Apex Triggers:

Get Started with Apex Triggers:

Account Address Trigger:

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

Bulk Apex Trigger:

```
trigger ClosedOpportunityTrigger on Opportunity (before 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 test:

1. **verifyData** :-

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

```
  //check for date2 being in the past
        if( date2 < date1) { return false; }
        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1
  if( date2 >= date30Days ) { return false; }
  else { return true; }
 }
 //method to return the end of the month of a given date
 private static Date SetEndOfMonthDate(Date date1) {
  Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
  Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
  return lastDay;
 }
}
```

2.**TestVerifyDate :-**

```
@isTest
public class TestVerifyDate
{
  static testMethod void testMethod1()
  {
     Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
     Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
  }
}
```

Test 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');
                }
        }
}
```

TestRestrictContactByName:-

```
@IsTest
public class TestRestrictContactByName {
  @IsTest static void createBadContact(){
     Contact c=new Contact(Firstname='John',LastName='INVALIDNAME');
     Test.startTest();
```

```
        Database.SaveResult result = Database.insert(c, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
    }
}
```

Create Test Data For Apex Tests:

RandomContactFactory:-

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numofContacts, String lastName) {
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numofContacts;i++) {
            Contact c = new Contact(FirstName='Test ' + i,LastName=lastName);
            contacts.add(c);
        }
        System.debug(contacts);
        return contacts;
    }
}
```

Asynchronous Apex:

Use Future Methods:

AccountProcessor:-

```
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];
        // process account records to do awesome stuff
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

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:-

```
public class LeadProcessor implements
   Database.Batchable<sObject> {
   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){
    }
}
```

LeadProcessorTest:-

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 10 accounts
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i, Company='Test Co'));
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(MyLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

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
private class DailyLeadProcessorTest {
    // Dummy CRON expression: midnight on March 15.
    // Because this is a test, job executes
    // immediately after Test.stopTest().
    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
    static testmethod void testScheduledJob() {
        // Create some out of date Opportunity records
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<200; i++) {
            Lead l = new Lead(
                FirstName = 'First ' + i,
                LastName = 'Lastname',
                Company = 'The Inc'
            );
            leads.add(l);
        }
        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('ScheduledApexTest',
            CRON_EXP,
            new DailyLeadProcessor());

        Test.stopTest();
        // Now that the scheduled job has executed,
        // check that we have 200 Leads with dreamforce
        List<Lead> checkleads = new List<Lead>();
        checkleads = [SELECT Id
            FROM Lead
            WHERE LeadSource='Dreamforce' and Company =  'The Inc'];
        System.assertEquals(200,
            checkleads.size(),
            'Leads were not created');
    }
}
```

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 {
    // 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;
   }
```

```
}
```

Apex SOAP Callouts:

ParkLocator:-

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

ParkLocatorTest:-

```
public class ParkLocatorTest {
    @isTest static void testCallout(){
        test.setMock(WebServiceMock.class, new ParkServiceMock());
        string country = 'United states';
        List<string> results = ParkLocator.country(country);
         List<string> parks =new list<string>{'Yellowstone','Mackinac National Park', 'Yosemite'};
         system.assertEquals(parks,results);
    }
}
```

ParkLocatorMock:-

```
@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 responseNam,
        string responseType
    ) {
        ParkService.byCountryResponse response_x = new  ParkService.byCountryResponse();
    response_x.return_x = new list<string> {'Yellowstone','Mackinac National Park', 'Yosemite'};
        response.put('response_x', response_x);
        }
}
```

Apex Web Service:

AccountManager:-

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

AccountManagerTest:-

```
@isTest
private class AccountManagerTest {
  @isTest static void testgetContactsByAccountId() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
              + recordId;
    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);
  }
  // Helper method
    static Id createTestRecord() {
    // Create test record
    Account accountTest = new Account(
      Name='Test record');
    insert accountTest;
    Contact contactTest;
    FirstName='John';
    LastName='Doe';
    AccountId=accountTest.Id);
```

```
    insert contactTest;
    return accountTest.Id;
  }
}
```

SUPER BADGE:

Apex Specialist:

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));
            } 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;
    }
  }
}
```

MaintenanceRequestHelperTest:-

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

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();
  }
}
```

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]);
    }
}
```

WarehouseCalloutServiceTestMock:-

```
@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":"Gen
erator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseSyncSchedule:-

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

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 ');
    }
}
```