Nikitha M.A.

PRACTICE MODULES AND PROJECT PROGRAMS

Project Modules:

- **APEX TRIGGERS**
- **Get started with Apex Triggers**

```
trigger AccountAddressTrigger on Account (before insert,before update) {
List<Account> acclst=new List<Account>();
for(account a:trigger.new){
if(a.Match_Billing_Address__c==true && a.BillingPostalCode!=null){
 a.ShippingPostalCode=a.BillingPostalCode;
      }
    }
   }
```

- **Bulk Apex Triggers**

```
 trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List<Task> taskList = new List<Task>();

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

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

- **APEX TESTING**
- **Getting started with Apex Testing**

```
  trigger AccountAddressTrigger on Account (before insert,before update) {
 List<Account> acclst=new List<Account>();
for(account a:trigger.new){
if(a.Match_Billing_Address__c==true && a.BillingPostalCode!=null){
```

```
        a.ShippingPostalCode=a.BillingPostalCode;
              }
           }
        }
```

➤ **Test Apex Triggers**

RestrictContactByName :

```
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 :

```
@isTest
private class TestRestrictContactByName {
static testMethod void  metodoTest()
{
List<Contact> listContact= new List<Contact>();
Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
listContact.add(c1);
listContact.add(c2);

Test.startTest();
  try
  {
     insert listContact;
  }


  catch(Exception ee)
  {
  }

Test.stopTest();

  }

}
```

➤ **Create Test Data for Apex Tests**

RandomContactFactory class :

```
  public class RandomContactFactory {
  public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
  List<Contact> contactList = new List<Contact>();
  for(Integer i=0;i<numContactsToGenerate;i++) {
        Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
        contactList.add(c);
        System.debug(c);
     }
     //insert contactList;
     System.debug(contactList.size());
     return contactList;
  }

}
```

➢ **ASYNCHRONOUS APEX**

➢ **Use Future Methods**

```
public class AccountProcessor {
public static void countContacts(List<Id> accountIds){
     List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
     List<Account> updatedAccounts = new List<Account>();
     for(Account account : accounts){
       account.Number_of_Contacts__c = [Select count() from Contact Where AccountId =:
account.Id];
        System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
        updatedAccounts.add(account);
     }
     update updatedAccounts;
  }

}


//test class for the same

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

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

     Contact c2 = new Contact();
```

```
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

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

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

}
```

➤ **Use Batch Apex**

```
public class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
          for (Lead Lead : leads) {
             lead.LeadSource = 'Dreamforce';
          }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
     }

}

//test class for the same

public class LeadProcessorTest{
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company ='demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }
```

```
    static void test() {
     Test.startTest();
     LeadProcessor leadProcessor = new LeadProcessor();
     Id batchId = Database.executeBatch(leadProcessor);
     Test.stopTest();
   }
}
```

➤ **Control Processes with Queueable Apex**

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

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

   }

}

//test class for the same


public class AddPrimaryContactTest
{
  static void TestList()
```

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

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

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

► Schedule Jobs Using the Apex Scheduler

```
public class DailyLeadProcessor implements Schedulable  {
   Public void execute(SchedulableContext SC){
     List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
      for(Lead l:LeadObj){
         l.LeadSource='Dreamforce';
         update l;
      }
   }
}


//test class for the same


private class DailyLeadProcessorTest {
static testMethod void testDailyLeadProcessor()
{
          String CRON_EXP = '0 0 1 * * ?';
          List<Lead> lList = new List<Lead>();
           for (Integer i = 0; i < 200; i++) {
          lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open -
Not Contacted'));
                    }
                    insert lList;

                    Test.startTest();
```

```
                String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
        }
}
```

- **APEX INTEGRATION SERVICES**
- **Apex REST Callouts**

```
//Class AnimalLocator class

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');
   }
}
```

```
//AnimalLocatorTest class

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 class


global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
      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;
   }
```

```
}
```

➤ **Apex SOAP Callouts**

```
//ParkLocator class

public class ParkLocator {
   public static string[] country(string theCountry) {
      ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort();
      return parkSvc.byCountry(theCountry);
   }
}



//ParkLocatorTest class

private class ParkLocatorTest {
    static void testCallout() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock ());
      String country = 'United States';
      List<String> result = ParkLocator.country(country);
      List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
       System.assertEquals(parks, result);
   }
}




//ParkServiceMock class

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>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
      response.put('response_x', response_x);
  }
}
```

➤ Apex Web Services

```
//AccountManagerTest class

private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
      request.httpMethod = 'GET';
      RestContext.request = request;
      // Call the method to test
      Account thisAccount = AccountManager.getAccount();
      // Verify results
      System.assert(thisAccount != null);
      System.assertEquals('Test record', thisAccount.Name);

   }

   // Helper method
      static Id createTestRecord() {
      // Create test record
      Account TestAcc = new Account(
        Name='Test record');
      insert TestAcc;
      Contact TestCon= new Contact(
      LastName='Test',
      AccountId = TestAcc.id);
      return TestAcc.Id;
   }
}


//AccountManager class

@RestResource(urlMapping='/Accounts/*/contacts')
global 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;
   }
}
```

Project Superbadge modules

- **APEX SPECIALIST**

➤ **Step 2**

```
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);
            }
         }
      }

          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>();

         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 (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;
    }
  }
}
```

➤ Step 3

```
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();
                //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');
    }

}
```

➤ **Step 4**

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

➤ **Step 5**

```
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);
            }
         }
      }

          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>();

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


  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;
  }
```

```
    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;
    }

    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.g
et(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);
    }
}
```

➤ **Step 6**

```
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();
            //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');
    }

}
```

```
@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","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;
   }
}
```

```
@IsTest
private class WarehouseCalloutServiceTest {
          @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);
   }
}
```

➤ **Step 7**

```
@IsTest
private class WarehouseCalloutServiceTest {
          @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);
    }
}
```

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

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @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();
    }
}
```

- **Process Automation Specialist**

➤ Step 2

```
OR(
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", State)),
LEN(State) <> 2,
NOT(OR(Country ="US",Country ="USA",Country ="United States", ISBLANK(Country)))
)
```

➤ **Step 3**

```
[ValidationForBilling]
OR(
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", BillingState)),
LEN(BillingState) <> 2,
NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States",
ISBLANK(BillingCountry))),
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:
MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:
WI:WY", ShippingState)),
LEN(ShippingState) <> 2,
NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States",
ISBLANK(ShippingCountry )))
)

[ValidationForType]
ISCHANGED(Name) && (OR(ISPICKVAL(Type, 'Customer - Direct'), ISPICKVAL(Type, 'Customer
- Channel')))
```

➤ **Step 4**

```
CASE(weekday(Date__c),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
5,"Thusday",
6,"Friday",
7,"Saturday",
Text(weekday(Date__c))
)
```

➤ **Step 5**

```
if((Amount > 1000 && Approved__c = false && ispickVal(stageName  , "Closed Won")),true ,
false )
```

➤ **Step 8**

```
CASE(
MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7),
0, [Opportunity].CloseDate + 181,
6, [Opportunity].CloseDate + 182,
[Opportunity].CloseDate + 180
)
```