Salesforce
Developer Catalyst

Project Document by

Shaik Ahmed Razha Khan

B.Tech in Computer Science and Engineering

ANITS, Visakhapatnam

What Is Salesforce?

Salesforce is your customer success platform, designed to help you sell, service, market, analyze, and connect with your customers.
Salesforce has everything you need to run your business from anywhere. Using standard products and features, you can manage relationships with prospects and customers, collaborate and engage with employees and partners, and store your data securely in the cloud.

Relationships & Process Automation

What are relationships in Salesforce?
Relationship in Salesforce is a 2-way association between 2 objects. Using relationships we can link objects with each other and we can make connections and display data about other related objects.
Automation tools are one of the declarative tools in Salesforce that make it more powerful by automating the number of administrative tasks that sales representatives and their managers have to perform manually. Salesforce provides multiple automation tools to automate your organization's repetitive business processes: Workflow, Process Builder, Flow Builder, Approvals. Each tool comes with its own unique features. The best automation tool for your needs depends on the type of business process that you're automating.

Workflow
Workflows are used to automate basic processes which can be summarized with a single if/then statement which means that if certain conditions are met, then the corresponding actions will be executed otherwise it will not be executed. These actions can be either Immediate Action that happens immediately or Time-Dependent Action that will execute at a specific time. In the case of Time-Dependent Action, criteria are evaluated again at the specified time before performing an action and if the criteria are found to be false, the Time-Dependent Action will not execute otherwise it will execute. The actions that can be performed through workflow are:

- Field Update
- Email Alert
- Send outbound message
- Create Task

Process Builder

Process Builder is the latest Salesforce automation tool that helps you to automate almost all of your business processes in the background, you don't have any interaction with the process you just see the results when they are finished. It includes all the functionality of workflows except sending outbound messages without Apex code. It can handle multiple if/then statements and can perform different-different actions based on the conditional statements in a single process that would take multiple Workflows. It can be triggered when a record is created or updated, by

another process or when a platform event occurs. It can be used to create new records, new tasks, send email alerts, post messages to chatter, submit records for approval, update any related records and call Apex code. Process Builder cannot be used to:

- send outbound messages without Apex code
- delete records.

Approval

Approval Process in Salesforce automates the necessary steps required for a record to be approved and specifies the approver to approve it at each step. It specifies what activities will happen when submitting a record for endorsement, when an approver or all approvers approve the record, and furthermore when an approver rejects a record. The activities allowed through the Approval Process are Field Update, Email Alert, Create Task, and Outbound Message. Whenever a user request approval, Initial Submission Actions occur in which the record is locked so that other user can't change the record while approval is pending. If the request is rejected by the approver, the Final Rejection Actions are executed, and approval status is set as Rejected.

Flows & Security

Salesforce Flow provides declarative process automation for every Salesforce app, experience, and portal. Included in Salesforce Flow are two point-and-click automation tools: Flow Builder, which lets you build flows, and Process Builder, which lets you edit existing processes.

Salesforce Data security deals with the security or sharing settings of data and visibility between users or groups of users across the organization. Force.com platform provides a flexible, layered sharing model that makes it easy to assign different data sets to different sets of users.

Apex, Testing And Debugging

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Lightning platform server in conjunction with calls to the Lightning Platform? API. ... Apex code can be initiated by Web service requests and from triggers on objects.

The Apex testing framework enables you to write and execute tests for your Apex classes and triggers on the Lightning Platform. Apex unit tests ensure high quality for your Apex code and let you meet requirements for deploying Apex.
Testing is the key to successful long-term development and is a critical component of the development process. The Apex testing framework makes it easy to test your Apex code. Apex code can only be written in a sandbox environment or a Developer org, not in production. Apex code can be deployed to a production org from a sandbox. Also, app developers can distribute Apex code to customers from their Developer orgs by uploading packages to the Lightning

Platform AppExchange. In addition to being critical for quality assurance, Apex unit tests are also requirements for deploying and distributing Apex. The following are the benefits of Apex unit tests.

- Ensuring that your Apex classes and triggers work as expected
- Having a suite of regression tests that can be rerun every time classes and triggers are updated to ensure that future updates you make to your app don't break existing functionality
- Meeting the code coverage requirements for deploying Apex to production or distributing Apex to customers via packages
- High-quality apps delivered to the production org, which makes production users more productive
- High-quality apps delivered to package subscribers, which increase your customers trust

Integration

When we talk about integration, it means to create a connection between a specific Salesforce instance and another database or system. The connection could be inbound, outbound, or bi-directional, and you may be connecting to another database, another Salesforce instance, or another cloud-based data source

Apex Triggers

Apex triggers enable you to perform custom actions before or after events to records in Salesforce, such as insertions, updates, or deletions. Just like database systems support triggers, Apex provides trigger support for managing records.
Typically, you use triggers to perform operations based on specific conditions, to modify related records or restrict certain operations from happening. You can use triggers to do anything you can do in Apex, including executing SOQL and DML or calling custom Apex methods.
Use triggers to perform tasks that can't be done by using the point-and-click tools in the Salesforce user interface. For example, if validating a field value or updating a field on a record, use validation rules and workflow rules instead.
Triggers can be defined for top-level standard objects, such as Account or Contact, custom objects, and some standard child objects. Triggers are active by default when created. Salesforce automatically fires active triggers when the specified database events occur.

Apex Testing

The Apex testing framework enables you to write and execute tests for your Apex classes and triggers on the Lightning Platform. Apex unit tests ensure high quality for your Apex code and let you meet requirements for deploying Apex.
Testing is the key to successful long-term development and is a critical component of the development process. The Apex testing framework makes it easy to test your Apex code. Apex code can only be written in a sandbox environment or a Developer org, not in production. Apex

Code can be deployed to a production org from a sandbox. Also, app developers can distribute Apex code to customers from their Developer orgs by uploading packages to the Lightning Platform AppExchange. In addition to being critical for quality assurance, Apex unit tests are also requirements for deploying and distributing Apex. The following are the benefits of Apex unit tests.

- Ensuring that your Apex classes and triggers work as expected
- Having a suite of regression tests that can be rerun every time classes and triggers are updated to ensure that future updates you make to your app don't break existing functionality
- Meeting the code coverage requirements for deploying Apex to production or distributing Apex to customers via packages
- High-quality apps delivered to the production org, which makes production users more productive
- High-quality apps delivered to package subscribers, which increase your customers' trust

Asynchronous Apex

In a nutshell, asynchronous Apex is used to run processes in a separate thread, at a later time. An asynchronous process is a process or function that executes a task "in the background" without the user having to wait for the task to finish.
Here's a real-world example. Let's say you have a list of things to accomplish before your weekly Dance Revolution practice. Your car is making a funny noise, you need a different color hair gel and you have to pick up your uniform from your mom's house. You could take your car to the mechanic and wait until it is fixed before completing the rest of your list (synchronous processing), or you could leave it there and get your other things done, and have the shop call you when it's fixed (asynchronous processing). If you want to be home in time to iron your spandex before practice, asynchronous processing allows you to get more stuff done in the same amount of time without needless waiting.
You'll typically use Asynchronous Apex for callouts to external systems, operations that require higher limits, and code that needs to run at a certain time.

Apex Integration Services

An Apex callout enables you to tightly integrate your Apex code with an external service. The callout makes a call to an external web service or sends an HTTP request from Apex code, and then receives the response.
Apex callouts come in two flavors.

- Web service callouts to SOAP web services use XML, and typically require a WSDL document for code generation.
- HTTP callouts to services typically use REST with JSON.

These two types of callouts are similar in terms of sending a request to a service and receiving a response. But while WSDL-based callouts apply to SOAP Web services, HTTP callouts can be used with any HTTP service, either SOAP or REST.

So you are probably asking yourself right now, “Which one should I use?” Whenever possible, use an HTTP service. These services are typically easier to interact with, require much less code, and utilize easily readable JSON. All the “cool kids” have been switching to REST services over the last couple of years, but that’s not to say that SOAP Web services are bad. They’ve been around forever (in Internet years) and are commonly used for enterprise applications. They are not going away anytime soon. You’ll probably use SOAP mostly when integrating with legacy applications or for transactions that require a formal exchange format or stateful operations.

Formulas And Validations

Introduction to Formula Fields

You’ve got a lot of data in your organization. Your users need to access and understand this data at a glance without doing a bunch of calculations in their heads. Enter formula fields, the powerful tool that gives you control of how your data is displayed.
Let’s say you wanted to take two numeric fields on a record and divide them to create a percentage. Or perhaps you want to turn a field into a clickable hyperlink for easy access to important information from a record’s page layout. Maybe you want to take two dates and calculate the number of days between them. All these things and more are possible using formula fields.

Salesforce Flow

In Salesforce, a flow is an application that automates complex business processes. Simply put, it collects data and then does something with that data. Flow Builder is the declarative interface used to build individual flows. Flow Builder can be used to build code-like logic without using a programming language.

Leads & Opportunities For Lightning Experience

Leads are people who are interested in your product and service. Converting leads to loyal customers will provide success within a business. By managing your leads in a systematic and structured way, you can increase both the numbers of leads you generate and how many leads you convert.

In Salesforce, an opportunity is a sale or pending deal. Multiple opportunities make up your sales pipeline, which contributes to your sales forecast. It's important to keep your Salesforce opportunities updated to ensure your sales forecast is accurate.

APEX CODES :

Apex Triggers

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert) {
        for(Account a:Trigger.New) {
      if(a.Match_Billing_Address__c==True) {
        a.ShippingPostalCode=a.BillingPostalCode;
      }
        }
}
```

ClosedOpportunityTrigger.apxt

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

Apex Testing

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
        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;
        }

}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {

   //testing that if date2 is within 30 days of date1, should return date 2
   @isTest static void testDate2within30daysofDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 04, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 04, 11);
      System.assertEquals(testDate,resultDate);
   }

   //testing that date2 is before date1. Should return "false"
   @isTest static void testDate2beforeDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 02, 11);
```

```
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }

    //Test date2 is outside 30 days of date1. Should return end of month.
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

TestRestrictContactByName .apxc

```
@isTest
private class TestRestrictContactByName {

    @isTest static void testInvalidName() {
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                            result.getErrors()[0].getMessage());
    }
}
```

RandomContactFactory.apxc

```
//@isTest
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
```

```
        Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
        contactList.add(c);
        System.debug(c);
      }
      //insert contactList;
      System.debug(contactList.size());
      return contactList;
   }

}
```

Asynchronous Apex

AccountProcessor.apxc

```
public class AccountProcessor {
@future
  public static void countContacts(Set<id> setId)
  {
     List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from contacts )
from account where id in :setId ];
     for( Account acc : lstAccount )
     {
        List<Contact> lstCont = acc.contacts ;

        acc.Number_of_Contacts__c = lstCont.size();
     }
     update lstAccount;
  }

}
```

AccountProcessorTest.apxc

```
@IsTest
public class AccountProcessorTest {
   public static testmethod void TestAccountProcessorTest()
   {
      Account a = new Account();
      a.Name = 'Test Account';
      Insert a;
```

```
        Contact cont = New Contact();
        cont.FirstName ='Bob';
        cont.LastName ='Masters';
        cont.AccountId = a.Id;
        Insert cont;

        set<Id> setAccId = new Set<ID>();
        setAccId.add(a.id);

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

        Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT 1];
        System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
    }

}
```

LeadProcessor.apxc

```
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

   // instance member to retain state across transactions
   global Integer recordsProcessed = 0;

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

   global 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';
            // increment the instance member counter
            recordsProcessed = recordsProcessed + 1;

      }
      update leads;
   }
```

```
    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');

    }
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }

    static testmethod void test() {
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
    }
}
```

AddPrimaryContact.apxc

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

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT
```

```
200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }

        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();

        System.assertEquals(50, [select count() from Contact]);
    }
}
```

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;
      }
   }
}
```

DailyLeadProcessorTest.apxc

```
@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

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');
  }
}
```

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;
  }
}
```

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 );
    }
}
```

ParkLocator.apxc

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

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);
    }
}
```

ParkService.apxc

```
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;
        }
    }
}
```

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);
  }
}
```

AccountManager.apxc

```
@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.apxc

```
@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);

    }

    // Helper method
        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;
    }
}
```