CODES FOR HANDS-ON CHALLENGES IN SALESFORCE SELF LEARNING:

Apex triggers module:

Get Started with Apex Triggers Challenge:

1)AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert,before update) {

    for(Account acct:Trigger.new)
    {
        if(acct.Match_Billing_Address__c == True)
            acct.ShippingPostalCode = Acct.BillingPostalCode;
    }
}
```

Bulk Apex Triggers Challenge:

2) ClosedOpportunityTrigger.aptxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update)
{

    List<Task> taskList = new List<Task>();

    for(Opportunity opp : Trigger.new) {

        if(Trigger.isInsert) {
            if(Opp.StageName == 'Closed Won') {
                taskList.add(new Task(Subject = 'Follow Up Test Task',
WhatId = opp.Id));
            }
        }
        if(Trigger.isUpdate) {
            if(Opp.StageName == 'Closed Won' && Opp.StageName !=
Trigger.oldMap.get(opp.Id).StageName) {
                taskList.add(new Task(Subject = 'Follow Up Test Task',
WhatId = opp.Id));
```

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

APEX TESTING MODULE

Get Started with Apex Unit Tests:

1) VerifyDate.apxc

```
      public class VerifyDate {

      public static Date CheckDates(Date date1, Date date2) {
            if(DateWithin30Days(date1,date2)) {
                  return date2;
            } else {
                  return SetEndOfMonthDate(date1);
            }
      }

      private static Boolean DateWithin30Days(Date date1, Date date2) {
      if( date2 < date1) { return false; }

      Date date30Days = date1.addDays(30);
            if( date2 >= date30Days ) { return false; }
            else { return true; }
      }

      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;
      }

}
```

2) TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
    static testMethod void TestVerifyDate() {
      VerifyDate.CheckDates(System.today(),System.today().addDays(10));
       VerifyDate.CheckDates(System.today(),System.today().addDays(78));
    }
}
```

Test Apex Triggers:

1) RestrictContactByName.apxt

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

        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }
        }
}
```

2) TestRestrictContactByName.apxc

```
@isTest
private class TestRestrictContactByName {
    @isTest static void metodoTest()
    {
        Contact c = new Contact(LastName = 'INVALIDNAME');
        Database.SaveResult result = Database.insert(c, false);
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
```

```
                                result.getErrors()[0].getMessage());
    }
}
```

Create Test Data For Apex Tests:

1) RandomContactFactory.apxc

```
public class RandomContactFactory
{
        public static List<Contact> generateRandomContacts(integer numofContacts,string
LastNameGen)
  {
        List<Contact> con= new List<Contact>();
        for(Integer i=0;i<numofContacts;i++)
        {
        LastNameGen='Test'+ i;
        Contact a=new Contact(FirstName=LastNameGen,LastName=LastNameGen);
       con.add(a);
    }
    return con;
  }
}
```

ASYNCHRONOUS APEX MODULE:

Use Future Methods:

1)AccountProcessor.apxc:

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN
: accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
```

```
            account.Number_of_Contacts__c = [Select count() from Contact
Where AccountId =: account.Id];
             System.debug('No Of Contacts = ' +
account.Number_of_Contacts__c);
             updatedAccounts.add(account);
         }
         update updatedAccounts;
     }
}
```

2)AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }

}
```

USE BATCH APEX:

1) LeadProcessor.apxc

```
global class LeadProcessor implements Database.Batchable <SObject> {
    global Database.QueryLocator start(Database.BatchableContext bc){
        String Query='Select id,LeadSource from Lead';
        return Database.getQueryLocator(Query);
            }
    global void execute(Database.BatchableContext bc, List<Lead> scope){
        for(Lead l: scope){
            l.LeadSource='DreamForce';
        }
        update scope;
    }
    global void finish(Database.BatchableContext bc){
        Id job= bc.getJobId();
        System.debug(job);
    }
}
```

2) LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    @istest
    static void tetslead(){
        List<Lead> l= new List<Lead>();
        lead l1= new Lead();
        l1.LastName='surya';
        l1.Company='Company';
        l1.Status='Closed-Converted';
        l1.LeadSource='Dreamforce';
        l.add(l1);
        insert l;

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

Control Processes with Queueable Apex:

1) AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select
id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState =
:state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
                 Contact cont = c.clone(false,false,false,false);
                 cont.AccountId =  acc.id;
                 lstContact.add( cont );
         }

         if(lstContact.size() >0 )
         {
             insert lstContact;
         }

    }

}
```

2) AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
```

```
{
     @isTest static void TestList()
     {
         List<Account> Teste = new List <Account>();
         for(Integer i=0;i<50;i++)
         {
             Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
         }
         for(Integer j=0;j<50;j++)
         {
             Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
         }
         insert Teste;

         Contact co = new Contact();
         co.FirstName='demo';
         co.LastName ='demo';
         insert co;
         String state = 'CA';

          AddPrimaryContact apc = new AddPrimaryContact(co, state);
          Test.startTest();
            System.enqueueJob(apc);
          Test.stopTest();
      }
 }
```

Schedule Jobs Using the Apex Scheduler:

1) DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where
LeadSource = null limit 200];
        list<lead> led = new list<lead>();
        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
                led.add(l);
            }
            update led;
```

```
        }
    }
}
```

2) DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest{

    static testMethod void testMethod1()
    {
                Test.startTest();

        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            Lead led = new Lead();
            led.FirstName ='FirstName';
            led.LastName ='LastName'+i;
            led.Company ='demo'+i;
            lstLead.add(led);
        }

        insert lstLead;

        DailyLeadProcessor ab = new DailyLeadProcessor();
         String jobId = System.schedule('jobName','0 5 * * * ? ' ,ab) ;


        Test.stopTest();
    }
}
```

ASYNCHRONOUS APEX MODULE:

Apex REST Callouts :

1)AnimalLocator.apxc

```
public class AnimalLocator
{
```

```
  public static String getAnimalNameById(Integer id)
   {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-

        request.setMethod('GET');
        HttpResponse response = http.send(request);
          String strResp = '';
           system.debug('******response '+response.getStatusCode());
           system.debug('******response '+response.getBody());
        // If the request is successful, parse the JSON response.
        if (response.getStatusCode() == 200)
        {
            // Deserializes the JSON string into collections of primitive
data types.
           Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
           Map<string,object> animals = (map<string,object>)
results.get('animal');
            System.debug('Received the following animals:' + animals );
            strResp = string.valueof(animals.get('name'));
            System.debug('strResp >>>>>>' + strResp );
        }
        return strResp ;
   }

}
```

2) 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);
    }
}
```

Apex SOAP Callouts :

1)ParkService.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-

        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://calculator.services/', 'calculatorServices',
'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;
        }
    }
}
```

2)ParkLocator.apxc

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

3) ParkLocatorTest.apxc

```
@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
        List<String> result = new List<String>();
        List<String> expectedvalue = new
List<String>{'Park1','Park2','Park3'};

        result = ParkLocator.country('India');
        // Verify that a fake result is returned
        System.assertEquals(expectedvalue, result);
    }
}
```

4) ParkServiceMock.apxc

```
@isTest
```

```
global class ParkServiceMock implements WebServiceMock{
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
        // start - specify the response you want to send
        ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();

        List<String> myStrings = new List<String>
{'Park1','Park2','Park3'};

        response_x.return_x = myStrings;
        // end
        response.put('response_x', response_x);
   }
}
```

Apex Web Services :

1) AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        String accountId =
request.requestURI.substringBetween('Accounts/','/contacts');
        system.debug(accountId);
        Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM Contacts)
FROM Account WHERE Id = :accountId LIMIT 1];
        return objAccount;
    }
}
```

2) AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest{
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'test Account');
        insert objAccount;
        Contact objContact = new Contact(LastName = 'test Contact',
                                         AccountId = objAccount.Id);
        insert objContact;
        Id recordId = objAccount.Id;
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://sandeepidentity-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'
            + recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount!= null);
        System.assertEquals('test Account', thisAccount.Name);
    }
}
```

CODES FOR APEX SUPERBADGE IN SALESFORCE DEVELOPER SPECIALIST CHALLENGE:

STEP 2:Automate record creation:

1)MaintenanceRequestHelper.apxc

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


        For (Case c : updWorkOrders){
```

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



                }
            }
        }

        if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                          FROM Case WHERE Id IN
:validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
        }

            for(Case cc : closedCasesM.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                Status = 'New',
                    Subject = '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));
                } else {
                    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                }
```

```
                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;
        }
    }
}
```

2) MaitenanceRequest.apxt

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

STEP 3: Synchronize Salesforce data with an external system:

1)WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements
Queueable {
 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);

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }
            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the

            }
        }
```

```
    }

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

In Anonymous window execute this method:
System.enqueueJob(new WarehouseCalloutService());

STEP 4: Schedule synchronization using Apex code:

1)WarehouseSyncShedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

STEP 5: TEST AUTOMATION LOGIC:

1)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;
```

```
91
92          Equipment_Maintenance_Item__c workP =
    createWorkPart(equipmentId, emptyReq.Id);
93          insert workP;
94
95          test.startTest();
96          emptyReq.Status = WORKING;
97          update emptyReq;
98          test.stopTest();
99
100         list<case> allRequest = [select id
101                                  from case];
102
103         Equipment_Maintenance_Item__c workPart = [select id
104                                                   from
    Equipment_Maintenance_Item__c
105                                                   where
    Maintenance_Request__c = :emptyReq.Id];
106
107         system.assert(workPart != null);
108         system.assert(allRequest.size() == 1);
109     }
110
111     @istest
112     private static void testMaintenanceRequestBulk(){
113         list<Vehicle__C> vehicleList = new list<Vehicle__C>();
114         list<Product2> equipmentList = new list<Product2>();
115         list<Equipment_Maintenance_Item__c> workPartList = new
    list<Equipment_Maintenance_Item__c>();
116         list<case> requestList = new list<case>();
117         list<id> oldRequestIds = new list<id>();
118
119         for(integer i = 0; i < 300; i++){
120            vehicleList.add(createVehicle());
121             equipmentList.add(createEq());
122         }
123         insert vehicleList;
124         insert equipmentList;
125
126         for(integer i = 0; i < 300; i++){
127
    requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
    equipmentList.get(i).id));
128         }
```

```
        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);
    }
```

2)MaintenanceRequestHelper.apxc

```
1  public with sharing class MaintenanceRequestHelper {
2      public static void updateworkOrders(List<Case> updWorkOrders,
   Map<Id,Case> nonUpdCaseMap) {
3          Set<Id> validIds = new Set<Id>();
4
5
6          For (Case c : updWorkOrders){
7              if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
   c.Status == 'Closed'){
8                  if (c.Type == 'Repair' || c.Type == 'Routine
```

```
                    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()

                );
```

```
39
40                  If (maintenanceCycles.containskey(cc.Id)){
41                      nc.Date_Due__c =
   Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
42                  }
43
44                  newCases.add(nc);
45              }
46
47              insert newCases;
48
49              List<Equipment_Maintenance_Item__c> clonedWPs = new
   List<Equipment_Maintenance_Item__c>();
50              for (Case nc : newCases){
51                  for (Equipment_Maintenance_Item__c wp :
   closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
52                      Equipment_Maintenance_Item__c wpClone =
   wp.clone();
53                      wpClone.Maintenance_Request__c = nc.Id;
54                      ClonedWPs.add(wpClone);
55                  }
56              }
57              insert ClonedWPs;
58          }
59      }
60 }
```

2)MaitenanceRequest.apxt

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

STEP 6)Test callout logic:-

1)WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-

public static void runWarehouseEquipmentSync(){

    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);


    List<Product2> warehouseEq = new List<Product2>();

    if (response.getStatusCode() == 200){
      List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
      System.debug(response.getBody());

      for (Object eq : jsonResponse){
        Map<String,Object> mapJson = (Map<String,Object>)eq;
        Product2 myEq = new Product2();
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
        myEq.Cost__c = (Decimal) mapJson.get('lifespan');
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
        warehouseEq.add(myEq);
      }

      if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse one');
        System.debug(warehouseEq);
      }

    }
  }
}
```

2)WarehouseCalloutServiceMock.apxc

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

        System.assertEquals('https://th-superbadge-
));
        System.assertEquals('GET', request.getMethod());

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":fal



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

3)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]);
    }
}
```

STEP 7)Test scheduling logic:

1)WarehouseSyncSchedule.apxc :

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

2) 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();

        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime >
today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```