Project Document

Apex Triggers

Get started with Apex Triggers:

AccountAddressTigger:

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

Bulk Apex Triggers:

ClosedOpportunityTigger:

```
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;
    }
}
```

Apex Testing

Get Started with Apex Unit Tests:

VerifyDate

```
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {                                    return SetEndOfMonthDate(date1);

                }
                      }

        @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }
        Date date30Days = date1.addDays(30);
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }
        @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;
        }
}
```

TestVerifyDate

```
@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('01/01/2022'));
   }
}
```

Test Apex Triggers

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert, before update) {
	For (Contact c : Trigger.New) {
		if(c.LastName == 'INVALIDNAME') {
			c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
		}
	}
}
```

TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
	@isTest static void Test_insertupdateContact()
   {
       Contact cnt= new Contact();
       cnt.LastName='INVALIDNAME';
       Test.startTest();
       Database.SaveResult
       result=Database.insert(cnt,false); Test.stopTest();
       System.assert(!result.isSuccess());
       System.assert(result.getErrors().size()>0);
```

```
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
             }
}
```
Create Test Data for Apex Testes

RandomContactFactory

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numcnt,string lastname){
        List <Contact> contacts= new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt=new Contact(FirstName='Test'+i,LastName=lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}
```

Asynchronous Apex

Use Future Methods

AccountProcessor

```
public class AccountProcessor {
@future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountToUpdate = 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();
            accountToUpdate.add(acc);
        }
        Update accountToUpdate;
    }
}
```

AccountProcessorTest

```
@isTest
public 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();
   }
}
```

Use Batch Apex

LeadProcessor

```
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);
    }
}
```

LeadProcessorTest

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

Control Processes with Queueable Apex

AddPrimaryContact

```
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;
        }
    }
}
```

AddPrimaryContactTest

```
@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<50;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')]);
    }
}
```

Schedule jobs Using the Apex Scheduler

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable {
   global void execute(SchedulableContext ctx){
      List<lead> leadstoupdate=new List<lead>();
      List <Lead> leads=[Select id from Lead where LeadSource=NULL Limit 200];
      for(Lead l:leads){
         l.LeadSource='Dreamforce';
         leadstoupdate.add(l);
      }
   update leadstoupdate;
   }
}
```

DailyLeadProcessorTest

```
@isTest
public class DailyLeadProcessorTest {

   static testMethod void testMethod1(){
      Test.startTest();
      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;

      DailyLeadProcessor ab = new DailyLeadProcessor(); String
      jobId = System.schedule('jobName', '0 5 * * * ?',ab);

      Test.stopTest();
   }
}
```

Apex Integration Services

Apex REST Callouts

AnimalLocator

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
       Http http = new Http();
       HttpRequest req = new HttpRequest();
       req.setEndpoint('https:/ th-apex-http-callout.herokuapp.com/animals/' + x);
       req.setMethod('GET');
       Map<String, Object> animal= new Map<String, Object>();
       HttpResponse res = http.send(req);
          if (res.getStatusCode() == 200) {
       Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
     animal = (Map<String, Object>) results.get('animal');
       }
return (String)animal.get('name');
   }
}
```

AnimalLocatorTest

```
@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 );
   }
}
```

AnimalLocatorMock

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

Apex Web Services

AccountManager

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

AccountManagerTest

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri = 'https:/ na1.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 TestAcc = new Account(
        Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId =
        TestAcc.id); return
        TestAcc.Id;
    }
}
```

Apex Specialist Superbadge

MaintenanceRequest

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

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;
      }
   }
}
MaintenanceRequestHelperTest

@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);
    }
}
```

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);
            }

        }
    }
}
```

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https:/ th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        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;
    }
}
```

WarehouseCalloutServiceTest

```
@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]);
   }
}
```

WarehouseSyncSchedule

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {
      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

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();
      CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
      System.assertEquals(jobID, a.Id,'Schedule ');
   }
}
```