1)Apex REST Callouts

//AnimalLocator Class

```
1  public class AnimalLocator{
1      public static String getAnimalNameById(Integer x){
2          Http http = new Http();
3          HttpRequest req = new HttpRequest();
4                          req.setEndpoint('https://th-apex-http-

5          req.setMethod('GET');
6                Map<String, Object> animal = new Map<String,
   Object>();
7         HttpResponse res = http.send(req);
8    if(res.getStatusCode() == 200) {
9    Map<String,Object>results=(Map<String,
   Object>)JSON.deserializeUntyped(res.getBody());
10        animal = (Map<String, Object>) results.get('animal');
11              }
12 return (String)animal.get('name');
13          }
14  }
```

//AnimalLocatorTest Class

```
1  @isTest
2  public class AnimalLocatorTest {
3      @isTest static void AnimalLocatorMock1() {
4                        Test.setMock(HttpCalloutMock.class,  new
   AnimalLocatorMock());
5          string result = AnimalLocator.getAnimalNameById(3);
6          String expectedResult = 'chicken';
7          System.assertEquals(result,expectedResult );
```

```
    }
}
```

//AnimalLocatorMock Class

```
@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');
            response.setBody('{"animals": ["majestic badger",


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

2)Apex SOAP Callouts

//ParkLocator Class

```
public class ParkLocator {
    public static List<String> country(String country) {
                ParkService.ParksImplPort  parkservice  =  new
parkService.ParksImplPort();
        return parkService.byCountry(country);
    }
}
```

//ParkLocatorTest Class

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
                    Test.setMock(WebServiceMock.class,  new
ParkServiceMock());
        // Call the method that invokes a callout
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>();
    parks.add('Yosemite');
              parks.add('Yellowstone');
              parks.add('Another Park');
        // Verify that a fake result is returned
        System.assertEquals(parks, result);
    }
}
```

//ParkServiceMock Class

```
@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
```

```
        List<String> parks = new List<string>();
                parks.add('Yosemite');
                parks.add('Yellowstone');
                parks.add('Another Park');
        ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();
        response_x.return_x = parks;
        // end
        response.put('response_x', response_x);
    }
}
```

3)ApexWebServices

//AccountManager Class

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
                                    String    accountId    =
  request.requestURI.substringBetween('Accounts/','/contacts'
  );
           Account result =  [SELECT Id, Name, (Select Id,
  Name from Contacts) from Account where Id=:accountId];
        return result;
    }
}
```

//AccountManagerTest Class

```
1 @IsTest
2 private class AccountManagerTest {
3     @isTest static void testGetContactsByAccountId() {
4         Id recordId = createTestRecord();
5         // Set up a test request
6         RestRequest request = new RestRequest();
7 request.requestUri=
  'https://yourInstance.salesforce.com/services/apexrest/Acco

8         request.httpMethod = 'GET';
9         RestContext.request = request;
10        // Call the method to test
11        Account thisAccount = AccountManager.getAccount();
12        // Verify results
13        System.assert(thisAccount != null);
14                        System.assertEquals('Test   record',
  thisAccount.Name);
15    }
16    // Helper method
17    static Id createTestRecord() {
18
19// Create test record
20        Account accountTest = new Account(
21            Name='Test record');
22        insert accountTest;
23
24        Contact contactTest = new Contact(
25 FirstName='John',
26            LastName='Doe',
27            AccountId=accountTest.Id);
28        insert contactTest;
```

```
        return accountTest.Id;
    }
}
```

4)copy processes with queueable apex

//class AddPrimaryContact

```
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;
```

```
        }
    }
```

//class AddPrimaryContactTest

```
@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

  }
  for(Integer j=0;j<50;j++){
  testAccounts.add(new Account(Name='Account

  }
  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')]);
    }
}

}
```

5)Schedule Jobs Using the Apex Scheduler

//DailyLeadProcessor Class

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext SC){
            List<Lead>  LeadObj=[SELECT  Id  FROM  Lead  where
LeadSource=null limit 200];
        for(Lead l:LeadObj){
            l.LeadSource='Dreamforce';
 update l;
        }
       }
}
```

//DailyLeadProcessorTest Class

```
@isTest
private class DailyLeadProcessorTest {
    static testMethod void testDailyLeadProcessor() {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<lead>();
        for (Integer i = 0; i < 200; i++) {
                lList.add(new  Lead(LastName='Dreamforce'+i,
Company='Test1 Inc.', Status='Open - Not Contacted'));
        }
        insert lList;
```

```
10        Test.startTest();
11    String  jobId  =  System.schedule('DailyLeadProcessor',
  CRON_EXP, new DailyLeadProcessor());
12
13    }
14}
```

6)Use Batch apex

//Class LeadProcessor

```
1 global        class        LeadProcessor        implements
  Database.Batchable<sObject> {
2     global Integer count = 0;
3
4                           global      Database.QueryLocator
  start(Database.BatchableContext bc) {
5               return  Database.getQueryLocator('SELECT  ID,

6     }
7
8       global  void  execute  (Database.BatchableContext  bc,
```

```
   List<Lead> L_list) {
9          List<lead> L_list_new = new List<lead>();
10
11         for(lead L:L_list){
12             L.leadsource = 'Dreamforce';
13             L_list_new.add(L);
14             count += 1;
15         }
16          update L_list_new;
17     }
18     global void finish(Database.BatchableContext bc) {
19         system.debug('count = ' + count);
20     }
21
22}
```

//Class LeadProcessorTest

```
1 @isTest
2 public class LeadProcessorTest {
3     @isTest
4     public static void testit(){
5         List<lead> L_list = new List<lead>();
6         for(Integer i=0; i<200; i++){
7             Lead L = new lead();
8             L.LastName = 'name' + 1;
9             L.Company = 'Company';
10            L.Status = 'Random Status';
11            L_list.add(L);
12        }
13        insert L_list;
14
15        Test.startTest();
```

```
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

7)USE FUTURE METHOD

//OPEN EXECUTE ANONYMOUS WINDOW
// Class AccountProcessor

```
List<Id> accountIds = new List<Id>();
accountIds.add('001Iw000002RZIKIA4');

AccountProcessor.countContacts(accountIds);

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);

        }
```

```
20 update accountsToUpdate;
21     }
22}
```

//Class AccountProcessorTest

```
1 @isTest
2 private class AccountProcessorTest {
3  @isTest
4     Private static void testCountContacts(){
5               Account  newAccount  =  new  Account(Name='Test

6         insert newAccount;
7
8                           Contact    newContact1    =    new
  Contact(FirstName='John',LastName='Doe',AccountId          =
  newAccount.Id);
9         insert newContact1;
10
11                          Contact    newContact2    =    new
  Contact(FirstName='Jane',LastName='Doe',AccountId          =
  newAccount.Id);
12        insert newContact2;
13
14        List<Id> accountIds = new List<Id>();
15        accountIds.add(newAccount.Id);
16
17        Test.startTest();
18        AccountProcessor.countContacts(accountIds);
19        Test.stopTest();
20
```

```
    }
}
```

8)Apex Specialist Superbagde

Task:Automate record creation

//MaintenanceRequest Trigger

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

//Class MaintenanceRequestHelper

```
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);
                }
            }
```

```
        }
//When an existing maintenance request of type Repair or Routine Maintenance is //closed,
          //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
                        Map<Id,Case> closedCases = 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>();

              //calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.
                      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'));
            }

```

```
29              List<Case> newCases = new List<Case>();
30              for(Case cc : closedCases.values()){
31                  Case nc = new Case (
32                      ParentId = cc.Id,
33                      Status = 'New',
34                      Subject = 'Routine Maintenance',
35                      Type = 'Routine Maintenance',
36                      Vehicle__c = cc.Vehicle__c,
37                      Equipment__c =cc.Equipment__c,
38                      Origin = 'Web',
39                      Date_Reported__c = Date.Today()
40                  );
41
42                    //If multiple pieces of equipment are used
  in the maintenance request,
43                      //define the due date by applying the
  shortest maintenance cycle to today's date.
44                  If (maintenanceCycles.containskey(cc.Id)){
45                                              nc.Date_Due__c  =
  Date.today().addDays((Integer)
  maintenanceCycles.get(cc.Id));
46                  } else {
47                                              nc.Date_Due__c  =
  Date.today().addDays((Integer)
  cc.Equipment__r.maintenance_Cycle__c);
48                  }
49
50                  newCases.add(nc);
51              }
52
53              insert newCases;
54
55                List<Equipment_Maintenance_Item__c> clonedList
  = new List<Equipment_Maintenance_Item__c>();
56              for (Case nc : newCases){
57                          for  (Equipment_Maintenance_Item__c
  clonedListItem                                            :
```

```
    closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r
    ){
58                          Equipment_Maintenance_Item__c item =
    clonedListItem.clone();
59                      item.Maintenance_Request__c = nc.Id;
60                      clonedList.add(item);
61                  }
62              }
63              insert clonedList;
64          }
65      }
66  }
67
68  Task:Synchronize Salesforce data with an external system
69
70  public    with    sharing    class    WarehouseCalloutService
    implements Queueable {
71          private  static  final  String  WAREHOUSE_URL  =
    'https://th-superbadge-apex.herokuapp.com/equipment';
72
73          //Write a class that makes a REST callout to an
    external warehouse system to get a list of equipment that
    needs to be updated.
74          //The callout's JSON response returns the equipment
    records that you upsert in Salesforce.
75
76      @future(callout=true)
77      public static void runWarehouseEquipmentSync(){
78          System.debug('go into runWarehouseEquipmentSync');
79          Http http = new Http();
80          HttpRequest request = new HttpRequest();
81
82          request.setEndpoint(WAREHOUSE_URL);
83          request.setMethod('GET');
```

```
84          HttpResponse response = http.send(request);
85
86          List<Product2> product2List = new List<Product2>();
87          System.debug(response.getStatusCode());
88          if (response.getStatusCode() == 200){
89                                  List<Object>  jsonResponse  =
  (List<Object>)JSON.deserializeUntyped(response.getBody());
90              System.debug(response.getBody());
91
92              //class maps the following fields:
93                    //warehouse SKU will be external ID for
  identifying  which  equipment  records  to  update  within
  Salesforce
94              for (Object jR : jsonResponse){
95                                  Map<String,Object>  mapJson  =
  (Map<String,Object>)jR;
96                  Product2 product2 = new Product2();
97                  //replacement part (always true),
98                      product2.Replacement_Part__c = (Boolean)
  mapJson.get('replacement');
99                  //cost
100                                 product2.Cost__c  =  (Integer)
  mapJson.get('cost');
101                 //current inventory
102                     product2.Current_Inventory__c = (Double)
  mapJson.get('quantity');
103                 //lifespan
104                      product2.Lifespan_Months__c = (Integer)
  mapJson.get('lifespan');
105                 //maintenance cycle
106                   product2.Maintenance_Cycle__c = (Integer)
  mapJson.get('maintenanceperiod');
```

```
107                 //warehouse SKU
108                       product2.Warehouse_SKU__c = (String)
  mapJson.get('sku');
109
110                                   product2.Name  =  (String)
  mapJson.get('name');
111                             product2.ProductCode  =  (String)
  mapJson.get('_id');
112                 product2List.add(product2);
113             }
114
115             if (product2List.size() > 0){
116                 upsert product2List;
117                     System.debug('Your equipment was synced

118             }
119         }
120     }
121
122     public static void execute (QueueableContext context){
123         System.debug('start runWarehouseEquipmentSync');
124         runWarehouseEquipmentSync();
125         System.debug('end runWarehouseEquipmentSync');
126     }
127
128}
129
130
131//debug part to generate interface
132System.enqueueJob(new WarehouseCalloutService());
```

Task:Schedule synchronization

//Class WarehouseSyncSchedule

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

Task:Test automation logic

//Class MaintenanceRequestHelper

```
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);
                }
            }
        }

           //When an existing maintenance request of type Repair or Routine Maintenance is closed,
          //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
                        Map<Id,Case>  closedCases  =  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>();

            //calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.
                    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'));
            }

            List<Case> newCases = new List<Case>();
            for(Case cc : closedCases.values()){
                Case nc = new Case (
```

```
33                      ParentId = cc.Id,
34                      Status = 'New',
35                      Subject = 'Routine Maintenance',
36                      Type = 'Routine Maintenance',
37                      Vehicle__c = cc.Vehicle__c,
38                      Equipment__c =cc.Equipment__c,
39                      Origin = 'Web',
40                      Date_Reported__c = Date.Today()
41                  );
42
43                    //If multiple pieces of equipment are used in the maintenance request,
44                      //define the due date by applying the shortest maintenance cycle to today’s date.
45                                                              //If (maintenanceCycles.containskey(cc.Id)){
46                                                  nc.Date_Due__c  = Date.today().addDays((Integer)
  maintenanceCycles.get(cc.Id));
47                  //} else {
48                                      //        nc.Date_Due__c  = Date.today().addDays((Integer)
  cc.Equipment__r.maintenance_Cycle__c);
49                  //}
50
51                  newCases.add(nc);
52              }
53
54              insert newCases;
55
56                List<Equipment_Maintenance_Item__c> clonedList = new List<Equipment_Maintenance_Item__c>();
```

```
            for (Case nc : newCases){
                        for  (Equipment_Maintenance_Item__c clonedListItem                                                    : closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r ){
                    Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
            }
            insert clonedList;
        }
    }
}
```

//Class MaintenanceRequestHelperTest

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing

        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
          product2 equipment = new product2(name = 'Testing
```

```
13
  lifespan_months__c = 10,
14
  maintenance_cycle__c = 10,
15
  replacement_part__c = true);
16        return equipment;
17    }
18
19    // createMaintenanceRequest
20         private  static  Case  createMaintenanceRequest(id
  vehicleId, id equipmentId){
21        case cse = new case(Type='Repair',
22                            Status='New',
23                            Origin='Web',
24                            Subject='Testing subject',
25                            Equipment__c=equipmentId,
26                            Vehicle__c=vehicleId);
27        return cse;
28    }
29
30    // createEquipmentMaintenanceItem
31           private   static   Equipment_Maintenance_Item__c
  createEquipmentMaintenanceItem(id            equipmentId,id
  requestId){
32                              Equipment_Maintenance_Item__c
  equipmentMaintenanceItem                =                new
  Equipment_Maintenance_Item__c(
33           Equipment__c = equipmentId,
34           Maintenance_Request__c = requestId);
35        return equipmentMaintenanceItem;
```

```
36      }
37
38      @isTest
39      private static void testPositive(){
40          Vehicle__c vehicle = createVehicle();
41          insert vehicle;
42          id vehicleId = vehicle.Id;
43
44          Product2 equipment = createEquipment();
45          insert equipment;
46          id equipmentId = equipment.Id;
47
48                                      case     createdCase     =
    createMaintenanceRequest(vehicleId,equipmentId);
49          insert createdCase;
50
51                                  Equipment_Maintenance_Item__c
    equipmentMaintenanceItem                                    =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
52          insert equipmentMaintenanceItem;
53
54          test.startTest();
55          createdCase.status = 'Closed';
56          update createdCase;
57          test.stopTest();
58
59          Case newCase = [Select id,
60                          subject,
61                          type,
62                          Equipment__c,
63                          Date_Reported__c,
64                          Vehicle__c,
```

```
                        Date_Due__c
                       from case
                       where status ='New'];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from
  Equipment_Maintenance_Item__c
                                                 where
  Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
                system.assertEquals(newCase.Type,  'Routine

                  SYSTEM.assertEquals(newCase.Equipment__c,
  equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
              SYSTEM.assertEquals(newCase.Date_Reported__c,
  system.today());
    }

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

        product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;
```

```
92
93                                  case    createdCase    =
  createMaintenanceRequest(vehicleId,equipmentId);
94          insert createdCase;
95
96                      Equipment_Maintenance_Item__c   workP   =
  createEquipmentMaintenanceItem(equipmentId,
  createdCase.Id);
97          insert workP;
98
99          test.startTest();
100          createdCase.Status = 'Working';
101          update createdCase;
102          test.stopTest();
103
104          list<case> allCase = [select id from case];
105
106                               Equipment_Maintenance_Item__c
  equipmentMaintenanceItem = [select id
107                                                    from
  Equipment_Maintenance_Item__c
108                                                   where
  Maintenance_Request__c = :createdCase.Id];
109
110          system.assert(equipmentMaintenanceItem != null);
111          system.assert(allCase.size() == 1);
112      }
113
114      @isTest
115      private static void testBulk(){
116                      list<Vehicle__C>   vehicleList   =   new
  list<Vehicle__C>();
```

```
117                     list<Product2>   equipmentList   =   new
  list<Product2>();
118                         list<Equipment_Maintenance_Item__c>
  equipmentMaintenanceItemList              =              new
  list<Equipment_Maintenance_Item__c>();
119         list<case> caseList = new list<case>();
120         list<id> oldCaseIds = new list<id>();
121
122         for(integer i = 0; i < 300; i++){
123             vehicleList.add(createVehicle());
124             equipmentList.add(createEquipment());
125         }
126         insert vehicleList;
127         insert equipmentList;
128
129         for(integer i = 0; i < 300; i++){
130
  caseList.add(createMaintenanceRequest(vehicleList.get(i).i

131         }
132         insert caseList;
133
134         for(integer i = 0; i < 300; i++){
135
  equipmentMaintenanceItemList.add(createEquipmentMaintenance

136         }
137         insert equipmentMaintenanceItemList;
138
139         test.startTest();
140         for(case cs : caseList){
141             cs.Status = 'Closed';
```

```
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                              from case
                              where status ='New'];

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

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

        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

//Class MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after
  update) {
    if(Trigger.isUpdate && Trigger.isAfter){

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

```
}
```

Task:Test callout logic

//Class WarehouseCalloutService

```
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());
```

```
22
23            //class maps the following fields:
24                  //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
25            for (Object jR : jsonResponse){
26                                  Map<String,Object> mapJson = (Map<String,Object>)jR;
27                Product2 product2 = new Product2();
28                //replacement part (always true),
29                    product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
30                //cost
31                                  product2.Cost__c = (Integer) mapJson.get('cost');
32                //current inventory
33                    product2.Current_Inventory__c = (Double) mapJson.get('quantity');
34                //lifespan
35                     product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
36                //maintenance cycle
37                   product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
38                //warehouse SKU
39                        product2.Warehouse_SKU__c = (String) mapJson.get('sku');
40
41                                      product2.Name = (String) mapJson.get('name');
42                              product2.ProductCode = (String) mapJson.get('_id');
```

```
                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                    System.debug('Your equipment was synced

            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }
}
```

//Class WarehouseCalloutServiceMock

```
@isTest
global    class    WarehouseCalloutServiceMock    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","repla
```

```
        response.setStatusCode(200);

        return response;
    }
}
```

//Class WarehouseCalloutServiceTest

```
@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 = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
```

```
15              System.assertEquals('55d66226726b611100aaf741',
  product2List.get(0).ProductCode);
16              System.assertEquals('55d66226726b611100aaf742',
  product2List.get(1).ProductCode);
17              System.assertEquals('55d66226726b611100aaf743',
  product2List.get(2).ProductCode);
18      }
19}
```

Task: Test scheduling logic

//Class WarehouseCalloutServiceMock

```
1 @isTest
2 global    class    WarehouseCalloutServiceMock    implements
  HttpCalloutMock {
3     // implement http mock callout
4      global static HttpResponse respond(HttpRequest request)
  {
5
6         HttpResponse response = new HttpResponse();
7                             response.setHeader('Content-Type',
  'application/json');
8
  response.setBody('[{"_id":"55d66226726b611100aaf741","repla
```

```
        response.setStatusCode(200);

        return response;
    }
}
```

//Class WarehouseSyncSchedule

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


//Class
@isTest WarehouseSyncScheduleTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
                    Test.setMock(HttpCalloutMock.class,  new WarehouseCalloutServiceMock());
          String jobId = System.schedule('Warehouse Time to

WarehouseSyncSchedule());
```

```
19            CronTrigger  c  =  [SELECT  State  FROM  CronTrigger
  WHERE Id =: jobId];
20                                System.assertEquals('WAITING',
  String.valueOf(c.State), 'JobId does not match');
21
22          Test.stopTest();
23      }
24}
```

9)Process Automation Specialist SuperBadge

Task: Automate Leads

```
1 //[IfUsOrNot]
2 OR(
3 NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:


  BillingState)),
4 LEN(State) <> 2,
5 NOT(OR(Country    ="US",Country    ="USA",Country    ="United

6 )
```

Task: Automate Accounts

```
//[ValidationForBilling]
OR(
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:
BillingState)),
LEN(BillingState) <> 2,
NOT(OR(BillingCountry ="US",BillingCountry
="USA",BillingCountry ="United States",
ISBLANK(BillingCountry))),
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:
ShippingState)),
LEN(ShippingState) <> 2,
NOT(OR(ShippingCountry ="US",ShippingCountry
="USA",ShippingCountry ="United States",
ISBLANK(ShippingCountry )))
)

//[ValidationForType]
ISCHANGED(Name)  &&  (OR(ISPICKVAL(Type,  'Customer  -
```

Task: Create Robot Setup Object

```
CASE  (weekday(Date__c),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
```

```
5,"Thusday",
6,"Friday",
7,"Saturday",
Text(weekday(Date__c))
)
```

Task: Create Sales Process and Validate Opportunities

```
//ValidationForHighValue
if((Amount > 1000 && Approved__c = false &&
ispickval(stageName,"Closed Won")),true,false)
```

Task: Automate Setup

```
CASE(
MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7),
 0, [Opportunity].CloseDate + 181,
 6, [Opportunity].CloseDate + 182,
 [Opportunity].CloseDate + 180
)
```