Apex Triggers

1)Get Started with Apex Triggers

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

2)Bulk Apex Triggers

```
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

1)Get Started with Apex Unit Tests

```
@isTest
public class TestVerifyDate {

    //date within 30 days
    @isTest static void case1(){
        Date D1 = VerifyDate.CheckDates(date.parse('03-18-2022'),date.parse('03-22-2022'));
        // comparing the dates
        System.assertEquals(date.parse('03-22-2022'), D1);  // (expected, actual)
    }

    //date not within 30 days
    @isTest static void case2(){
        Date D2 = VerifyDate.CheckDates(date.parse('03-18-2022'),date.parse('06-22-2022'));
        // comparing the dates
        System.assertEquals(date.parse('06-22-2022'), D2);  // (expected, actual)
    }
}
```

2)Test Apex Triggers

```
@isTest
public class TestRestrictContactByName {

    @isTest
    public static void testContact(){
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        Database.SaveResult res = Database.insert(ct,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());
    }

}
```

3)Create Test Data for Apex Tests

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num,String lastName){
      List<Contact>  contactList = new List<Contact>();
        for (Integer i=1;i<=num ; i++){
            contact ct = new contact(FirstName = 'Test'+i, LastName = lastName );
            contactList.add(ct);
        }
        return contactList;
    }

}
```

Asynchronous Apex

1)Use Future Methods

```
public without sharing class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts =[SELECT Id, (SELECT Id FROM Contacts) FROM Account
WHERE Id IN :accountIds];
```

```
        for(Account acc: accounts){
            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }
        update accounts;
    }

}


@isTest
private class AccountProcessorTest {
    @isTest
    private static void countContactsTest(){

        List<Account> accounts= new List<Account>();
        for(Integer i=0; i<300 ; i++){
            accounts.add(new Account(Name ='Test Account'+i));
        }
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for(Account acc:accounts){
            contacts.add(new
Contact(FirstName=acc.Name,LastName='TestContact',AccountId=acc.Id));
            accountIds.add(acc.id);
        }
        insert contacts;

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();

    }

}
```

2)Use Batch Apex

```
public without sharing class LeadProcessor implements Database.Batchable<sObject>{

    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name FROM Lead]);

    }
    public void execute(Database.BatchableContext dbc , List<Lead> leads){
        for(Lead l : leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }

    public void finish(Database.BatchableContext dbc){
        System.debug('Done');
    }


}

@isTest
private class LeadProcessorTest {

    @isTest
    private static void testBatchClass(){

        List<Lead> leads = new List<Lead>();
        for(Integer i=0; i<200 ;i++){
            leads.add(new Lead(LastName ='Connak',Company ='Salesfrce'));
        }
       insert leads;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        List<Lead> updatedLeads =[SELECT Id FROM Lead WHERE LeadSource
='Dreamforce'];
        System.assertEquals(200, updatedLeads.size(),'ERROR: At least 1 lead record not updated
correctly');

    }



}
```

3)Control Processes with Queueable Apex

```
public without sharing class AddPrimaryContact implements Queueable {

    private Contact contact;
    private String state;

    public AddPrimaryContact (Contact inputContact , String inputState){
        this.contact = inputContact;
        this.state = inputState;
    }

    public void execute (QueueableContext context){
        //retrive 200 Account records
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state
LIMIT 200];

        //create empty list of contact records
        List<Contact> contacts = new List<Contact>();

        //Iterate through acc record
        for( Account acc : accounts){

            //copy con record , make thet copy a child of specific acc rec
            //& add to list of contacts
            Contact contactClone = contact.clone();
            contactClone.AccountId=acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }

}

@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass(){

        //load test data
        List<Account> accounts = new List<Account>();
        for(Integer i =0; i<500; i++){
            Account acc =new Account(Name  ='Tect account');
            if (i<250){
                acc.BillingState = 'NY';
            }else{
                acc.BillingState = 'CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact  = new Contact(FirstName='Simon',LastName='Connock');
        insert contact;

        //Perform the test
        Test.startTest();
        Id jobId =System.enqueueJob(new AddPrimaryContact(contact,'CA'));
        Test.stopTest();

        //check result
        List<Contact> contacts =[SELECT Id FROM Contact WHERE
Contact.Account.BillingState ='CA'];
        System.assertEquals(200,contacts.size(),'ERROR: Incorrect no of contact records found');
    }

}
```

4) Schedule Jobs Using the Apex Scheduler

```
public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext  ctx){
        //Get 200 Lead records & modify the leadsource field
        List<Lead> leads =[SELECT Id,LeadSource FROM Lead WHERE LeadSource = null
LIMIT 200];
        for(Lead l : leads){
            l.LeadSource = 'DreamForce';
        }

        //update modified rec
        update leads;
    }

}

@isTest
public class DailyLeadProcessorTest {

    private static String CRON_EXP ='0 0 0 ? * * *';//midnight every day

    @isTest
    private static void testSchedulableClass(){

        //Load test data
        List<Lead> leads = new List<Lead>();
        for(Integer i=0;i<500;i++){
            if(i<250){
                leads.add(new Lead(LastName='Connock',Company='Salesforce'));
            }else{
                  leads.add(new
Lead(LastName='Connock',Company='Salesforce',LeadSource='Other'));
            }
        }
        insert leads;

        //perform test
        Test.startTest();
        String jobId = System.schedule('Process Leads',CRON_EXP,new DailyLeadProcessor());
        Test.stopTest();

        //check result
        List<Lead> updatedLeads =[SELECT Id,LeadSource FROM Lead WHERE LeadSource
='Dreamforce'];
        System.assertEquals(200,updatedLeads.size(),'ERROR: At least 1 record not updated
correctly');

        //check the sheduled time
        List<CronTrigger> cts=[SELECT Id,TimesTriggered, NextFireTime FROM CronTrigger
WHERE Id = :jobId];
        System.debug('Next fire time'+ cts[0].NextFireTime);
    }

}
```

Apex Integration Services

1)Apex REST Callouts

```
public class AnimalLocator {

    public static String getAnimalNameById (Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        //if the request is successful, parse the 350N response.
        Map<String, Object> result =(Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
        Map<String, Object> animal =(Map<String, Object>)result.get('animal');
        System.debug('name: '+String.valueOf(animal.get('name')));
        return String.valueOf(animal.get('name'));


    }

}

@isTest
private class AnimalLocatorTest {

    @isTest
    static void animalLocatorTest1(){
        Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }

}

@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('ContentType', 'application/json');
        response.setBody('{"animal": {"id":1, "name":"moose",
"eats":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

2)Apex SOAP Callouts

```
//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/', 'ParksServices'};
        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;
        }
    }
}

public class ParkLocator {

    public static List<String> country(String country){
        ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
        return prkSvc.byCountry(country);
    }

}

@isTest
private class ParkLocatorTest {

    @isTest
    static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        System.assertEquals(new List<String>{'Yosemita','Sequioa','Crater
Lake'},ParkLocator.country(country));
    }

}

@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>{'Yosemita','Sequioa','Crater Lake'};
        // end
        response.put('response_x', response_x);
   }
```

```
}
```

3)Apex Web Services

```
@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,Name FROM Contacts) FROM Account
WHERE Id = :accountId];
        return result;
    }

}

@isTest
private class AccountManagerTest {

    @isTest  static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();

        RestRequest request = new RestRequest();
        request.requestURI
='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        // Call the method to test
        Account thisAcc = AccountManager.getAccount();

        // Verify results
        System.assert(thisAcc != null);
        System.assertEquals('Test record', thisAcc.Name);
    }

    //Helper class
    static Id createTestRecord(){
        //creating record
        Account accountTest = new Account(
            Name='Test record');
        insert accountTest;

        Contact contactTest = new Contact(
            FirstName='John',
            LastName='Doe',
            AccountId=accountTest.Id
        );
        insert contactTest;

        return accountTest.Id;

    }
}
```

APEX SPECIALIST SUPERBADGE

2)Automate record creation

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

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

        //When an existing maintenance request of type Repair or Routine Maintenance is closed,
        //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                        (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                        FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

        //calculate the maintenance request due dates by using the maintenance cycle defined on
the related equipment records.
        AggregateResult[] results = [SELECT Maintenance_Request__c,
                                    MIN(Equipment__r.Maintenance_Cycle__c)cycle
                                    FROM Equipment_Maintenance_Item__c
                                    WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.values()){
          Case nc = new Case (
            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 multiple pieces of equipment are used in the maintenance request,
          //define the due date by applying the shortest maintenance cycle to today’s date.
          //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> clonedList = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c item = clonedListItem.clone();
                    item.Maintenance_Request__c = nc.Id;
                    clonedList.add(item);
                }
            }
            insert clonedList;
        }
    }
}
```

3)Synchronize Salesforce data with an external system

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

    //class that makes a REST callout to an external warehouse system
to get a list of equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object jR : jsonResponse){
```

```
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            //cost
            product2.Cost__c = (Integer) mapJson.get('cost');
            //current inventory
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

          if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
          }
        }
    }

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

}
```

4) Schedule synchronization

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

5)Test automation logic

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



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

      //When an existing maintenance request of type Repair or Routine Maintenance is closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                          (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                          FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

         //calculate the maintenance request due dates by using the maintenance cycle defined on
the related equipment records.
         AggregateResult[] results = [SELECT Maintenance_Request__c,
                              MIN(Equipment__r.Maintenance_Cycle__c)cycle
                              FROM Equipment_Maintenance_Item__c
                              WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

         List<Case> newCases = new List<Case>();
         for(Case cc : closedCases.values()){
            Case nc = new Case (
               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 multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today’s date.
            //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> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
      }
   }
}

@isTest
public with sharing class MaintenanceRequestHelperTest {

   // createVehicle
   private static Vehicle__c createVehicle(){
      Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
      return vehicle;
   }
```

```
    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }

    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }


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

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

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
```

```
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
    update createdCase;
    test.stopTest();

      Case newCase = [Select id,
                  subject,
                  type,
                  Equipment__c,
                  Date_Reported__c,
                  Vehicle__c,
                  Date_Due__c
                 from case
                 where status ='New'];

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

    system.assert(newCase != null);
    system.assert(newCase.Subject != null);
    system.assertEquals(newCase.Type, 'Routine Maintenance');
    SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}

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

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

      case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;
```

```
        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;

          test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
          system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){
            equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.ge
t(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;
        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

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



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

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

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

6)Test callout logic

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

    //Write a class that makes a REST callout to an external warehouse system to get a list of
equipment that needs to be updated.
    //The callout’s JSON response returns the equipment records that you upsert in Salesforce.


     @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

                //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                    //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');

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

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

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

}
```

test

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
            @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}
```

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

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,
"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d6622672
6b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611
100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

7)test scheduling logic

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

@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```

[redacted]