);
        String s =
AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

Animal Locator Mock :

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger',
'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chick
en food","says":"cluck cluck"}}');
        return response;
    }

}
```

Async Park Service :

```
public class AsyncParkService {
    public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.en
dInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new
ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
      }
  }
}
```

Daily Lead Processor :

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead
WHERE LeadSource = ''];
        
        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();
            
            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }
            
            update newLeads;
        }
    }
}
```

Daily Lead Processor Test :

```
    @isTest
private class DailyLeadProcessorTest{
  //Seconds Minutes Hours Day_of_month Month Day_of_week
optional_year
  public static String CRON_EXP = '0 0 0 2 6 ? 2022';

  static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();

      for(Integer i = 0; i < 200; i++){
          Lead lead = new Lead(LastName = 'Test ' + i, LeadSource
= '', Company = 'Test Company ' + i, Status = 'Open - Not
Contacted');
          leads.add(lead);
      }
```

```
        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to
DreamForce', CRON_EXP, new DailyLeadProcessor());

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

Lead Processor :

```
public class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {
    public LeadProcessor() {

    }

    public Database.QueryLocator
start(Database.BatchableContext BC) {
        String query = 'SELECT Id FROM Lead';
        return Database.getQueryLocator (query);
    }
    public void execute(Database.BatchableContext BC,
List<Lead> leads) {
        for (Lead l : leads) {
            l.LeadSource  = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext BC) {
        
    }
}
```

Lead Processor Test :

```
@isTest
private class LeadProcessorTest {

    private static User testAdminUser = new User(Id =
UserInfo.getUserId());

    static testMethod void LeadProcessorTest() {
        
        System.runAs(testAdminUser) {
            
            List<Lead> leads = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                leads.add(new Lead(LastName = 'Yoshikawa',
Company = 'T.Yoshikawa Labs'));
            }
            insert leads;
            System.assertEquals(leads.size(), 200);

            Test.startTest();

            LeadProcessor batchable = new LeadProcessor();
            Database.executeBatch(batchable);

            Test.stopTest();

            List<Lead> results = [SELECT Id,LeadSource FROM
Lead];
            for (Lead l : results) {
                System.assertEquals(l.LeadSource, 'Dreamforce');
            }
            System.assertEquals(results.size(), 200);
        }
    }
}
```

Park Locator :

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

Park Locator Test :

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new
ParkServiceMock());
        String[] arrayOfParks =
ParkLocator.country('India');
        
        System.assertEquals('Park1',
arrayOfParks[0]);
    }
}
```

Park Service :

```
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;
        }
    }
}
```

Park Service Mock:

```
@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);
    }
}
```

Random Contact Factory :

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i,
LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }

}
```

Test Restrict Contact By Name :

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        //try inserting a Contact with INVALIDNAME
        Contact myConact = new
Contact(LastName='INVALIDNAME');
        insert myConact;
        
        // Perform test
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact,
false);
        Test.stopTest();
        // Verify
        // In this case the creation should have been stopped by the
trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last
name.',
                              result.getErrors()[0].getMessage());
        
    }
}
```

Test Verify Date :

```
@isTest
private class TestVerifyDate {

    //testing that if date2 is within 30 days of date1, should return
date 2
    @isTest static void testDate2within30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 04, 11);
        System.assertEquals(testDate,resultDate);
    }
    
    //testing that date2 is before date1. Should return "false"
    @isTest static void testDate2beforeDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 02, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }
    
    //Test date2 is outside 30 days of date1. Should return end of
month.
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

APEX TRIGGERS

Account Address Trigger :

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

Closed Opportunity Trigger :

```
 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;
    }
}
```

Restrict Contact By Name Trigger :

```
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 allowed
for DML');
```

```
  }

 }
```

APEX SUPER BADGE

Challenge 1: Automate record creation

Maintenance Request Helper Class :

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> caseList) {
List<case> newCases = new List<Case>();
Map<String,Integer> result=getDueDate(caseList);
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case>
CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<AggregateResult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
from Work_Part__c where  Maintenance_Request__r.ID in
:caseKeys.keySet() group by            Maintenance_Request__r.ID
];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
```

}

Maintenance Request Trigger :

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

Challenge 2 : Synchronize Salesforce data with external system

Warehouse Call Out Service Class :

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

    //Write a class that makes a REST callout to an external
```

```
warehouse system to get a list of equipment that needs to be
updated.
    //The callout’s JSON response returns the equipment records
that you upsert in Salesforce.
    
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        
        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            
            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which
equipment records to update within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson =
(Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double)
mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String)
mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the 
warehouse one');
            }
        }
    }
    
    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }
    
}
```

```
WarehouseCalloutService.runWarehouseEquipmentSync();
(Executed code in anonymous window)
```

Challenge 3 : Schedule Synchronization

WarehouseSync Schedule Class :

```
global  class WarehouseSyncSchedule implements Schedulable{
// implement scheduled code here
global  void execute (SchedulableContext sc){
WarehouseCalloutService.runWarehouseEquipmentSync();
//optional this can be done by debug mode
String sch = '00 00 01 * * ?';//on 1 pm
System.schedule('WarehouseSyncScheduleTest', sch, new
WarehouseSyncSchedule());
}
}
```

WarehouseSyncSchedule scheduleInventoryCheck(); (Executed code in anonymous window)

Challenge 4 : Test Automation Logic

Maintenance Request Trigger :

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

Maintenance Request Test Class :

```
@isTest
public  class MaintenanceRequestTest {
static  List<case> caseList1 = new List<case>();
static List<product2> prodList = new List<product2>();
static List<work_part__c> wpList = new List<work_part__c>();
@testSetup
static void getData(){
caseList1= CreateData( 300,3,3,'Repair');
```

```
}
public static List<case>   CreateData( Integer numOfcase, Integer
numofProd, Integer numofVehicle,
String type){
List<case> caseList = new List<case>();
//Create Vehicle
Vehicle__c vc = new Vehicle__c();
vc.name='Test Vehicle';
upsert vc;
//Create Equiment
for(Integer i=0;i<numofProd;i++){
Product2 prod = new Product2();
prod.Name='Test Product'+i;
if(i!=0)
prod.Maintenance_Cycle__c=i;
prod.Replacement_Part__c=true;
prodList.add(prod);
}
upsert  prodlist;
//Create Case
for(Integer i=0;i< numOfcase;i++){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
if( math.mod(i, 2) ==0)
newCase.Type='Routine Maintenance';
else
newCase.Type='Repair';
newCase.Subject='Routine Maintenance of Vehicle' +i;
newCase.Vehicle__c=vc.Id;
if(i<numofProd)
newCase.Equipment__c=prodList.get(i).ID;
else
newCase.Equipment__c=prodList.get(0).ID;
caseList.add(newCase);
}
upsert caseList;
for(Integer i=0;i<numofProd;i++){
Work_Part__c wp = new Work_Part__c();
wp.Equipment__c   =prodlist.get(i).Id   ;
wp.Maintenance_Request__c=caseList.get(i).id;
wplist.add(wp) ;
}
upsert wplist;
return caseList;
}
public static testmethod void testMaintenanceHelper(){
Test.startTest();
getData();
for(Case cas: caseList1)
cas.Status ='Closed';
update caseList1;
Test.stopTest();
}
```

```
}
```

Challenge 5 : Test Call Out Logic

Warehouse Call Out Service Test Class :

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

Warehouse Call Out Service Mock :

```
@isTest
public class WarehouseCalloutServiceMock implements
HTTPCalloutMock {
// implement http mock callout
public HTTPResponse respond (HttpRequest request){
HttpResponse response = new HTTPResponse();
response.setHeader('Content-type','application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replace
ment":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"},{"_id":"55d66226726b611100aaf742","replacement":true,"q
uantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004
"},{"_id":"55d66226726b611100aaf743","replacement":true,"quanti
ty":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}
]');
response.setStatusCode(200);
return response;
}
}
```

Challenge 6 : Test Scheduling Logic

WarehouseSync Schedule Test Class :

```
@isTest
private class WarehouseSyncScheduleTest {
public static String CRON_EXP = '0 0 0 15 3 ? 2022';
static testmethod void testjob(){
MaintenanceRequestTest.CreateData( 5,2,2,'Repair');
Test.startTest();
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
String joBID= System.schedule('TestScheduleJob', CRON_EXP,
new WarehouseSyncSchedule());
// List<Case> caselist = [Select count(id) from case where case]
Test.stopTest();
}
}
```