My TrailheadURL: https://trailblazer.me/id/saikiranchebolu

APEX TRIGGERS

1. Get Started with Apex Triggers

CHALLENGE : Create an Apex trigger

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before
insert,before update) {
    for (account acct:trigger.new)
    {if(acct.Match_Billing_Address__c == true)
       {acct.shippingPostalCode = acct.billingPostalCode;}
    }
}
```

2. Bulk Apex Triggers

CHALLENGE : Create a Bulk Apex trigger

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after
insert, after update) {
   List<Task> taskList = new List<Task>();
    for (Opportunity o :[SELECT Id,StageName FROM
Opportunity WHERE StageName ='Closed Won' AND Id IN
:Trigger.New])
    {
        if(o.StageName == 'Closed Won')
        {
            taskList.add(new task (Subject ='Follow Up Test
Task' , WhatId=o.Id));
        }
 }
    if(taskList.size() > 0){
         insert taskList;
   }
}
```

APEX TESTING

1. Get Started with Apex Unit Tests

CHALLENGE : Create a Unit Test for a Simple Apex Class

VerifyDate.apxc

```
public class VerifyDate {

    public static Date CheckDates(Date date1, Date date2) {

         if(DateWithin30Days(date1,date2)) {

             return date2;

         } else {

             return SetEndOfMonthDate(date1);

        }

    }

 private static Boolean DateWithin30Days(Date date1, Date
date2) {

         if( date2 < date1) { return false; }

         Date date30Days = date1.addDays(30);

if( date2 >= date30Days ) { return false; }

         else { return true; }

    }

private static Date SetEndOfMonthDate(Date date1) {

         Integer totalDays = Date.daysInMonth(date1.year(),
date1.month());
```

```
         Date lastDay = Date.newInstance(date1.year(),
date1.month(), totalDays);

         return lastDay;

    }

}
```

TestVerifyDate.apxc

```
@isTest

public class TestVerifyDate {

    @isTest static void test1(){

        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('
01/03/2020'));

     System.assertEquals(Date.parse('01/03/2020'),d);

    }

@isTest static void test2(){

        Date d =
VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('
02/03/2020'));

        System.assertEquals(Date.parse('01/31/2020'),d);

    }

}
```

2. Test Apex Triggers

CHALLENGE : Create a Unit Test for a Simple Apex Trigger

RestrictContactByName.apxt

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

    For (Contact c : Trigger.New) {

         if(c.LastName == 'INVALIDNAME') {    //invalidname
is invalid

             c.AddError('The Last Name "'+c.LastName+'" is
not allowed for DML');

         }

    }
```

TestRestrictContactByName.apxc

```
@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

CHALLENGE : Create a Contact Test Factory

RandomContactFactory.apxc

```
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

2. Use Future Methods

CHALLENGE : Create an Apex class that uses the @future annotation to update Account records.

AccountProcessor.apxc

```
public class AccountProcessor {


@future

public static void countContacts(List<Id>accountIds){


List<Account> accList = [SelectId,Number_Of_Contacts__c,
(Select Id from Contacts) from Account where Id in
:accountIds];


for(Account acc : accList){


acc.Number_Of_Contacts__c = acc.Contacts.size();

update accList;

              }

     }

}
```

AccountProcessorTest.apxc

```
@isTest

public class AccountProcessorTest {

public static testmethod void testAccountProcessor(){

        Account a = new Account();
        a.Name = 'Test Account';
        insert a;

        Contact con = new Contact();
        con.FirstName = 'Sai';
        con.LastName = 'Programming';
        con.AccountId = a.Id;

        insert con;

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

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

Account acc = [Select Number_of_Contacts__c from Account
where Id =: a.Id];

System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts_
_c),1);
    }
}
```

3. Use Batch Apex

CHALLENGE : Create an Apex class that uses Batch Apex to update Lead records.

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!');
```

```
    }
}
```

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']);
    }
}
```

4. Control Processes with Queueable Apex

CHALLENGE : Create a Queueable Apex class that inserts Contacts for Accounts.

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;
    }
}
```

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]);
```

```
    }
}
```

5. <u>Schedule Jobs Using the Apex Scheduler</u>

CHALLENGE : Create an Apex class that uses Scheduled Apex to update Lead records.

DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];


        if(leads.size() > 0){

            List<Lead> newLeads = new List<Lead>();

for(Lead lead : leads){

                lead.LeadSource = 'DreamForce';

                newLeads.add(lead);

            }

            update newLeads;

        }

    }

}
```

DailyLeadProcessorTest.apxc

```
@isTest

private class DailyLeadProcessorTest{

public static String CRON_EXP = '0 0 0 2 6 ? 2022';

    static testmethod void testScheduledJob(){

        List<Lead> leads = new List<Lead>();

        for(Integer i = 0; i < 200; i++){

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource =
'', Company = 'Test Company ' + i, Status = 'Open - Not
Contacted');

            leads.add(lead);

        }

        insert leads;

        Test.startTest();

String jobId = System.schedule('Update LeadSource to
DreamForce', CRON_EXP, new DailyLeadProcessor());

        Test.stopTest();

    }

}
```

APEX INTEGRATION SERVICES

2. Apex REST Callouts

CHALLENGE : Create an Apex class that calls a REST endpoint and write a test class.

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 ;

   }

}
```

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);

    }

}
```

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. <u>Apex SOAP Callouts</u>

CHALLENGE : Generate an Apex class using WSDL2Apex and write a test class.

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','fals
e'};

        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;
        }
    }
}
```

ParkLocator.apxc

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

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    testMethod static void testCallout(){
         Test.setMock(WebServiceMock.class, new
ParkServiceMock());

         String country = 'United States';

         String[] result = ParkLocator.country(country);

        System.assertEquals(new List<String>{'Garner State
Park', 'Fowler Park', 'Hoosier National Forest Park'},
result);

    }

}
```

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) {

ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();

response_x.return_x = new List<String>{'Garner State Park',
'Fowler Park', 'Hoosier National Forest Park'};

                response.put('response_x',response_x);

        }

}
```

4. Apex Web Services

CHALLENGE : Create an Apex REST service that returns an account and its contacts.

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;

    }
}
```

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;
    }
}
```

VISUALFORCE BASICS

2. Create & Edit Visualforce Pages

CHALLENGE : Create a simple Visualforce page that displays an image

```
<apex:page showHeader="false" title="DisplayImage" sidebar="false">
    <apex:form>
        <table>
            <tr>
                <td width="1000px" height="600px" align="center">
                    <apex:image url="https://developer.salesforce.com/files/salesforce-developer-network-logo.png"/>
                </td>
            </tr>
        </table>
    </apex:form>
</apex:page>
```

3. Use Simple Variables and Formulas

CHALLENGE : Create a Visualforce page that shows user information

```
<apex:page >
    <apex:pageBlockSection columns="1">
        {! $User.FirstName} {! $User.LastName} {!
$User.Username})
    </apex:pageBlockSection>
</apex:page>
```

4. Use Standard Controllers

CHALLENGE : Create a Visualforce page that shows a basic Contact record

```
<apex:page standardController="Contact">
    <apex:pageBlock  title="Account Summary">
        <apex:pageBlockSection>
            First Name: {! Contact.FirstName} <br/>
            Last Name: {! Contact.LastName} <br/>
            Owner's Email: {! Contact.Owner.Email} <br/>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

5. Display Records, Fields, and Tables

CHALLENGE : Create a Visualforce page that displays a variety of output fields

```
<apex:page standardController="Opportunity">
    <apex:pageBlock title="Opportunity Page">
        <apex:pageBlockSection>
            <apex:outputField value="{!
Opportunity.Name}"/>
```

```
            <apex:outputField value="{!
Opportunity.Amount}"/>
            <apex:outputField value="{!
Opportunity.CloseDate}"/>
            <apex:outputField value="{!
Opportunity.Account.Name}"/>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

6. Input Data Using Forms

CHALLENGE : Create a Visualforce form which inserts a basic Contact record

```
<apex:page standardController="Contact">
    <apex:form>
        <apex:pageBlock   title="Add contacts">
            <apex:pageBlockSection columns="1">
                <apex:inputField value="{!
Contact.FirstName}"/>
                <apex:inputField value="{!
Contact.LastName}"/>
                <apex:inputField value="{!
Contact.email}"/>
            </apex:pageBlockSection>

            <apex:pageBlockButtons>
                <apex:commandButton action="{! save}"
```

```
value="Save"/>
            </apex:pageBlockButtons>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

7. Use Standard List Controllers

CHALLENGE : Create a Visualforce page that shows a list of Accounts linked to their record pages

```
<apex:page standardController="Account"
recordSetVar="Accounts">
    <apex:pageBlock>
        <apex:repeat var="a" value="{!Accounts}"
rendered="true" id="account_list">
            <li>
                <apex:outputLink value="/{!a.ID}">
                    <apex:outputText value="{!a.Name}"/>
                </apex:outputLink>
            </li>
        </apex:repeat>
    </apex:pageBlock>
</apex:page>
```

8. Use Static Resources

CHALLENGE : Use a static resource to display an image on a Visualforce Page

```
<apex:page >

    <apex:image url="{!URLFOR($Resource.vfimagetest,
'cats/kitten1.jpg')}" />

</apex:page>
```

9. Create & Use Custom Controllers

CHALLENGE : Create a Visualforce page that displays new cases

NewCaseListController

```
public class NewCaseListController {


    public List<Case> getNewCases(){
            List<Case>  cases = [select id, CaseNumber from
Case where status='New'];
        return cases;
    }
}
```

NewCaseList

```
<apex:page controller="NewCaseListController" >
    <apex:pageBlock title="New Cases List" id="cases_list">
<li>
        <apex:repeat  value="{!NewCases}" var="Case"
```

```
rendered="true" >
        <p><apex:outputLink
value="/{!Case.ID}">{!Case.CaseNumber}</apex:outputLink></
p>
        </apex:repeat>
    </li>

    </apex:pageBlock>
</apex:page>
```

Quick Start: Visualforce

2. Add a Standard Controller to the Page

```
<apex:page standardController="Contact">
    <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width,
initial-scale=1" />
      <title>Quick Start: Visualforce</title>
      <!-- Import the Design System style sheet -->
      <apex:slds />
    </head>
    <body>
              <apex:form>
      <apex:pageBlock title="New Contact">
        <!--Buttons -->
         <apex:pageBlockButtons>
            <apex:commandButton action="{!save}"
value="Save"/>
         </apex:pageBlockButtons>
         <!--Input form -->
         <apex:pageBlockSection columns="1">
         <apex:inputField value="{!Contact.Firstname}"/>
         <apex:inputField value="{!Contact.Lastname}"/>
         <apex:inputField value="{!Contact.Email}"/>
        </apex:pageBlockSection>
      </apex:pageBlock>
      </apex:form>
    </body>
</apex:page>
```

APEX SPECIALIST

CHALLENGE 2 : Automate 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));
                 } else {
                     nc.Date_Due__c =
Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                 }

                 newCases.add(nc);
             }

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

MaintenanceRequest.apxt :-

```
 trigger MaintenanceRequest on Case (before update, after
update) {

    if(Trigger.isUpdate && Trigger.isAfter){


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

    }

}
```

CHALLENGE 3 : Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :-

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

     @future(callout=true)
     public static void runWarehouseEquipmentSync(){
         Http http = new Http();
         HttpRequest request = new HttpRequest();

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

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

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

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

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

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

 }
```

CHALLENGE 4 : Schedule synchronization

WarehouseSyncShedule.apxc :-

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

CHALLENGE 5 : 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.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 6 : Test callout logic

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService
implements Queueable {
    private static final String WAREHOUSE_URL =
'https://th-superbadge-apex.herokuapp.com/equipment';
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        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());


            for (Object jR : jsonResponse){
                Map<String,Object> mapJson =
(Map<String,Object>)jR;
                Product2 product2 = new Product2();
                product2.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
                product2.Cost__c = (Integer)
mapJson.get('cost');
                product2.Current_Inventory__c = (Double)
mapJson.get('quantity');
                product2.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
                product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
                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');
    }
}
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request)
{
       HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","repla
cement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku
":"100003"},{"_id":"55d66226726b611100aaf742","replacement"
:true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"1
00004"},{"_id":"55d66226726b611100aaf743","replacement":tru
e,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"10
0005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutServiceTest.apxc :-

```
@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);
```

```
    }
}
```

CHALLENGE 7 : Test scheduling logic

WarehouseSyncSchedule.apxc :-

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

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 ');
    }
}
```