Superbadge-1(Apex Specialist)

Apex trigger:

//Create an apex trigger-1: 'AccountAdressTrigger'

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

//Create an apex trigger-2: 'ClosedOpportunityTrigger'

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
  List<task> tl=new List<task>();
   for(Opportunity opp:Trigger.new)
   {
      if(opp.StageName =='Closed Won')
      {
         tl.add(new Task(Subject = 'Follow Up Test Task',WhatId=opp.Id));
      }
   }
   if(tl.size()>0)
   {
      insert tl;
   }
}
```

Apex Testing:

//Create an apex class 'VerifyDate'

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
  List<task> tl=new List<task>();
   for(Opportunity opp:Trigger.new)
   {
      if(opp.StageName =='Closed Won')
      {
         tl.add(new Task(Subject = 'Follow Up Test Task',WhatId=opp.Id));
      }
   }
   if(tl.size()>0)
```

```
    {
        insert tl;
    }
}
```

//Create an apex trigger 'RestrictContactbyName'

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

//Create a class 'TestRestrictContactByName'

```
@isTest
public class TestRestrictContactByName{
    @isTest static void testRestrictContactByName () {
        Contact c = new Contact(LastName='INVALIDNAME');
        try{
            insert c;
        }
        catch(DMLException e){
              System.assert(e.getMessage().contains('The Last Name "'+c.LastName+'" is not allowed
for DML'));
            }
        }
    }
```

//Create an apex class 'RandomContactFactory'

```
public class RandomContactFactory {
  public static List<Contact> generateRandomContacts(Integer numcnt,string ln)
  {
     List<Contact> contacts=new List<Contact>();
     for(Integer i=1;i<=numcnt;i++){
        Contact ct=new Contact(FirstName='Test  '+i, LastName=ln);
        contacts.add(ct);
     }
     return contacts;
  }
}
```

Superbadge-1(Apex Specialist)

Asynchronous Apex:

//Create an apex class 'AccountProcessor'

```
public class AccountProcessor {
      @future
   public static void countContacts(List<ID>accountIds)
   {
      List<Account> atu=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> cl=acc.Contacts;
         acc.Number_Of_Contacts__c=cl.size();
         atu.add(acc);
      }
      update atu;
   }
}
```

//Create an apex classn'AccountProcessoeTest'

```
@isTest
public class AccountProcessorTest {
   @isTest
   private static void testCountContacts()
   {
      Account nA=new Account(Name='Test Account');
      insert nA;
      Contact nc1=new Contact(FirstName='John',
                        LastName='Doe',
                        AccountId=nA.Id);
      insert nc1;
      Contact nc2=new Contact(FirstName='Jane',
                        lastName='Doe',
                        AccountId=nA.Id);
      insert nc2;
      List<Id> accountIds=new List<Id>();
       accountIds.add(nA.Id);
      Test.startTest();
      AccountProcessor.countContacts(accountIds);
      Test.stopTest();
```

```
    }
}
```

//Create an apex class 'LeadProcessor'

```
public class LeadProcessor implements
    Database.Batchable<sObject> {
    // instance member to retain state across transactions
public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead'
        );
    }
public void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead>leads=new List<Lead>();
        for (Lead lead : scope) {
           lead.LeadSource='Dreamforce';
            leads.add(lead);
              }
        update leads;
        }
public void finish(Database.BatchableContext bc){

    }
}
```

//Create an apex test class 'LeadProcessorTest'

```
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit() {
        List<lead> l_lst = new List<lead>();
        for (Integer i = 0; i<200; i++) {
            Lead l = new lead();
            l.LastName = 'name'+i;
            l.company = 'company';
            l.Status =  'somestatus';
            l_lst.add(l);
        }
        insert l_lst;
        test.startTest();
        Leadprocessor lp = new Leadprocessor();
```

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

//Create an apex class 'AddPrimaryContact'

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;
    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from account where
account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}
```

//Create an apex test class 'AddPrimaryContactTest'

```
@IsTest
public class AddPrimaryContactTest {
    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
```

```
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }

}
```

//Create an apex class 'DailyLeadProcessor'

```
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;
        }
    }
}
```

//Create an apex test class 'DailyLeadProcessorTest'

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    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();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());
        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

Apex Integration Services:

//Create an apex class 'AnimalLocator'

```
public class AnimalLocator
{
  public static String getAnimalNameById(Integer id)
    {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
          String strResp = '';
           system.debug('******response '+response.getStatusCode());
           system.debug('******response '+response.getBody());
        // If the request is successful, parse the JSON response.
        if (response.getStatusCode() == 200)
        {
            // Deserializes the JSON string into collections of primitive data types.
           Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
           Map<string,object> animals = (map<string,object>) results.get('animal');
            System.debug('Received the following animals:' + animals );
            strResp = string.valueof(animals.get('name'));
            System.debug('strResp >>>>>>' + strResp );
        }
        return strResp ;
    }
```

```
}
```

//Create an apex test class 'AnimalLocatorTest'

```
@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);
   }
}

//Generated by wsdl2apex
public class ParkService {
   public class byCountryResponse {
      public String[] return_x;
      private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-
1','false'};
      private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'return_x'};
   }
   public class byCountry {
      public String arg0;
      private String[] arg0_type_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
      private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'arg0'};
   }
   public class ParksImplPort {
      public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
      public Map<String,String> inputHttpHeaders_x;
      public Map<String,String> outputHttpHeaders_x;
      public String clientCertName_x;
      public String clientCert_x;
      public String clientCertPasswd_x;
      public Integer timeout_x;
      private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
      public String[] byCountry(String arg0) {
         ParkService.byCountry request_x = new ParkService.byCountry();
         request_x.arg0 = arg0;
         ParkService.byCountryResponse response_x;
         Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
```

```
        response_map_x.put('response_x', response_x);
        WebServiceCallout.invoke(
          this,
          request_x,
          response_map_x,
          new String[]{endpoint_x,
          '',
          'http://parks.services/',
          'byCountry',
          'http://parks.services/',
          'byCountryResponse',
          'ParkService.byCountryResponse'}
        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
    }
  }
}
```

//Create an apex class 'ParkLocator'

```
//Generated by wsdl2apex
public class ParkLocator {
  public static String[] country(String country){
    ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
    String[] parksname = parks.byCountry(country);
    return parksname;
  }
}
```

//Create an apex test class 'ParkLocatorTest'

```
@isTest
private class ParkLocatorTest{
  @isTest
  static void testParkLocator() {
    Test.setMock(WebServiceMock.class, new ParkServiceMock());
    String[] arrayOfParks = ParkLocator.country('India');

    System.assertEquals('Park1', arrayOfParks[0]);
  }
}
```

//Create an apex class 'AccountManager'

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

Superbadge-1(Apex Specialist)

```
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;
    }
}
```

//Create an apex test class 'AccountManagerTest'

```
@isTest
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);
      }
    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;
    }
}
```

Apex Specialist:

//Create an apex class 'MaintenanceRequest'

Superbadge-1(Apex Specialist)

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

//Create an apex class 'MaintenanceRequestHelper'

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
             }
         }
      }
      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
         AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

      for (AggregateResult ar : results){
         maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
      }
      for(Case cc : closedCasesM.values()){
            Case nc = new Case (
               ParentId = cc.Id,
            Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()

            );
```

```
            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }
                newCases.add(nc);
        }
        insert newCases;
        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
       for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);
                }
        }
        insert ClonedWPs;
    }
  }
}
```

//Create a class ' WarehouseCalloutService'

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
  //class that makes a REST callout to an external warehouse system to get a list of equipment that
needs to be updated.
  //The callout's JSON response returns the equipment records that you upsert in Salesforce.
  @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());
      //class maps the following fields: replacement part (always true), cost, current inventory,
lifespan, maintenance cycle, and warehouse SKU
      //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
      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();
    }

}

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

//Create a class ' WarehouseCalloutService'

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    //class that makes a REST callout to an external warehouse system to get a list of equipment that
needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.
    @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());
            //class maps the following fields: replacement part (always true), cost, current inventory,
lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            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();
    }
}
```

//Create a class 'WarehouseCalloutService'

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

//Create a class 'WarehouseCalloutserviceTest'

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

//Create a class ' WarehouseCalloutServiceMock'

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){
```

```
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        // Create a fake response
        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"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

//Create a class 'WarehouseSyncScheduleTest'

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