MODULE: Apex Triggers

Unit 1:
AccountAddressTrigger.apxt

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

Unit 2:
ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
        List<Task> taskList= new List<Task>();
  for(Opportunity o: Trigger.New)
  {
    if(o.StageName=='Closed Won')
      taskList.add(new Task(Subject= 'Follow Up Test Task', WhatId= o.Id));
  }
  insert taskList;
}
```

MODULE: Apex Testing

Unit 1:
TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
  @isTest static void testOldDate(){
    Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));
    System.assertEquals(date.newInstance(2022, 5, 31), dateTest);
  }

  @isTest static void testLessThan30Days(){
```

```
        Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(20));
        System.assertEquals(date.today().addDays(20), dateTest);
    }

    @isTest static void testMoreThan30Days(){
        Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(31));
        System.assertEquals(date.newInstance(2022, 5, 31), dateTest);
    }

}
```

Unit 2:
TestRestrictContactByName.apxc

```
@isTest
private class TestRestrictContactByName {

    static testMethod void  nameTest()
    {

        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='ABC', LastName='XYZ' ,
email='Test@test.com');
        Contact c2 = new Contact(FirstName='DEF', LastName =
'INVALIDNAME',email='Test@test.com');
        listContact.add(c1);
        listContact.add(c2);

        Test.startTest();
            try
            {
                insert listContact;
            }
            catch(Exception ee)
            {
            }

        Test.stopTest();
```

```
    }

}
```

Unit 3:
RandomContactFactory.apxc

```
public class RandomContactFactory
{
    public static List<Contact> generateRandomContacts(Integer n, String last)
    {
        List<Contact> contactsList=new List<Contact>();
        for(Integer i=0;i<n; i++)
        {
            Contact con=new Contact(FirstName='Test '+i, LastName=last);
            contactsList.add(con);
        }

        return contactsList;
    }

}
```

MODULE: Asynchronous Apex

Unit 2:
AccountProcessor.apxc

```
 public class AccountProcessor {
   @future
   public static void countContacts(Set<Id> conId)
   {
     List<Account> accountList = [SELECT ID, Name, Number_Of_Contacts__c, (SELECT ID
From Contacts)FROM Account WHERE ID IN :conId];
     for(Account a : accountList)
     {
       List<Contact> conList = a.Contacts;
       a.Number_Of_Contacts__c = conList.Size();
     }
```

```
    update accountList;
  }
 }
```

AccountProcessorTest.apxc

```
 @isTest
 public class AccountProcessorTest {
     public static testMethod void creatingAccount()
  {
    Account a = New Account();
    a.Name = 'Test-Account';
    Insert a;
    Contact con = New Contact();
    con.FirstName = 'Test-FirstName-Contact';
    con.LastName = 'Test-LastName-Contact';
    con.AccountId = a.Id;
    Insert con;
    Set<Id> aId = new Set<Id>();
    aId.add(a.Id);
    Test.startTest();
        AccountProcessor.countContacts(aId);
    Test.stopTest();
    Account aList = [SELECT Number_of_Contacts__c FROM Account where id = : a.Id
LIMIT 1];
    System.assertEquals(aList.Number_of_Contacts__c, 1);
  }
 }
```

Unit 3:

LeadProcessor.apxc

```
 global class LeadProcessor implements  Database.Batchable<Sobject>
 {
  global Database.QueryLocator start(Database.BatchableContext bc)
  {
    return Database.getQueryLocator([Select LeadSource From Lead ]);
  }
  global void execute(Database.BatchableContext bc, List<Lead> scope)
```

```
  {
     for (Lead Leads : scope)
     {
       Leads.LeadSource = 'Dreamforce';
     }
    update scope;
  }
  global void finish(Database.BatchableContext bc){  }
}
```

LeadProcessorTest.apxc

```
 @isTest
 public class LeadProcessorTest
 {
   static testMethod void testingMethod()
   {
     List<Lead> lstLead = new List<Lead>();
     for(Integer i=0 ;i <200;i++)
     {
       Lead led = new Lead();
       led.FirstName ='FirstName';
       led.LastName ='LastName'+i;
       led.Company ='demo'+i;
       lstLead.add(led);
     }
     insert lstLead;
     Test.startTest();
       LeadProcessor obj = new LeadProcessor();
       DataBase.executeBatch(obj);
     Test.stopTest();
   }
 }
```

Unit 4:

AddPrimaryContact.apxc

```
 public class AddPrimaryContact implements Queueable {
   public Contact con;
```

```
  public String state;
    public AddPrimaryContact(Contact con, String state)
  {
   this.con = con;
   this.state = state;
  }
  public void execute(QueueableContext qc)
  {
   List<Account> accList = [SELECT Id, Name, BillingState FROM Account WHERE
Account.BillingState =: this.state Limit 200];
   List<Contact> newContact = new List<Contact>();
   for(Account acc : accList)
   {
    Contact con = new Contact();
    con = this.con.clone(false, false, false, false);
    con.AccountId = acc.Id;
    newContact.add(con);
   }
   insert newContact;
  }
 }

AddPrimaryContactTest.apxc
 @isTest
 public class AddPrimaryContactTest {
 @testSetup
 static void setup() {
    List<Account> insertAccount = new List<Account>();
    for(integer i=0; i<=100; i++) {
    if(i <=50) {
    insertAccount.add(new Account(Name='Acc'+i, BillingState = 'NY'));
    }
   else {
        insertAccount.add(new Account(Name='Acc'+i, BillingState = 'CA'));
        }
    }
    insert insertAccount;
```

```
    }
 static testMethod void testAddPrimaryContact() {
  Contact con = new Contact(LastName = 'LastName');
  AddPrimaryContact addPC = new AddPrimaryContact(con, 'CA');
  Test.startTest();
  system.enqueueJob(addPC);
  Test.stopTest();
    system.assertEquals(50, [SELECT count() FROM Contact]);
}}
```

Unit 5:

DailyLeadProcessor.apxc

```
 global class DailyLeadProcessor implements Schedulable{
  global void execute(SchedulableContext sc){
    List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
    List<Lead> lstOfUpdatedLead = new List<Lead>();
    if(!lstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
        ld.LeadSource = 'Dreamforce';
        lstOfUpdatedLead.add(ld);
      }
      UPDATE lstOfUpdatedLead;
    }
  }
 }
```

DailyLeadProcessorTest.apxc

```
@isTest
 private class DailyLeadProcessorTest{
  @testSetup
  static void setup(){
    List<Lead> listOfLead = new List<Lead>();
    for(Integer i = 1; i <= 200; i++){
      Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working -
Contacted');
      listOfLead.add(ld);
    }
```

```
    Insert listOfLead;
  }
  static testmethod void testDailyLeadProcessorScheduledJob(){
    String sch = '0 5 12 * * ?';
    Test.startTest();
    String jobId = System.schedule('ScheduledApexTest', sch, new
DailyLeadProcessor());
    List<Lead> listOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT
200];
    System.assertEquals(200, listOfLead.size());
    Test.stopTest();
  }
}
```

MODULE: Apex Integration Services

Unit 2:
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 = '';
      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');
         strResp = string.valueof(animals.get('name'));
      }
```

```
        return strResp ;
    }

}
```

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;
    }
}
```

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);
    }
}
```

Unit 3:

ParkLocator.apxc

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

```
}
```

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();
      List<String> myStrings = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = myStrings;
      response.put('response_x', response_x);
  }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest  {
   @isTest static void testCallout() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      List<String> result = new List<String>();
      List<String> expectedvalue = new List<String>{'Park1','Park2','Park3'};
      result = ParkLocator.country('India');
      System.assertEquals(expectedvalue, result);
   }
}
```

Unit 4:

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
   @HttpGet
   global static Account getAccount(){
      RestRequest request = RestContext.request;
      String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
      Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM Contacts) FROM
Account WHERE Id = :accountId LIMIT 1];
      return objAccount;
   }
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest{
   static testMethod void testMethod1(){
      Account objAccount = new Account(Name = 'test Account');
      insert objAccount;
      Contact objContact = new Contact(LastName = 'test Contact',
                                 AccountId = objAccount.Id);
      insert objContact;
      Id recordId = objAccount.Id;
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://sandeepidentity-dev-ed.my.salesforce.com/services/apexrest/Accounts/'
         + recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisAccount = AccountManager.getAccount();
      System.assert(thisAccount!= null);
      System.assertEquals('test Account', thisAccount.Name);
   }
}
```

SUPERBADGE 1: APEX SPECIALIST

Unit 2:
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()){
         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;
    }
  }
}
```

MaintenanceRequest.apxt

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

Unit 3:

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');
   }
}
```

Unit 4:
WarehouseSyncSchedule.apxc

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

Unit 5:
MaintenanceRequestHelperTest.apxc

```
@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.
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> 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);
    }
}
```

Unit 6:

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","replacement":false,"quantity":5
,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceTest.apxc

```
@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);
    }
}
```

Unit 7:
WarehouseSyncScheduleTest.apxc

```
@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();
    }
}
```