APEX TRIGGERS:

➤ Get Started with Apex Triggers

1.AccountAddressTrigger.apxc:

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

➤ BULK APEX TRIGGERS:

1.ClosedOpportunityTrigger.apxc:

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

APEX TESTING:

➤ 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); //create a date 30 days away from date1
		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:`

```
@isTest
private class TestVerifyDate {
    @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);
    }
        @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);
    }
        @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);
    }
}
```

➤ Test Apex Triggers:

1. **RestrictContactByName.apxc:**

```
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');
                                                  }


                                             }
}
```

➤ Create Test Data for Apex Tests:

1.**RandomContactFactory.apxc:**

```
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');
           }
      }
}
```

Asynchronous Apex:

➤ Use Future Methods:

1.**AccountProcessor.apxc:**

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accountsToUpdate = new List<Account>();
      List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where
Id in :accountIds];
      For(Account acc:accounts){
         List<Contact> contactList = acc.Contacts;
         acc.Number_Of_Contacts__c = contactList.size();
         accountsToUpdate.add(acc);
      }
      update accountsToUpdate;
   }
}
```

2.AccountProcessorTest.apxc:

```
@isTest
private class AccountProcessorTest {
                    @isTest
    private static void testNoOfContacts(){
        Account newAccount = new Account(Name ='Test Account');
        insert newAccount;
        Contact newContact1 = new
Contact(FirstName='John',LastName='Doe',AccountId = newAccount.Id);
        insert newContact1;
        Contact newContact2 = new
Contact(FirstName='Jane',LastName='Doe',AccountId = newAccount.Id);
        insert newContact2;
        List<Id>accountIds = new List<Id>();
        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
}
```

➤ Use Batch Apex:

1.LeadProcessor.apxc:

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

    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, LeadSource FROM
Lead');
    }

    global void execute(Database.BatchableContext bc, List<Lead>
scope){
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {
                lead.LeadSource = 'Dreamforce';
                recordsProcessed = recordsProcessed + 1;
        }
        update leads;
    }

    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed.
Shazam!');
    }
}
```

2.LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }
    static testmethod void test() {
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();
        System.assertEquals(200, [select count() from lead where
LeadSource = 'Dreamforce']);
    }
}
```

➤ Control Processes with Queueable Apex:

1.AddPrimaryContact.apxc:

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE
BillingState = :state LIMIT 200];
        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;
            lstOfConts.add(conInst);
        }
        INSERT lstOfConts;
    }
}
```

2.AddPrimaryContactTest.apxc:

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState =
'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState =
'CA'));
        }
        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');
        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();
        System.assertEquals(50, [select count() from Contact]);
    }
}
```

➤ Schedule Jobs Using the Apex Scheduler:

1.DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {
        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where
LeadSource = '' LIMIT 200];
        for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}
```

2.DailyLeadProcessorTest.apxc:

```
@isTest
private class DailyLeadProcessorTest {
    @isTest
    public static void testDailyLeadProcessor(){
        List<Lead> leads = new List<Lead>();
        for (Integer x = 0; x < 200; x++) {
            leads.add(new Lead(lastname='lead number ' + x,
company='company number ' + x));
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 *
* ?', new DailyLeadProcessor());
        Test.stopTest();
        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where
LeadSource = 'Dreamforce' LIMIT 200];
        System.assertEquals(200, listResult.size());
    }
}
```

Apex Integration Services:

➤ 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-
callout.herokuapp.com/animals/'+id);
         request.setMethod('GET');
         HttpResponse response = http.send(request);
           String strResp = '';
            system.debug('******response '+response.getStatusCode());
            system.debug('******response '+response.getBody());
         if (response.getStatusCode() == 200)
         {
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            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.AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
       response.setHeader('Content-Type', 'application/json');
       response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
       response.setStatusCode(200);
       return response;
```

```
  }
}
```

3.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.ParkLocator.apxc:

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

2. ParkService .apxc:

```
//Generated by wsdl2apex
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;
        }
    }
}
```

3.ParkSerciceMock.apxc

```
@isTest
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
         ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
         List<String> lstOfDummyParks = new List<String>
{'Park1','Park2','Park3'};
         response_x.return_x = lstOfDummyParks;
         response.put('response_x', response_x);
     }
}
```

4.ParkLocatorTest.apxc:

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

➤Apex Web Services:

1.AccountManager.apxc:

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

2.AccountManagerTest.apxc:

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

Apex Specialist SuperBadge

➤ AUTOMATE RECORD CREATION:

1.MaintenanceRequest.apxt:

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

2.Maintenance RequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
  public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }
      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                             FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
         AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];
      for (AggregateResult ar : results){
         maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
      }
         for(Case cc : closedCasesM.values()){
            Case nc = new Case (
               ParentId = cc.Id,
            Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()
            );
            If (maintenanceCycles.containskey(cc.Id)){
               nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }
            newCases.add(nc);
```

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

➤ SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

1.WarehouseCalloutService.apxc:

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    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);
            }
        }
    }
}
```

➤ SCHEDULE SYNCHRONIZATION USING APEX CODE:

1.WarehouseSyncSchedule.apxc:

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {
      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

➤ TEST AUTOMATION LOGIC:

1.Maintenance RequestHelperTest.apxc:

```
@istest
public with sharing class MaintenanceRequestHelperTest {
   private static final string STATUS_NEW = 'New';
   private static final string WORKING = 'Working';
   private static final string CLOSED = 'Closed';
   private static final string REPAIR = 'Repair';
   private static final string REQUEST_ORIGIN = 'Web';
   private static final string REQUEST_TYPE = 'Routine Maintenance';
   private static final string REQUEST_SUBJECT = 'Testing subject';
   PRIVATE STATIC Vehicle__c createVehicle(){
      Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
      return Vehicle;
   }
   PRIVATE STATIC Product2 createEq(){
      product2 equipment = new product2(name = 'SuperEquipment',
                                        lifespan_months__C = 10,
                                maintenance_cycle__C = 10,
                                replacement_part__c = true);
    return equipment;
  }
  PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cs = new case(Type=REPAIR,
                      Status=STATUS_NEW,
                      Origin=REQUEST_ORIGIN,
                      Subject=REQUEST_SUBJECT,
                      Equipment__c=equipmentId,
                      Vehicle__c=vehicleId);
    return cs;
  }
  PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
    Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                      Maintenance_Request__c = requestId);
    return wp;
  }
  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;
    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;
    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;
    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();
     Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
```

```
                  from case
                  where status =:STATUS_NEW];
    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c =:newReq.Id];
    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
  }
  @istest
  private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;
    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;
    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;
    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();
    list<case> allRequest = [select id
                             from case];
    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c = :emptyReq.Id];
    system.assert(workPart != null);
    system.assert(allRequest.size() == 1);
  }
  @istest
  private static void testMaintenanceRequestBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new
```

```
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();
        for(integer i = 0; i < 300; i++){
           vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;
        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;
        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;
        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();
        list<case> allRequests = [select id
                                 from case
                                 where status =: STATUS_NEW];
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                        from Equipment_Maintenance_Item__c
                                                        where Maintenance_Request__c in: oldRequestIds];
        system.assert(allRequests.size() == 300);
    }
}
```

2.MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
```

```
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }
      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                             FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
         AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];
      for (AggregateResult ar : results){
         maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
      }
         for(Case cc : closedCasesM.values()){
            Case nc = new Case (
               ParentId = cc.Id,
            Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()
            );
            If (maintenanceCycles.containskey(cc.Id)){
               nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }
            newCases.add(nc);
         }
        insert newCases;
        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
               Equipment_Maintenance_Item__c wpClone = wp.clone();
```

```
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);
            }
        }
        insert ClonedWPs;
    }
  }
}
```

3.MaintenanceRequest.apxt:

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

➤ TEST CALLOUT LOGIC:

1.WarehouseCalloutService.apxc:

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    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.WarehouseCalloutServiceTest.apxc:

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

3.WarehouseCalloutServiceMock.apxc:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":fal
se,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}
]');
        response.setStatusCode(200);
        return response;
    }
}
```

➤ 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 ');
    }
}
```