APEX MODULE CODES

MODULE: APEX TRIGGERS

1.Get started with apex triggers:(AccountAddressTrigger)

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

2.Bulk Apex Trigger:(ClosedOpportunityTrigger)

```
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;
    }
}
```

MODULE: APEX TESTING

1.Get started with Apex Unit Tests:(VerifyDate)

```
public class VerifyDate {

        //method to handle potential checks against two dates
```

```
	public static Date CheckDates(Date date1, Date date2) {
		//if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month
		if(DateWithin30Days(date1,date2)) {
			return date2;
		} else {
			return SetEndOfMonthDate(date1);
		}
	}

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

	//check that date2 is within (>=) 30 days of date1
	Date date30Days = date1.addDays(30); //create a date 30 days away from date1
		if( date2 >= date30Days ) { return false; }
		else { return true; }
	}

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

}
```

Test.apxc:(TestVerifyDate)

```
@isTest
private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
	DateD=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/202)
```

```
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2() {
                        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
                Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
                Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
                Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
            }
}
```

2.Test apex Triggers:(Create an Apex trigger RestrictContactByName on the contact objcet)

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

        //check contacts prior to insert or update for invalid data
```

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

Create separate test class TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {

   @isTest static void Test_insertupdateContact(){
      Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';

      Test.startTest();
      Database.SaveResult result = Database.insert(cnt, false);
      Test.stopTest();

      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
   }
}
```

3.Create Test Data For Apex Triggers:(RandomContactFactory)

```
public class RandomContactFactory {

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

```
        }
        return contacts;
    }
}
```

MODULE: ASYNCHRONOUS APEX

1.Use Future Methods(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];
        // process account records to do awesome stuff
        for(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

Test.apxc(AccountProcessorTest)

```
@IsTest
private class AccountProcessorTest{
  @IsTest
  private static void testCountContacts() {
      Account newAccount = new Account(Name = 'Test Account');
      insert newAccount;

      Contact newContact1 = new Contact(FirstName='John',
```

```
                            LastName='Doe',
                            AccountId=newAccount.Id);
    insert newContact1;

    Contact newContact2 = new Contact(FirstName='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();

  }
}
```

2.Use Batch Apex:(LeadProcessor.apxc)

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

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

   global void execute (Database.BatchableContext bc, List<Lead> L_list){
      List<Lead> L_list_new = new List<lead>();

      for(lead L:L_list){
         L.leadsource = 'Dreamforce';
         L_list_new.add(L);
         count +=1;
      }
      update L_list_new;
   }
```

```
    global void finish(Database.BatchableContext bc){
        system.debug('count =' + count);
    }
}
```

Test.apxc:(LeadProcessorTest)

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<Lead> L_list = new List<lead>();

        for(Integer i=0;i<200;i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;

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

3.Control Processes with Queuable Apex:(AddPrimaryContact.apxc)

```
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;

      }
        }

}
```

Test.apxc:(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' +i,BillingState='CA'));
      }
      for(Integer j=0;j<50;j++){
        testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));
            }
```

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

4.Schedule Jobs Using Apex Scheduler:(DailyLeadProcessor.apxc)

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<lead> leadstoupdate = new List<lead>();
        List<Lead> leads = [SELECT Id
            FROM Lead
            WHERE LeadSource = NULL Limit 200
            ];
                    for(Lead l:leads){
            l.LeadSource = 'Dreamforce';
            leadstoupdate.add(l);
        }
        update leadstoupdate;
                    }
}
```

Test.apxc:(DailyLeadProcessorTest)

```
@isTest
private class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
```

```
    static testmethod void testScheduleJob() {
        List<Lead> leads = new List<lead>();
        for (Integer i=0; i<200; i++) {
            Lead l = new Lead(
            FirstName = 'First ' +i,
            LastName = 'LastName',
            Company = 'The Inc'
            );
            leads.add(l);
        }
        insert leads;

        Test.startTest();
                        String jobId = System.schedule('ScheduleApexTest',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> checkleads = new List<Lead>();
            checkleads = [SELECT Id FROM Lead WHERE LeadSource = 'Dreamforce' and
Company = 'The Inc'];

        System.assertEquals(200, checkleads.size(), 'Leads were not created');
    }
}
```

MODULE:APEX INTEGRATION SERVICES

1.Apex REST Callouts:(AnimalLocator.apxc)

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer x) {
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal= new Map<String, Object>();
        HttpResponse res = http.send(req);
```

```
        if(res.getStatusCode() == 200) {
                                    Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
    animal = (Map<String, Object>) results.get('animal');
        }
return(String)animal.get('name');
  }
}
```

AnimalLocatorMock.apxc

```
@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", "fluffy bunny", "scary
bear", "chicken", "mighty moose"]}');
            response.setStatusCode(200);
    return response;
  }
}
```

AnimalLocatorTest.apxc

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

2.Apex SOAP Callouts:(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;
      }
   }
}
```

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();
        response_x.return_x = new List<String>{'Yosemite','Sequoia','Crater Lake'};
        response.put('response_x', response_x);
  }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {

   @isTest static void testCallout () {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String country = 'United States';
      List<String> expectedParks = new List<String>{'Yosemite','Sequoia','Crater Lake'};

      System.assertEquals(expectedParks,ParkLocator.country(country));

   }
}
```

3.Apex Web Services:(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');
           Account result = [SELECT ID,Name,(SELECT ID, FirstName, LastName FROM
Contacts)
                             FROM Account
                             WHERE Id = :accountId];
      return result;
   }
```

```
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

    @isTest
    static void testGetAccount() {
        Account a = new Account(Name = 'TestAccount');
        insert a;
        contact c = new Contact(AccountId=a.Id, FirstName='Test', LastName='Test');
        insert c;

        RestRequest request = new RestRequest();
                                                                request.requestUri
='https://yourInstance.salesforce.com/service/apexrest//Accounts/'+a.Id+'/contacts';
        request.httpMethod = 'Get';
        RestContext.request = request;

        Account myAcct = AccountManager.getAccount();
        //verify results
        System.assert(myAcct != null);
        System.assertEquals('TestAccount', myAcct.Name);

    }
}
```

APEX SPECIALIST SUPERBADGE CODES

CHALLENGE 1: AUTOMATED RECORD CREATION:
(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;
      }
   }
}
```

MaintenanceRequest.apxt

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

```
    }
}
```

CHALLENGE-2:SYNCHRONIZE SALESFORCE DATE WITH AN EXTERNAL SYSTEM: (WarehouseCalloutService.apxc)

```
public with sharing class WarehouseCalloutService {

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

    @future( callout = true )
    public static void runWarehouseEquipmentSync() {

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

        request.setEndpoint( WAREHOUSE_URL );
        request.setMethod( 'GET' );
        request.setHeader( 'Content-Type', 'text-xml' );
        response = http.send( request );

        List<WarehouseEquipment> warehouseEquipmentList = new
WarehouseEquipment().parse( response.getBody() );
        List<Product2> productsToUpsert = new List<Product2>();

        // Update Salesforce Records
        for ( WarehouseEquipment whrEquip : warehouseEquipmentList ) {
            Product2 newProduct = new Product2( Warehouse_SKU__c  = whrEquip.id );
            newProduct.Replacement_Part__c = true;
            newProduct.Cost__c = whrEquip.cost;
            newProduct.Current_Inventory__c = whrEquip.quantity;
            newProduct.Lifespan_Months__c = whrEquip.lifespan;
            newProduct.Maintenance_Cycle__c = whrEquip.maintenanceperiod;
            newProduct.Name = whrEquip.name;
            productsToUpsert.add( newProduct );
```

```
        }

        upsert productsToUpsert;
    }

    public class WarehouseEquipment {
        public String name;
        public Boolean replacement;
        public Integer quantity;
        public Integer maintenanceperiod;
        public Integer lifespan;
        public Integer cost;
        public String sku;
        public String id;

        public List<WarehouseEquipment> parse( String json ) {
            json.replace( '"id":', '"_id ":' );
            return ( List<WarehouseEquipment> ) System.JSON.deserialize( json,
List<WarehouseEquipment>.class );
        }
    }

}
```

CHALLENGE-3:SCHEDULABLE SYNCHRONIZATION USING APEX CODE:
(WarehouseSyncShedule.apxc)

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

CHALLENGE-4:TEST AUTOMATION LOGIC:
(MaintenanceRequestHelperTest.apxc)

```
@istest
```

```
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

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

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

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

```
                                                Maintenance_Request__c = requestId);
        return wp;
    }


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

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

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

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

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

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

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

        system.assert(workPart != null);
```

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

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

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

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

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

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

        list<case> allRequest = [select id
                                 from case];

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

        system.assert(workPart != null);
```

```
        system.assert(allRequest.size() == 1);
    }

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

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

        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

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

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

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

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

MaintenanceRequestHelper.apxt

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

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;
      }
   }
}
```

CHALLENGE-5:TEST CALLOUT LOGIC:

(WarehouseCalloutService.apxc)

```
public with sharing class WarehouseCalloutService {

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

   @future( callout = true )
   public static void runWarehouseEquipmentSync() {

      Http http = new Http();
      HttpRequest request = new HttpRequest();
      HTTPResponse response = new HTTPResponse();
```

```
        request.setEndpoint( WAREHOUSE_URL );
        request.setMethod( 'GET' );
        request.setHeader( 'Content-Type', 'text-xml' );
        response = http.send( request );

        List<WarehouseEquipment> warehouseEquipmentList = new
WarehouseEquipment().parse( response.getBody() );
        List<Product2> productsToUpsert = new List<Product2>();

        // Update Salesforce Records
        for ( WarehouseEquipment whrEquip : warehouseEquipmentList ) {
            Product2 newProduct = new Product2( Warehouse_SKU__c  = whrEquip.id );
            newProduct.Replacement_Part__c = true;
            newProduct.Cost__c = whrEquip.cost;
            newProduct.Current_Inventory__c = whrEquip.quantity;
            newProduct.Lifespan_Months__c = whrEquip.lifespan;
            newProduct.Maintenance_Cycle__c = whrEquip.maintenanceperiod;
            newProduct.Name = whrEquip.name;
            productsToUpsert.add( newProduct );
        }

        upsert productsToUpsert;
    }

    public class WarehouseEquipment {
        public String name;
        public Boolean replacement;
        public Integer quantity;
        public Integer maintenanceperiod;
        public Integer lifespan;
        public Integer cost;
        public String sku;
        public String id;

        public List<WarehouseEquipment> parse( String json ) {
            json.replace( '"id":', '"_id ":' );
            return ( List<WarehouseEquipment> ) System.JSON.deserialize( json,
```

```
List<WarehouseEquipment>.class );
        }
    }

}
```

WarehouseCalloutServiceTest.apxc

```
@isTest
private class WarehouseCalloutServiceTest {

    @isTest static  void warehouseServiceTest() {
        Test.startTest();
        Test.SetMock(HttpCallOutMock.class, new WarehouseCalloutServiceMock());

        WarehouseCalloutService.runWarehouseEquipmentSync();

        List<Product2> productsToUpsert = [SELECT Replacement_Part__c, Cost__c,
Current_Inventory__c, Lifespan_Months__c,
                                Maintenance_Cycle__c, Name FROM Product2];

        System.assert( true, productsToUpsert.size() == 22 );

        // Update Salesforce Records
        for ( Product2 equipmentUpserted : productsToUpsert ) {
            System.assert( true, equipmentUpserted.Replacement_Part__c );
            System.assert( true, equipmentUpserted.Cost__c != null );
            System.assert( true, equipmentUpserted.Current_Inventory__c != null );
            System.assert( true, equipmentUpserted.Lifespan_Months__c != null );
            System.assert( true, equipmentUpserted.Maintenance_Cycle__c != null );
            System.assert( true, equipmentUpserted.Name != null );
        }

        Test.stopTest();
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'text-xml');
        response.setBody(getJsonResponse());
        response.setStatusCode(200);
        return response;
    }

    public String getJsonResponse() {
        return
'[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator
1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,' +

'"sku":"100003"},{"_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"n
ame":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,' +

'"sku":"100004"},{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"n
ame":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22, ' +

'"sku":"100005"},{"_id":"55d66226726b611100aaf744","replacement":false,"quantity":5,"na
me":"Generator 2000 kw","maintenanceperiod":365,"lifespan":120,"cost":6000, '+

'"sku":"100006"},{"_id":"55d66226726b611100aaf745","replacement":true,"quantity":142,"n
ame":"Fuse 25A","maintenanceperiod":0,"lifespan":0,"cost":28,' +

'"sku":"100007"},{"_id":"55d66226726b611100aaf746","replacement":true,"quantity":122,"n
ame":"Fuse 13A","maintenanceperiod":0,"lifespan":0,"cost":10, '+

'"sku":"100008"},{"_id":"55d66226726b611100aaf747","replacement":true,"quantity":90,"na
me":"Ball Valve 10 cm","maintenanceperiod":0,"lifespan":0,"cost":50,'+

'"sku":"100009"},{"_id":"55d66226726b611100aaf748","replacement":false,"quantity":2,"na
me":"Converter","maintenanceperiod":180,"lifespan":120,"cost":3000, '+
```

"'sku":"100010"},{"_id":"55d66226726b611100aaf749","replacement":true,"quantity":75,"name":"Ball Valve 8 cm","maintenanceperiod":0,"lifespan":0,"cost":42,'+

"'sku":"100011"},{"_id":"55d66226726b611100aaf74a","replacement":true,"quantity":100,"name":"Breaker 25A","maintenanceperiod":0,"lifespan":0,"cost":30,'+

"'sku":"100012"},{"_id":"55d66226726b611100aaf74b","replacement":true,"quantity":150,"name":"Switch","maintenanceperiod":0,"lifespan":0,"cost":100, '+

"'sku":"100013"},{"_id":"55d66226726b611100aaf74c","replacement":true,"quantity":200,"name":"Ball Valve 5 cm","maintenanceperiod":0,"lifespan":0,"cost":30, '+

"'sku":"100014"},{"_id":"55d66226726b611100aaf74d","replacement":false,"quantity":8,"name":"UPS 3000 VA","maintenanceperiod":180,"lifespan":60,"cost":1600,'+

"'sku":"100015"},{"_id":"55d66226726b611100aaf74e","replacement":false,"quantity":10,"name":"UPS 1000 VA","maintenanceperiod":180,"lifespan":48,"cost":1000,'+

"'sku":"100016"},{"_id":"55d66226726b611100aaf74f","replacement":true,"quantity":180,"name":"Breaker 8A","maintenanceperiod":0,"lifespan":0,"cost":10,'+

"'sku":"100017"},{"_id":"55d66226726b611100aaf750","replacement":false,"quantity":2,"name":"Cooling Tower","maintenanceperiod":365,"lifespan":120,"cost":10000,'+

"'sku":"100018"},{"_id":"55d66226726b611100aaf751","replacement":true,"quantity":165,"name":"Motor","maintenanceperiod":0,"lifespan":0,"cost":150,'+

"'sku":"100019"},{"_id":"55d66226726b611100aaf752","replacement":true,"quantity":210,"name":"Breaker 13A","maintenanceperiod":0,"lifespan":0,"cost":20,'+

"'sku":"100020"},{"_id":"55d66226726b611100aaf753","replacement":true,"quantity":100,"name":"Radiator Pump","maintenanceperiod":0,"lifespan":0,"cost":500, '+

"'sku":"100021"},{"_id":"55d66226726b611100aaf754","replacement":true,"quantity":129,"name":"Breaker 20A","maintenanceperiod":0,"lifespan":0,"cost":25,'+

'"sku":"100022"},{"_id":"55d66226726b611100aaf73f","replacement":false,"quantity":10,"name":"UPS 2000 VA","maintenanceperiod":180,"lifespan":60,"cost":1350, '+

'"sku":"100001"},{"_id":"55d66226726b611100aaf740","replacement":true,"quantity":194,"name":"Fuse 8A","maintenanceperiod":0,"lifespan":0,"cost":5,"sku":"100002"}]';
    }
}

CHALLENGE-6:TEST SCHEDULING LOGIC:

(WarehouseSyncSchedule.apxc)

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

WarehouseSyncScheduleTest.apxc

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void testScheduler() {
        Test.SetMock(HttpCallOutMock.class, new WarehouseCalloutServiceMock());

        String CRON_EXP = '0 0 0 1 1/1 ? *';   // To be executed monthly at day one
        Integer runDate = 1;

        DateTime firstRunTime = System.now();
        DateTime nextDateTime;

        if(firstRunTime.day() < runDate) {
            nextDateTime = firstRunTime;
        } else {
            nextDateTime = firstRunTime.addMonths(1);
        }
```

```
        Datetime nextRunTime = Datetime.newInstance(nextDateTime.year(),
nextDateTime.month(), runDate);

        Test.startTest();
        WarehouseSyncSchedule warehouseSyncSchedule = new
WarehouseSyncSchedule();

        String jobId = System.schedule('Test Scheduler',
                                CRON_EXP,
                                warehouseSyncSchedule);

        Test.stopTest();

        // Get the information from the CronTrigger API object
        CronTrigger ct = [SELECT Id,  CronExpression,  TimesTriggered,  NextFireTime
FROM CronTrigger WHERE Id = :jobId];

        // Verify the expressions are the same
        System.assertEquals(CRON_EXP, ct.CronExpression);

        // Verify the job has not run
        System.assertEquals(0, ct.TimesTriggered);

        // Verify the next time the job will run
        System.assertEquals(String.valueOf(nextRunTime),
String.valueOf(ct.NextFireTime));

    }

}
```