Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Name : Vivekkumar Panchal
Email : 200390116035@saffrony.ac.in
College Name : S. P. B. Patel Engineering College

1) Module Name :- **Apex Triggers**

Unit-1 :- **Get Started with Apex Triggers**

Code :-

```
trigger AccountAddressTrigger on Account(before insert, before update) {
    if(Trigger.isInsert){
        if (Trigger.isBefore) {
            for(Account a: Trigger.New){
              if(a.Match_Billing_Address__c){
                 a.ShippingPostalCode = a.BillingPostalCode;
              }
            }
            System.debug('Hello before insert');
        }
    }
     if(Trigger.isUpdate){
        if (Trigger.isBefore) {
            for(Account a: Trigger.New){
              if(a.Match_Billing_Address__c){
                 a.ShippingPostalCode = a.BillingPostalCode;
              }
            }
            System.debug('Hello before update');
        }
    }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Unit-2 :- **Bulk Apex trigger**

Code :-

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List<Task> taskList = new List<Task>();
        for(Opportunity opp: [SELECT Id, StageName FROM Opportunity WHERE
   StageName='Closed Won']){
    taskList.add(new Task(Subject='Follow Up Test Task',
                 whatId=opp.Id));
  }

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

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

2) Module Name :- **Apex Testing**

Unit-1 :- **Get Started with Apex Unit Tests**

Code :-

```
public class VerifyDate {
    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of
    the month
        if(DateWithin30Days(date1,date2)) {
                return date2;
        } else {
                return SetEndOfMonthDate(date1);
        }
    }

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

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

    //method to return the end of the month of a given date
    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;
    }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Code :-

```
@isTest
public class TestVerifyDate {
   @isTest static void testDate2Within30Days(){
      Date date1 = Date.newInstance(2022, 5, 13);
      Date date2 = Date.newInstance(2022, 6, 05);
      Date d = VerifyDate.CheckDates(date1,date2);
      System.assertEquals(date2, d);
   }
   @isTest static void testDate2IsNotWithin30Days(){
      Date date1 = Date.newInstance(2022, 5, 13);
      Date date2 = Date.newInstance(2022, 6, 28);
      Date d = VerifyDate.CheckDates(date1,date2);
           System.assertEquals(Date.newInstance(2022, 5, 31), d);
   }

}
```

Unit-2 :- **Test Apex Triggers**

Code :-

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

   //check contacts prior to insert or update for invalid data
   For (Contact c : Trigger.New) {
      if(c.LastName == 'INVALIDNAME') {   //invalidname is invalid
         c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
      }

   }
}
```

Unit-3 :- **Create Test Data for Apex Tests**

Code :-

```
public class RandomContactFactory {
  public static List<Contact> generateRandomContacts(Integer numConts, String lastName) {
      List<Contact> conts = new List<Contact>();
      for(Integer i=0;i<numConts;i++) {
```

```
            Contact a = new Contact(FirstName='Test'+i,LastName= lastName);
            conts.add(a);
        }
        return conts;
    }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

3) Module Name :- **Asynchronous Apex**

Unit-1 :- **Use Future Methods**

Code :-

```
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;
   }
}
```

Code :-

```
@isTest
private class  AccountProcessorTest {
     @isTest
   private static void countContacts(){
      //Load Test
      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;

      //Do the test
```

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


        //Check the result
        List<Account> accs = [SELECT Id, Number_Of_Contacts__c FROM Account];
        for(Account acc: accs){
            System.assertEquals(1, acc.Number_Of_Contacts__c, 'ERROR: At least 1 Account record
          with incorrect contacts');
        }
      }
    }
```

Unit-2 :- **Use Batch Apex**

Code :-

```
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){
        // process each batch of records
        for (Lead l : leads) {
          l.LeadSource = 'Dreamforce';
        }
        update leads;
    }

    public void finish(Database.BatchableContext bc){

    }
}
```

Code :-

```
    @isTest
    private class LeadProcessorTest {
      @isTest
      private static void testBatchClass(){
            //Load the data
            List<Lead> leads = new List<Lead>();
            for(Integer i=0;i<200;i++){
```

```
                leads.add(new Lead(LastName='Connock ',Company = 'Salesforce'));
            }
        insert leads;

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

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

Unit-3 :- **Control Processes with Queueable Apex**

Code :-

```
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) {
          //Retrieve 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 account records
        for(Account acc: accounts){
              //Clone (copy) the Contact record, make the clone a child of the specific Account record
          //and add to list of Contacts
```

```
                    Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }
}
```

Code :-

```
@isTest
private class AddPrimaryContactTest {
        @isTest
    private static void testQueueableClass(){

        //Load the data
                List<Account> accounts = new List<Account>();
        for(Integer i=0;i<500;i++){
            Account acc = new Account(Name='Test 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;

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

                //Check the result
                List<Contact> contacts = [SELECT Id FROM Contact WHERE
Contact.Account.BillingState = 'CA'];
        System.assertEquals(200, contacts.size(),'ERROR: Incorrect number of Contact records
found');
    }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Unit-4 :- **Schedule Jobs Using the Apex Scheduler**

Code :-

```
public without sharing class DailyLeadProcessor implements Schedulable{
        public void execute(SchedulableContext ctx) {
    //System.debug('Context' + ctx.getTriggerId());
    //Returns the Id of the CronTrigger Scheduled jobs

      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT
200];
     for(Lead l : leads){
        l.LeadSource = 'Dreamforce';
     }

     //Update the modified records
     update leads;
   }
}
```

Code :-

```
@isTest
private class DailyLeadProcessorTest {

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

      @isTest
  private static void testSchedulerClass(){
    //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
='Others'));
      }
    }
    insert leads;

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

```
        //Check the result
           List<Lead> updateLeads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =
'Dreamforce'];
            System.assertEquals(200, updateLeads.size(), 'ERROR: At leasr 1 record not update
correctly');

        //Check the scheduled time
           List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime FROM CronTrigger
WHERE Id= :jobId];
        System.debug('Next Fire Time' + cts[0].NextFireTime);
    }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

4) Module Name :- **Apex Integration Services**

Unit-1 :- **Apex REST callouts**

Code :-

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

Code :-

```
@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 );
  }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Unit-2 :- **Apex SOAP Callouts**

Code :-

```
public class ParkLocator {

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

Code :-

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

Unit-3 :- **Apex Web Services**

Code :-

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {

   @HttpGet
   global static Account getAccount(){
      RestRequest request = RestContext.request;
      String accountId = request.requestURI.substringBetween('Accounts/', '/contacts');
      Account result =  [SELECT Id,Name,(SELECT Id, FirstName, LastName FROM Contacts)
                  FROM Account
            WHERE Id = :accountId];
      return result;
   }
}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Code :-

```
@isTest
private class AccountManagerTest {

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

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

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

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Apex Specialist Superbadge

Step - 1 **: Automate record creation using Apex triggers.**

Code :-

```
public with sharing class MaintenanceRequestHelper {

   public static void updateWorkOrders() {
      // TODO: Complete the method to update workorders
      List<Case> cases = Trigger.New;
      Set<Id> validId = new Set<Id>();
      for(Case c: cases) {
         if(c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
            if(c.Status == 'Closed'){
               validId.add(c.Id);
            }
          }
          }
      if (!validId.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 :validId]);
         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 :validId GROUP BY
Maintenance_Request__c];

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

         List<Case> newCases = new List<Case>();
         for(Case c1 : closedCases.values()){
            Case newCase = new Case (
               ParentId = c1.Id,
               Status = 'New',
               Subject = 'Routine Maintenance',
```

```
                Type = 'Routine Maintenance',
                Vehicle__c = c1.Vehicle__c,
                Equipment__c =c1.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );

            If (maintenanceCycles.containskey(c1.Id)){
                                    newCase.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(c1.Id));
            } else {
                                    newCase.Date_Due__c = Date.today().addDays((Integer)
c1.Equipment__r.maintenance_Cycle__c);
            }

             newCases.add(newCase);
           }
           insert newCases;

                                List<Equipment_Maintenance_Item__c> ItemList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
                                        for (Equipment_Maintenance_Item__c ListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c item = ListItem.clone();
            item.Maintenance_Request__c = nc.Id;
            ItemList.add(item);
          }
        }
        insert ItemList;

         }
   }
}
```

Code :-

```
trigger MaintenanceRequest on Case (before update, after update) {
   // ToDo: Call MaintenanceRequestHelper.updateWorkOrders
   if(Trigger.isUpdate && Trigger.isAfter){
      MaintenanceRequestHelper.updateWorkOrders();
   }

}
```

Salesforce Virtual Internship Program - 2022 (Salesforce Developer)

Step - 2 **: Synchronize Salesforce data with an external system using asynchronous REST callouts.**

Code :-

```
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 makeCallout(){
    System.debug('Enter');
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);

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

      for(Object data: getData) {
        Map<String, Object> mapData = (Map<String, Object>) data;
        Product2 product2 = new Product2();
        product2.Replacement_Part__c = (Boolean) mapData.get('replacement');
        product2.Cost__c = (Integer) mapData.get('cost');
        product2.Current_Inventory__c = (Double) mapData.get('quantity');
        product2.Lifespan_Months__c = (Integer) mapData.get('lifeSpan');
        product2.Maintenance_Cycle__c = (Integer) mapData.get('maintenanceperiod');
        product2.Warehouse_SKU__c = (String) mapData.get('sku');
        product2.Name = (String) mapData.get('name');
        product2.ProductCode = (String) mapData.get('_id');
        product.add(product2);
      }

      if(product.size() > 0){
        upsert product;
        System.debug('updated');
      }
```

```
        }
    }

     public static void execute (QueueableContext context) {
        System.debug('Succesfully excuted');
        makeCallout();
        System.debug('Done');
    }
}
```

Step - 3 **: Schedule synchronization using Apex code.**

Code :-

```
Global with sharing class WarehouseSyncSchedule implements Schedulable{
    // implement scheduled code here
    Global static void execute(SchedulableContext ctx) {
        System.enqueueJOb(new WarehouseCalloutService());
    }
}
```

Step - 4 **: Test automation logic to confirm Apex trigger side effects**

Code :-

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
    // implement scheduled code here

    private static Product2 equipment(){
        Product2 equp = new Product2(name ='Sample Test Equipment',
                                     Lifespan_Months__c = 10,
                                     Maintenance_Cycle__c = 10,
                                     Replacement_Part__c = true);
                        return equp;
    }

    private static Vehicle__c vehicle(){
        Vehicle__c veh = new Vehicle__c(name = 'Sample Test Vehicle');
                        return veh;
    }
```

```
    private static Case maintenanceRequest(id vehId, id equpId){
        Case mR = new Case(Type = 'Repair',
                           Status = 'New',
                           Origin = 'Web',
                           Subject = 'Sample Testing',
                           Equipment__c = equpId,
                           Vehicle__c = vehId);
                return mR;
    }

    private static Equipment_Maintenance_Item__c equpMI(id equpId, id requestId){
                                        Equipment_Maintenance_Item__c eMI = new
Equipment_Maintenance_Item__c(Equipment__c = equpId,
                                                Maintenance_Request__c = requestId);
                return eMI;
    }

    @isTest
    private static void positiveCase(){
        Product2 Equipment = equipment();
        insert Equipment;
        id equpId = Equipment.Id;

        Vehicle__c Vehicle = vehicle();
        insert Vehicle;
        id vehId = Vehicle.Id;

        Case MaintenanceRequest = maintenanceRequest(vehId, equpId);
        insert MaintenanceRequest;

        Test.startTest();
        MaintenanceRequest.Status = 'Closed';
        update MaintenanceRequest;
        Test.stopTest();

        Case newCases = [Select id,
                Subject,
                Type,
                Equipment__c,
                Date_Reported__c,
                Vehicle__c,
                Date_Due__c
            FROM Case
            WHERE Status ='New'];
```

```
    Equipment_Maintenance_Item__c emiRecord = [SELECT id
                                    FROM Equipment_Maintenance_Item__c
                                    WHERE Maintenance_Request__c =:newCases.Id];
    List<Case> testCase = [SELECT id FROM Case];
    System.assert(testCase.size() == 2);

    System.assert(newCases != null);
    System.assert(newCases.Subject != null);
    System.assertEquals(newCases.Type, 'Routine Maintenance');
    System.assertEquals(newCases.Equipment__c, equpId);
    System.assertEquals(newCases.Vehicle__c, vehId);
    System.assertEquals(newCases.Date_Reported__c, system.today());
}

@isTest
private static void negativeCase(){
    Product2 Equipment = equipment();
    insert Equipment;
    id equpId = Equipment.Id;

    Vehicle__c Vehicle = vehicle();
    insert Vehicle;
    id vehId = Vehicle.Id;

    Case MaintenanceRequest = maintenanceRequest(vehId, equpId);
    insert MaintenanceRequest;

    Test.startTest();
    MaintenanceRequest.Status = 'Working';
    update MaintenanceRequest;
    Test.stopTest();

    List<Case> testCase = [SELECT id FROM Case];
    Equipment_Maintenance_Item__c emiRecord = [SELECT id
                                    FROM Equipment_Maintenance_Item__c
                                    WHERE Maintenance_Request__c =:MaintenanceRequest.Id];

    System.assert(emiRecord != null);
    System.assert(testCase.size() == 1);

}
```

```
    @isTest
    private static void bulkyTest(){
        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(Vehicle());
            equipmentList.add(Equipment());
        }
        insert vehicleList;
        insert equipmentList;

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

        for(integer i = 0; i < 300; i++){
                                equipmentMaintenanceItemList.add(equpMI(equipmentList.get(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> emiRecord = [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 - 5 **: Test integration logic using callout mocks**

Code :-

```
@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,"na
me":"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;
    }
}
```

Code :-

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

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

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

Step - 6 **: Test scheduling logic to confirm action gets queued**

Code :-

```
@isTest
public with sharing class WarehouseSyncScheduleTest {

    @isTest static void testSchedulecase(){
    String CRON_EXP = '00 00 00 * * ? *';
    Test.startTest();
    Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
                String jobId = System.schedule('WarehouseSyncJob', CRON_EXP, new
WarehouseSyncSchedule());
    CronTrigger ct = [SELECT State FROM CronTrigger WHERE Id =: jobId];
    System.assertEquals('WAITING', string.valueOf(ct.State), 'Job Id must be match');
    Test.stopTest();
  }
}
```