SPSGP-15231-Salesforce Developer Catalyst Self-Learning & Super Badges

Apex Integration Services Code:

1) 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 result = [SELECT Id, Name, (Select Id, Name from Contacts)
from Account where Id=:accountId Limit 1];
        return result;
    }
}
```

2)AccountManagerTest.apxc:

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
```

```
                        + recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }

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

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

        return accountTest.Id;
    }
}
```

3)AnimalLocator.apxc:

```
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');
    }
}
```

4)AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
     // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny",
"scary bear", "chicken", "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

5)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 );
    }
}
```

6)AsyncParkService.apxc:

```
//Generated by wsdl2apex

public class AsyncParkService {
    public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(
this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
```

```
            this,
            request_x,
            AsyncParkService.byCountryResponseFuture.class,
            continuation,
            new String[]{endpoint_x,
            '',
            'http://parks.services/',
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
      }
   }
}
```

7)ParkLocator.apxc:

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

8)ParkLocatorTest.apxc:

```
@isTest
private class ParkLocatorTest {
   @isTest 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);
    }
}
```

9)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) {
        // start - specify the response you want to send
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone', 'Mackinac
National Park', 'Yosemite'};
        // end
        response.put('response_x', response_x);
  }
```

}

Apex Specialist:

1)CreateDefaultData.apxc:

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine
Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to
know if Default data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest =
createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords =
createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }
```

```
    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings__c customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Toy Hauler RV'));
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV',
Air_Conditioner__c = true, Bathrooms__c = 2, Bedrooms__c = 2, Model__c
= 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c =
'55d66226726b611100aaf741',name = 'Generator 1000 kW',
Replacement_Part__c = true,Cost__c = 100 ,Maintenance_Cycle__c =
100));
        equipments.add(new Product2(name = 'Fuse
20B',Replacement_Part__c = true,Cost__c = 1000, Maintenance_Cycle__c
= 30  ));
```

```
        equipments.add(new Product2(name = 'Breaker
13C',Replacement_Part__c = true,Cost__c = 100  , Maintenance_Cycle__c
= 15));
        equipments.add(new Product2(name = 'UPS 20
VA',Replacement_Part__c = true,Cost__c = 200  , Maintenance_Cycle__c =
60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c>
vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c =
vehicles.get(1).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c =
vehicles.get(2).Id, Type = TYPE_ROUTINE_MAINTENANCE,
Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c>
createJoinRecords(List<Product2> equipment, List<Case>
maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new
List<Equipment_Maintenance_Item__c>();
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
```

```
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

2)CreateDefaultDataTest.apxc:

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id
FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4
```

```
vehicles created');
        System.assertEquals(4, equipment.size(), 'There should have been 4
equipment created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should
have been 2 maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been
6 equipment maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c       customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The
custom setting How_We_Roll_Settings__c.Is_Data_Created__c should be
false');

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The
custom setting How_We_Roll_Settings__c.Is_Data_Created__c should be
true');

    }
}
```

3)MaintenanceRequest.apxt:

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

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

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

    }

}
```

4)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;
    }
  }
}
```

5)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);
    }
}
```

6)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();
    }
}
```

//Execute window code:

System.enqueueJob(new WarehouseCalloutService());

7)WarehouseCalloutService2.apxc:

```
public with sharing class WarehouseCalloutService {

    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 = (Decimal) mapJson.get('lifespan');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
        }

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

8)WarehouseCalloutServiceMock.apxc:

```
@isTest
public class WarehouseCalloutServiceMock implements
HTTPCalloutMock {
// implement http mock callout
public HTTPResponse respond (HttpRequest request){
HttpResponse response = new HTTPResponse();
response.setHeader('Content-type','application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":f
alse,"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":"100004"},{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

response.setStatusCode(200);

return response;

}

}

9)WarehouseCalloutServiceTest.apxc:

```
@isTest
private class WarehouseCalloutServiceTest {
  @isTest
  static void testWareHouseCallout(){
    Test.startTest();
    Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.runWarehouseEquipmentSync();
    Test.stopTest();
    System.assertEquals(1, [SELECT count() FROM Product2]);
  }
}
```

10)WarehouseSyncSchedule.apxc:

```
global with sharing class WarehouseSyncSchedule implements Schedulable{

  global void execute(SchedulableContext ctx){

    System.enqueueJob(new WarehouseCalloutService());

  }
```

```
}
```

11)WarehouseSyncSchedule2.apxc:

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

12)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();
      //Contains schedule information for a scheduled job. CronTrigger is
similar to a cron job on UNIX systems.
      // This object is available in API version 17.0 and later.
      CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime >
today];
      System.assertEquals(jobID, a.Id,'Schedule ');
   }
}
```

Apex Testing:

1)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;

}
```

2)RestrictContactByName.apxt:

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

3)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());
    }
}
```

4)TestVerifyDate.apxc:

```
@isTest
private class TestVerifyDate{

    @istest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

        @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates (date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

        @isTest static void Test_DateWithin30Days_case1() {
        Boolean flag = VerifyDate.DateWithin30Days
(date.parse('01/01/2020'), date.parse( '12/30/2019'));
                System.assertEquals(false, flag);
    }

        @istest static void Test_DateWithin30Days_case2(){
```

```
    Boolean flag = VerifyDate.DateWithin30Days
(date.parse('01/01/2020'), date.parse('02/02/2020'));
            System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3() {
    Boolean flag = VerifyDate.DateWithin30Days
(date.parse('01/01/2020'), date.parse('01/15/2020'));
    System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
    Date returndate =
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }

}
```

5)VerifyDate.apxc:

```
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
    @TestVisible 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
    @TestVisible 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;
    }

}
```

Apex Trigers:

1)AccountAddressTrigger.apxt:

```
trigger AccountAddressTrigger on Account (before insert,before
update){

  for(Account account:Trigger.New){
    if(account.Match_Billing_Address__c == True){
      account.ShippingPostalCode = account.BillingPostalCode;
    }
```

```
    }
}
```

2)ClosedOpportunityTrigger.apxt:

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

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId =
opp.Id));
        }
    }

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

}
```

Asynchronous Apex:

1)AccountProcessor.apxc:

```
public class AccountProcessor{
        @future
    public static void countContacts(List<Id> accountIds){

        List<Account> accountsToUpdate = new List<Account>();

                List<Account> accounts = [Select Id, Name, (Select Id from
Contacts) from Account Where Id in :accountIds];
```

```
            For(Account acc:accounts){
                List<Contact> contactList = acc.Contacts;
        acc.Number_Of_Contacts__c = contactList.size();
        accountsToUpdate.add(acc);

    }
            update accountsToUpdate;
  }
}
```

Open Execute Anonymous window Code :

```
List<Id> accountIds = new List<Id>();
accountIds.add('001Iw000002XTKCIA4');

AccountProcessor.countContacts(accountIds);
```

2)AccountProcessorTest.apxc:

```
@IsTest
private class AccountProcessorTest{
  @IsTest
  private static void testCountContacts(){
    Account newAccount = new Account(Name = 'Test Account');
    insert newAccount;

    Contact newContact1 = new Contact(FirstName='John',
LastName='Doe', AccountId = newAccount.id);
    insert newContact1;

            Contact newContact2 = new Contact(FirstName='Jane',
LastName='Doe', AccountId = newAccount.id);
```

```
            insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

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

    }
}
```

3)AddPrimaryContact.apxc:

```
public class AddPrimaryContact implements Queueable{
    private Contact con;
    private String state;
    public AddPrimaryContact (Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName,
LastName, Id from contacts)
                                                    from Account where
BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();


        for (Account acc:accounts){
```

```
            Contact c = con.clone();
             c.AccountId = acc.Id;
             primaryContacts.add(c);
        }
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }
}
```

4)AddPrimaryContactTest.apxc:

```
@isTest
public class AddPrimaryContactTest{

    static testmethod void testQueueable(){
        List<Account> testAccounts=new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account (Name='Account
'+i,BillingState='CA'));
        }
        for(Integer j=0;j<5;j++){
            testAccounts.add(new Account (Name='Account '+j,
BillingState='NY'));
        }
        insert testAccounts;

        Contact testContact=new Contact(FirstName='John', LastName
='Doe');
        insert testContact;
```

```
        AddPrimaryContact addit=new addPrimaryContact(testContact,
'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

                System.assertEquals(50,[Select count() from Contact where
accountId in (Select Id from Account where BillingState='CA')]);
         }
}
```

5)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;
      }
   }
}
```

6)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();
    }
}
```

7)LeadProcessor.apxc:

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;

    global Database.Querylocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT ID, LeadSource FROM
Lead');
    }


    global void execute (Database.BatchableContext bc, List<Lead>
L_list){
        List<lead> L_list_new = new List<lead>();


        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
```

```
        update L_list_new;
    }

    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count);
    }
}
```

8)LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {

        @isTest
        public static void testit(){
                List<lead> L_list = new List<lead>();

                for(Integer i=0; i<200; i++){
                        Lead L = new lead();
                        L.LastName = 'name' + i;
                        L.Company = 'Company';
                        L.Status = 'Random Status';
                        L_list.add(L);
        }
                insert L_list;

                Test.startTest();
                LeadProcessor lp = new LeadProcessor();
                Id batchId = Database.executeBatch(lp);
                Test.stopTest();
    }
}
```