

**Configuration and Customization in Salesforce 2/28/2017**

**Legal Disclaimer**

This document contains unpublished, confidential and proprietary information of Excelacom Inc. that is protected by copyright and other intellectual property laws.

Excelacom Inc. software and its related documentation are subject to and made available only pursuant to the terms of a license agreement with Excelacom Inc. and may be used or copied only in accordance with the terms of that agreement.

Any use, copying, reproduction, alteration, transmission, or translation of these materials, in whole or in part, in any form or by any means, is strictly prohibited without the prior written permission of Excelacom Inc.

If this material is provided with software licensed by Excelacom Inc. the information herein is provided subject to the terms of the warranty provided with the product license. If this material is not provided with licensed software, the information herein is provided "as is" without warranty of any kind. In either case, there are no other warranties, either express or implied, including, but not limited to the implied warranties of merchantability, fitness for a particular purpose, no infringement, or quality. In no event shall Excelacom Inc. or any of its affiliates be liable for any direct or indirect, special, incidental, consequential, punitive, or exemplary damages arising out of the use or inability to use the materials and/or information contained herein. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Excelacom Inc. takes reasonable measures to ensure the quality of the data and other information produced by Excelacom Inc. that is made available herein. However, these materials may contain technical inaccuracies or typographical errors, and are not guaranteed to be error-free. Information may be changed or updated without notice. Excelacom Inc. has no obligation to update the information in this material based on changes to its products or services or those of third parties. Excelacom Inc. may also make improvements or changes to the products or services described in this information at any time without notice.

**Configuration and Customization in Salesforce:**

Configuration and Customization are terms you will hear thrown around in the Salesforce space frequently. Since the Salesforce platform is designed to handle those two things so well it happens at a really high percentage.

Configuration is anything you can do through the point and click environment of Salesforce. Using the declarative tools they provide you can configure the platform to do quite a lot for your business. Customization is anytime you need to start digging into code to accomplish your business objectives. This might mean that you have reached the limits of what can be done in the declarative tools and you need to use custom code to accomplish your objectives. Below are a few things that can be done within the Configuration and Customization categories.

|  |  |
| --- | --- |
| Configuration | Customization |
| * Adding Objects/Fields * Adding Tabs/Apps * Creating Profiles/Permission Sets * Workflows * Reports & Dashboards * Validation Rules * Email Templates * Page Layouts/Record Types * AppExchange Apps | * Apex Classes * Apex Triggers * Visualforce Pages/Components * Visualforce Emails * Integrations With 3rd Party Systems * Creating Lightning Components * Building Sites.com/Force.com Sites * Using CSS to alter the look of pages * Using JavaScript to add capabilities |

**Objects:**

Objects are a key element in Salesforce CRM as they provide a structure for storing data and are incorporated in the interface, allowing users to interact with the data.

Similar in nature to a database table, objects have properties such as:

* Fields which are similar in concept to a database column
* Records which are similar in concept to a database row
* Relationships to other objects
* Optional tabs which are user interface components to display the object data

### Standard objects

Salesforce provides standard objects in the application when you sign up and these include Account, Contact, Opportunity, and so on. These are the tables that contain the data records in any standard tab such as Accounts, Contacts, or Opportunities.

In addition to the standard objects, you can create custom objects and custom tabs.

### Custom objects

Custom objects are the tables you create to store your data. You can create a custom object to store data specific to your organization. Once you have the custom objects and have created records for these objects, you can also create reports and dashboards based on the record data in your custom object.

Objects can have relationship fields that define how records in one object relate to records in another object. These fields play the same role as primary and foreign keys in a database, but they’re more flexible, making it easier to design and implement your data model.

## **Fields:**

Fields in Salesforce are similar in concept to a database column and store the data for the object records. An object record is analogous to a row in a database table.

### Standard fields

Standard fields are predefined fields that are included as standard within the Salesforce CRM application. Standard fields cannot be deleted but non-required standard fields can be removed from page layouts whenever necessary.

With standard fields, you can customize visual elements that are associated to the field such as field labels and field-level help as well certain data definitions such as pick list values, the formatting of auto-number fields (which are used as unique identifiers for the records), and setting of field history tracking. Some aspects, however, such as the field name cannot be customized and some standard fields (such as Opportunity Probability) do not allow the changing of the field label.

### Custom fields

Custom fields are unique to your business needs and can not only be added and amended, but also deleted. Creating custom fields allow you to store the information that is necessary for your organization.

**Identity Field:**

Force.com automatically assigns an identity field (called ID) to every object, and manages the identity data in every record. This identity field usually comes in a 15-character case sensitive form, which you may have already seen while using the platform.

url : <https://ap2.salesforce.com/0062800000HgXJw>

Here, 0062800000HgXJw is the identifier for the record. Every record in your application will have such an identifier, and as a result you can view every record (across all objects) by simply using a URL of the above form. The platform will automatically retrieve the record and the associated metadata, and display that record using the automatically generated user interface.

Field Data Types:

**Auto Number:**

A system-generated sequence number that uses a display format you define. The number is automatically incremented for each new record.

**Formula:**

A read-only field that derives its value from a formula expression you define. The formula field is updated when any of the source fields change.

**Roll-Up Summary:**

A read-only field that displays the sum, minimum, or maximum value of a field in a related list or the record count of all records listed in a related list.

**Lookup Relationship:**

Creates a relationship that links this object to another object. The relationship field allows users to click on a lookup icon to select a value from a popup list. The other object is the source of the values in the list.

**External Lookup Relationship:**

Creates a relationship that links this object to an external object whose data is stored outside the Salesforce org.

**Checkbox:**

Allows users to select a True (checked) or false (unchecked) value.

**Currency:**

Allows users to enter a dollar or other currency amount and automatically formats the field as a currency amount. This can be useful if you export data to Excel or another spreadsheet.

**Date:**

Allows users to enter a date or pick a date from a popup calendar.

**Date/Time:**

Allows users to enter a date and time, or pick a date from a popup calendar. When users click a date in the popup, that date and the current time are entered into the Date/Time field.

**Email:**

Allows users to enter an email address, which is validated to ensure proper format. If this field is specified for a contact or lead, users can choose the address when clicking Send an Email. Note that custom email addresses cannot be used for mass emails.

**Geo location:**

Allows users to define locations. Includes latitude and longitude components, and can be used to calculate distance.

**Number:**

Allows users to enter any number. Leading zeros are removed.

**Percent:**

Allows users to enter a percentage number, for example, '10' and automatically adds the percent sign to the number.

**Phone:**

Allows users to select a value from a list you define.

**Pick list:**

Allows users to select multiple values from a list you define.

**Pick list (Multi-Select):**

Allows users to select multiple values from a list you define.

**Text:**

Allows users to enter any combination of letters and numbers.

**Text Area:**

Allows users to enter up to 255 characters on separate lines.

**Text Area (Long)**

Allows users to enter up to 131,072 characters on separate lines.

**Text Area (Rich)**

Allows users to enter formatted text, add images and links. Up to 131,072 characters on separate lines.

**Text (Encrypted)**

Allows users to enter any combination of letters and numbers and store them in encrypted form.

**URL**

Allows users to enter any valid website address. When users click on the field, the URL will open in aseparate browser window.

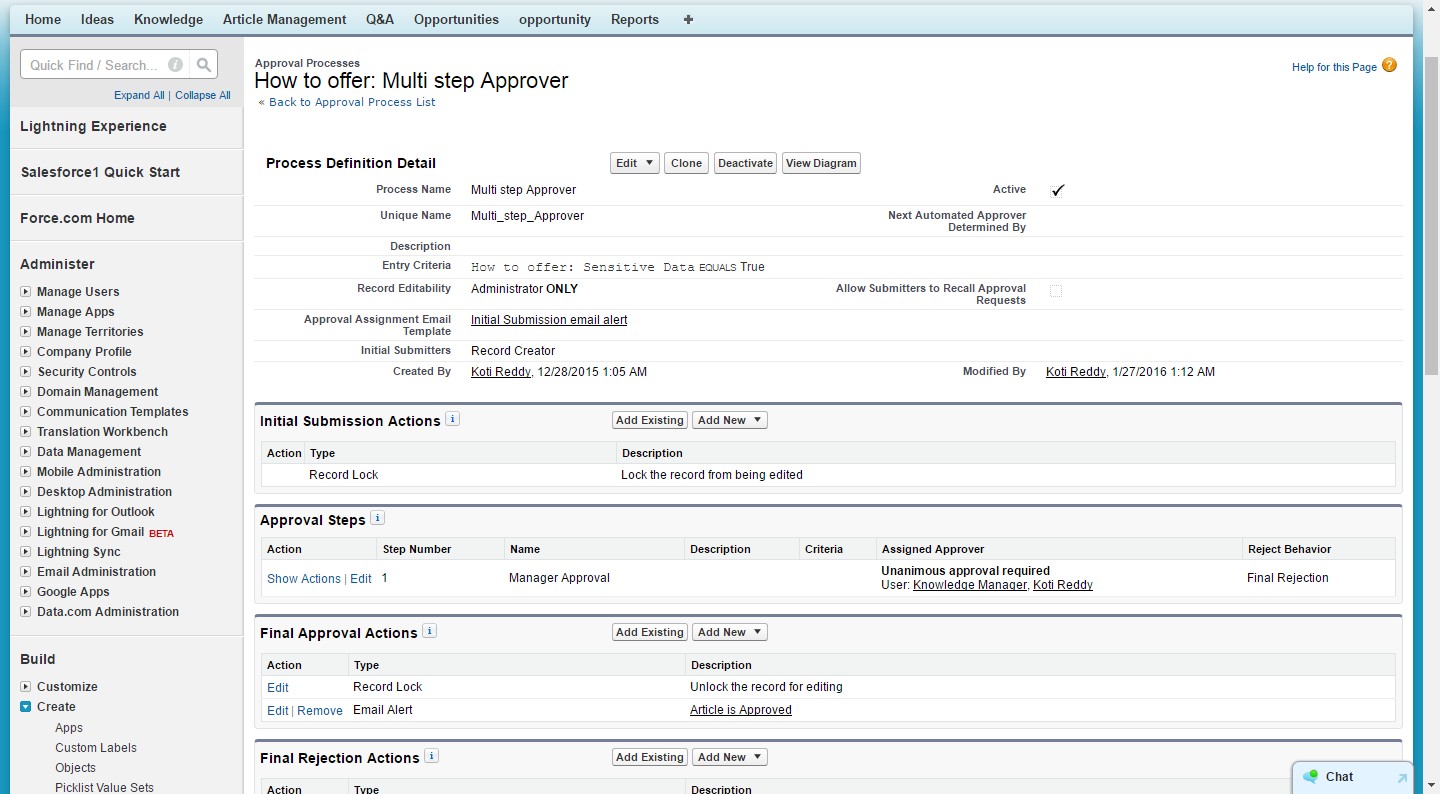
**Approval Process:**

An approval process is an automated process your organization can use to approve records in Salesforce.

An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. A step can apply to all records included in the process, or just records that have certain attributes.

An approval process also specifies the actions to take when a record is approved, rejected, recalled, or first submitted for approval.

This is the primary resource page for learning all about approval processes.



Simple or multiple –step approval processes to automate and enforce the approval of

Virtually anything in your company.

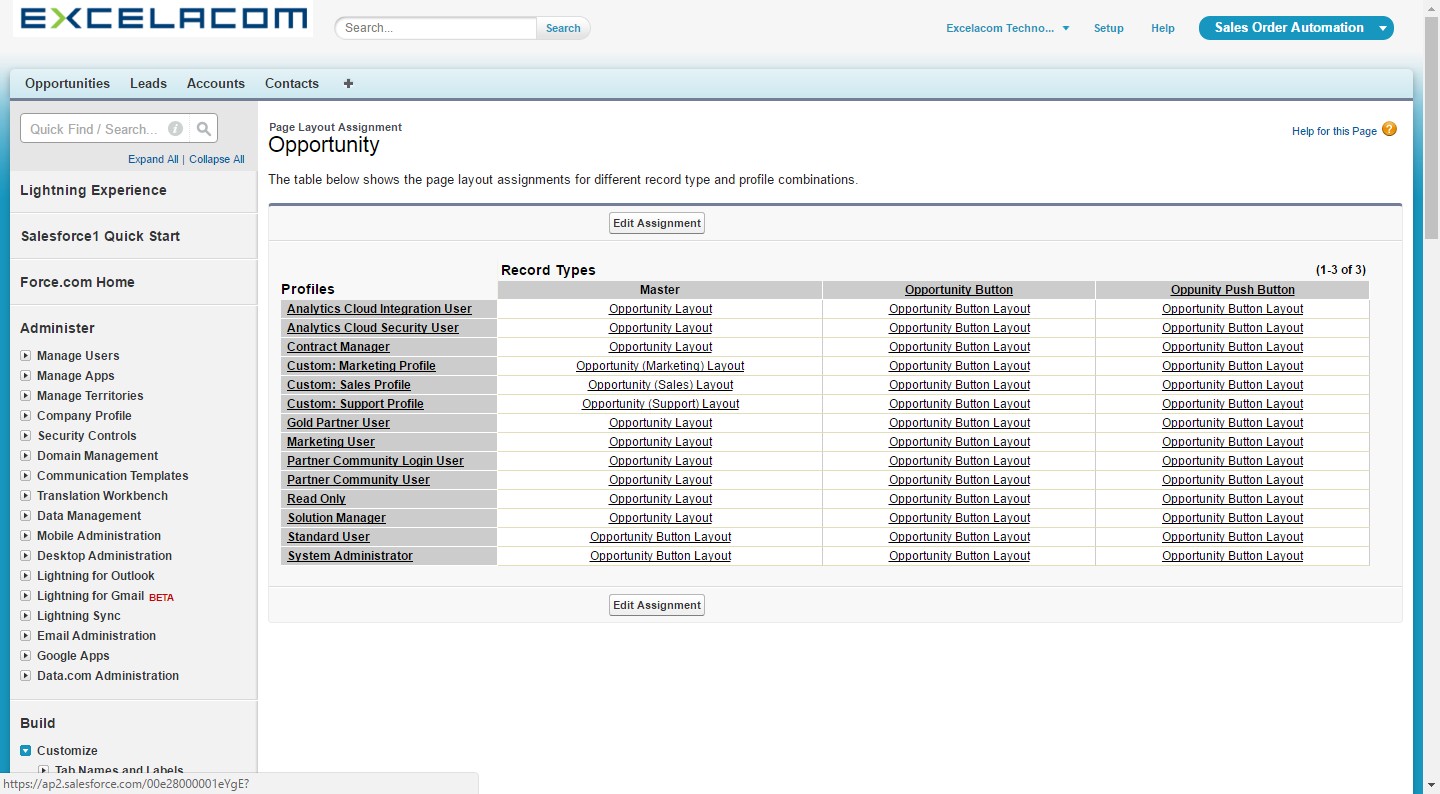
**Record Type:**

Record types allows administrators to configure different page layouts, business processes, and pick list values .It is important to note that record types are based on a user’s profile and not standard across the organization.

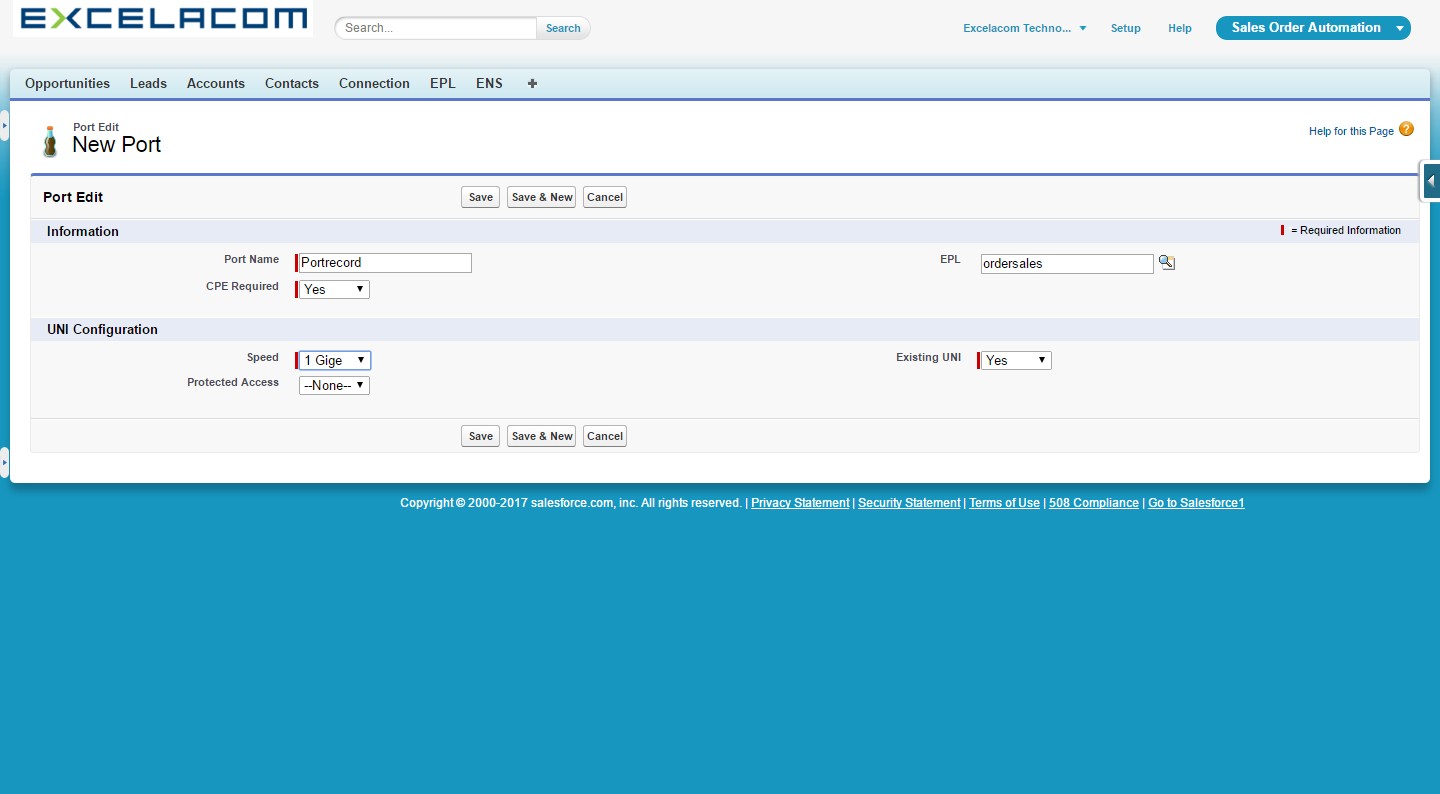
Record types can be used in a variety of ways, from displaying different page layouts for one object based on a set of criteria.

When you create a new record type without cloning an existing one, the new record type automatically includes the master pick list values for both standard and custom pick lists. You can then customize the pick list values for the record type. Enter a Record Type Label that's unique within the object.

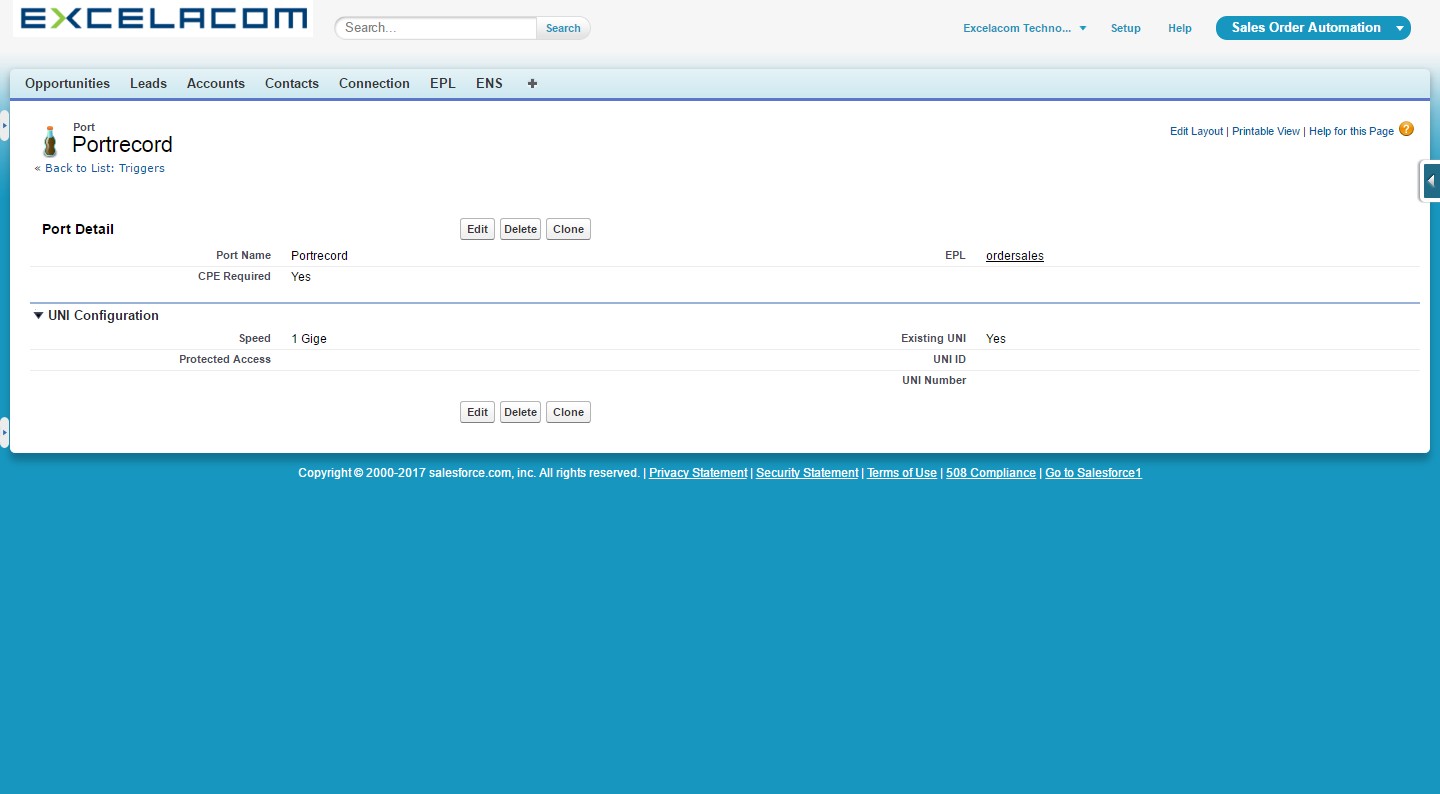
**Page Layout Assignment for Record type**



**Based on pick list value we can show different page layout for different record types:**



**Assigned page layout for Above Record type:**



**Name Field**

The name field of an object is a required field that has a special place in the life of the object. The record name is intended as a human-readable identifier for a record. It's not required to be a unique identifier, but it is supposed to be the primary way users distinguish one record from another. In the automatically generated user interfaces, you’ll see that the value for the name is always displayed as a link to the record itself, to its own detail page.

A name can be one of two types: a text string or an auto-number field. For an auto-number field, you must specify the format for the field and the starting number. Auto number field’s increment by one each time a record is created.

The Force.com database differs from relational databases in the way that record relationships are implemented. Instead of having to deal with primary keys and foreign keys, to define relationships between data, Force.com uses relationship fields. A relationship field stores the ID of the parent record in a relationship, as well as optionally providing user interface representations in both the parent and child records.

You can define two types of relationship fields.

• **Lookup**—this creates a relationship that links one object to another object. The relationship field allows you to navigate from records in one object to the related records in another object (both visually, and programmatically). Lookup relationships can be used to create one-to-one and one-to-many relationships.

• **Master-Detail**—this creates a special type of relationship between two objects (the child, or "detail") and another object (the parent, or "master"). Master-detail relationships can be used whenever there is a tight binding between two objects. For example, consider a blog and blog posts. If the blog is deleted, the blog posts should go too. They can also be used to create many-many relationships.

**Tabs /Apps:**

Custom tabs display custom object data or other web content embedded in the application.

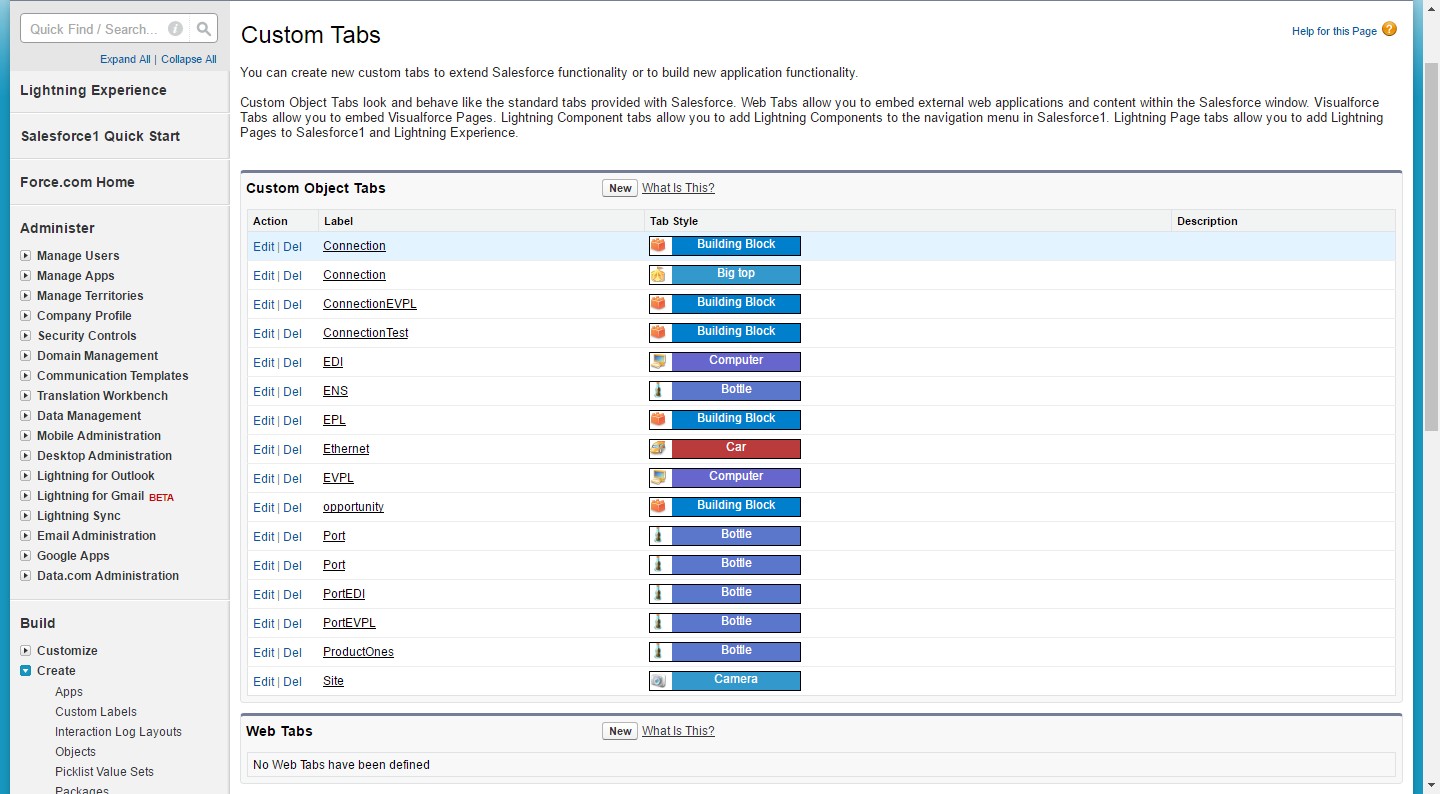
**Custom Object Tabs**: Custom Object Tabs display the data of your custom object in a user interface tab. Custom object tabs look and function just like standard tabs.

**Web Tabs:** Custom Web Tabs display any external Web-based application or Web page in a Salesforce tab. You can design Web tabs to include the sidebar or span across the entire page without the sidebar.

**Visualforce Tabs**: Visualforce Tabs display data from a Visualforce page. Visualforce tabs look and function just like standard tabs.

**Lightning Page tabs**: Add Lightning Pages to the Salesforce1 navigation menu.

**Custom Object Tabs:**

****

**Profiles and Permissions**:

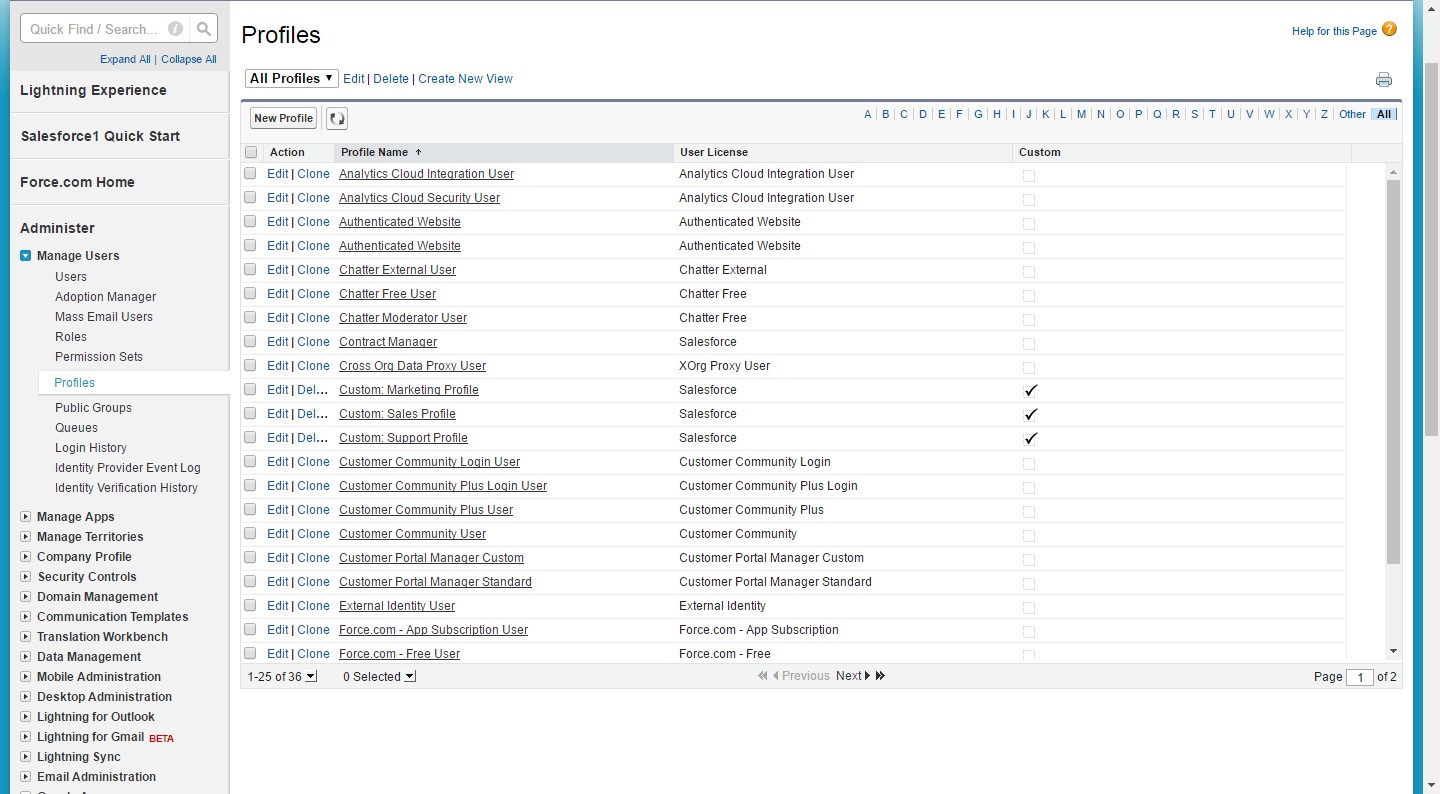
A profile is a group/collection of settings and permissions that define what a user can do in salesforce. A profile controls “Object permissions, Field permissions, User permissions, Tab settings, App settings, Apex class access, Visualforce page access, Page layouts, Record Types, Login hours & Login IP ranges.

**Types of Profiles in Salesforce**:

Standard Profile: By default Salesforce provides below standard profiles we can’t deleted standard ones. Read Only, Standard user, Marketing User, Contract Manager, Solution manager and System administration.

Custom Profile: custom Profile defined us they can be deleted there are no user assigned to particular ones.

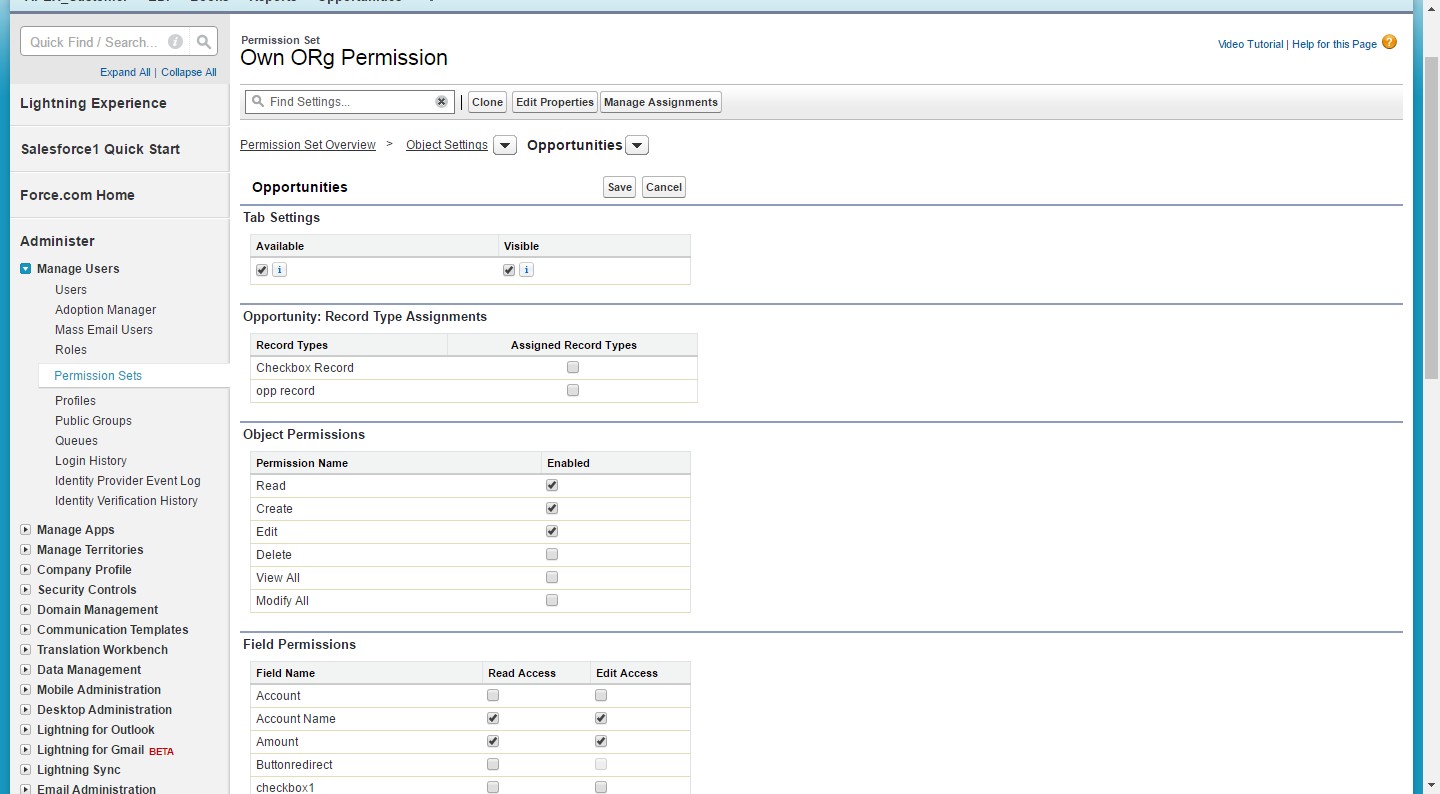
**Salesforce Standard Profiles**



**Permission sets:**

Permission sets is also very similar to profile. Whatever you can manage at profiles the same you can manage here also but main difference between these two is that users can have only one profile and can have multiple permission sets at time

So we can define profiles to grant minimum permissions and settings every type of user’s need that permission sets to grant additional access.



**Workflow:**

**Workflow** rules can help automate the following types of actions based on your organization's processes: Tasks: Assign a new task to a user, role, or record owner. Email Alerts: Send an email to one or more recipients you specify. Field Updates: Update the value of a field on a record.

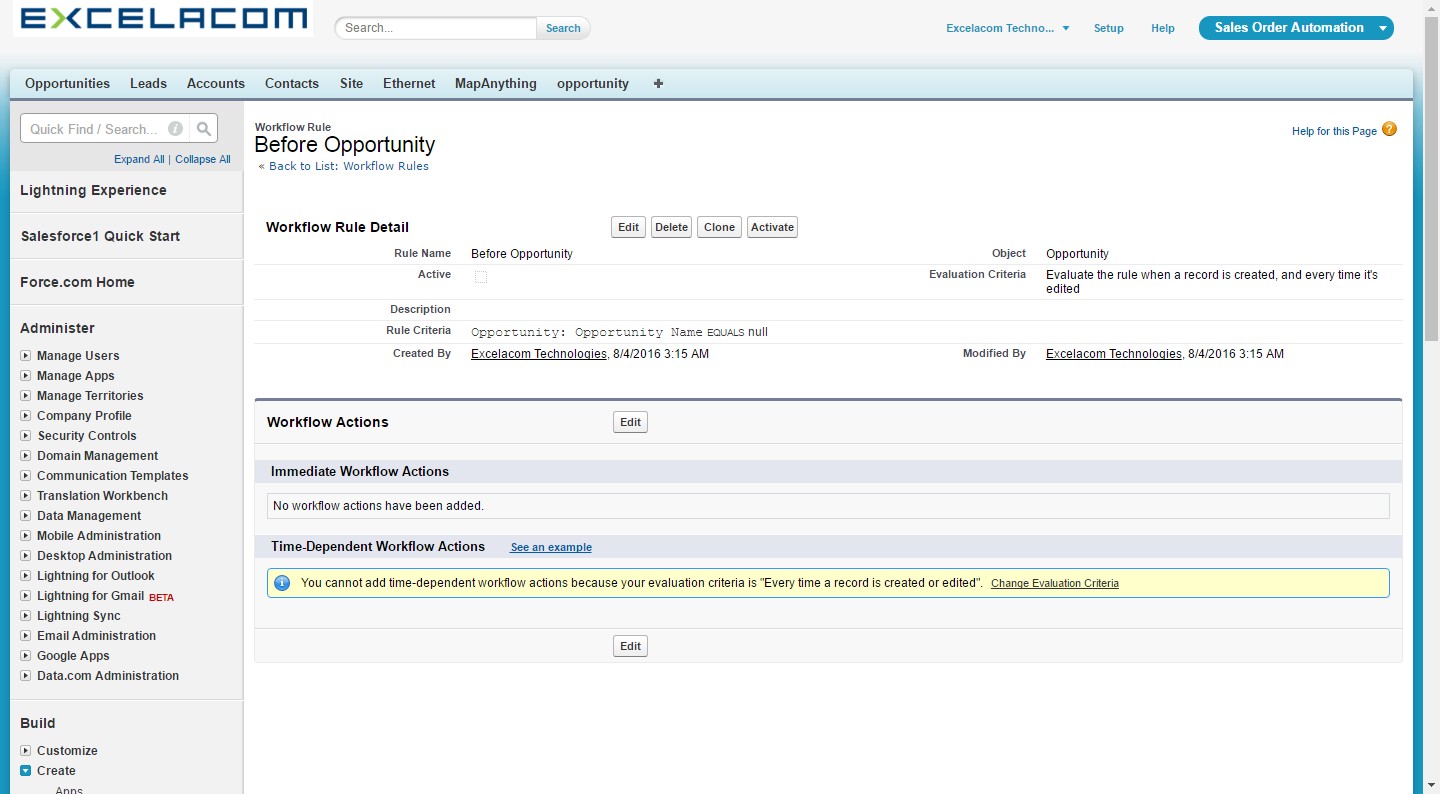
Workflow rules can help automate the following types of actions based on your organization's processes:

* Tasks: Assign a new task to a user, role, or record owner.
* Email Alerts: Send an email to one or more recipients you specify.
* Field Updates: Update the value of a field on a record.
* Outbound Messages: Send a secure, configurable API message (in XML format) to a designated listener.

**Each workflow rule consists of:**

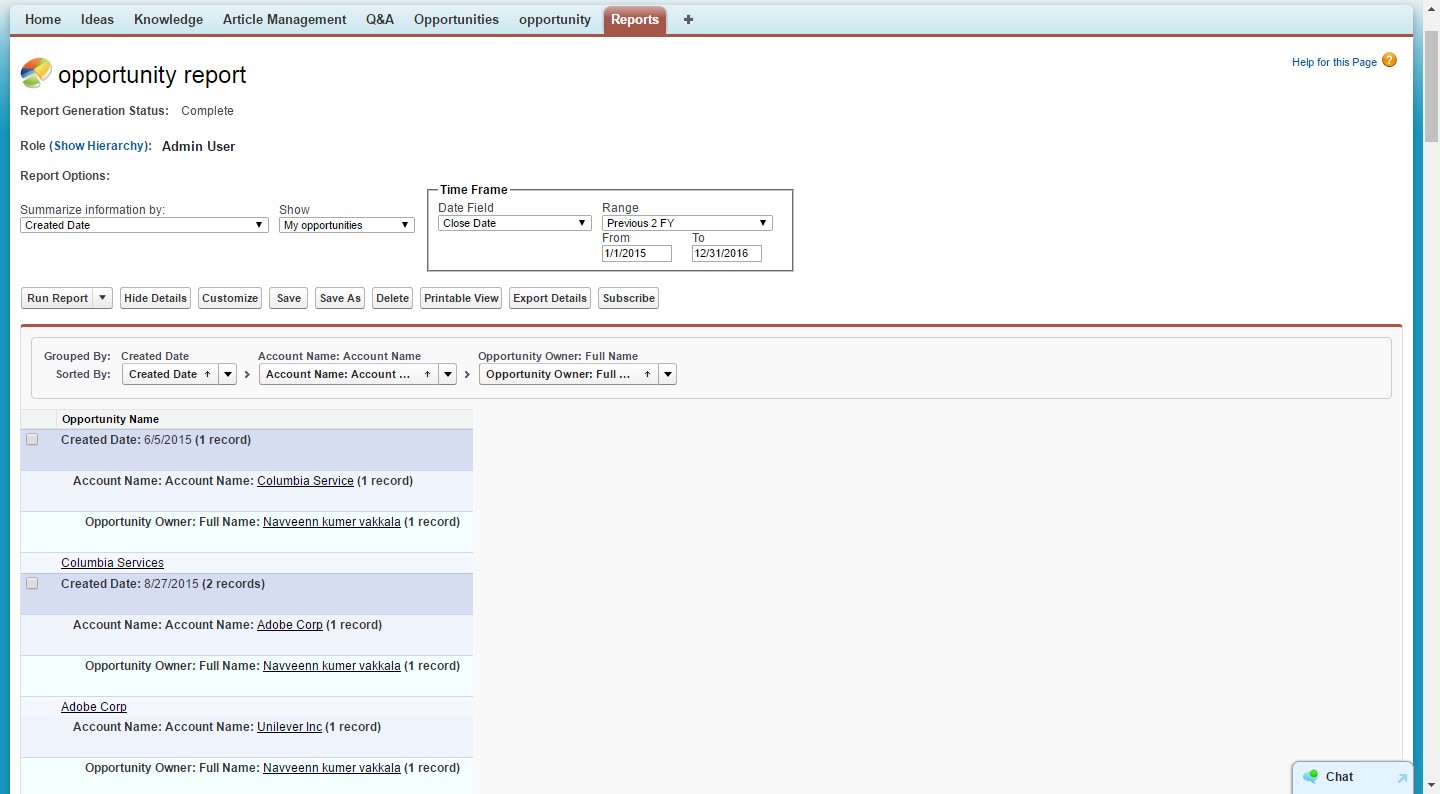
* Criteria that cause the workflow rule to run.
* Immediate actions that execute when a record matches the criteria. For example, salesforce.com can automatically send an email that notifies the account team when a new high-value opportunity is created.
* Time-dependent actions that queue when a record matches the criteria, and execute according to time triggers. For example, salesforce.com can automatically send an email reminder to the account team if a high-value opportunity is still open ten days before the close date.

**Salesforce Workflow Rule:**



**Reports:**

Reports and Dashboards. The wealth of information **Salesforce** puts at your fingertips—ranging from real-time snapshots of your business to yearly summaries.



Reports data is always generated in real time. When a report is saved, reports configuration parameters are stored - but the generated data is not stored.

**There are four type of reports**

* **Tabular report.**This is the most basic report. It displays just the row of records in a table like format with grand total. Tabular reports cannot be used for generating dashboards.
* **Summary report.**This is the most commonly type of report. It allows grouping of rows of data. It supports sorting and displaying subtotals. For example in a recruiting app, a summary report could be used to display open positions classified by department name.
* **Matrix report.**This is the most complex report format. Matrix report summarize information in a grid format. Matrix reports allows records to be grouped by both columns and rows.
* Summary and Matrix reports can be used to generate dashboards.
* **Joined Reports**

**Reports provide two options of exporting data into Excel.**

* Printable View - Export report with formatting into Excel
* Export Details - Export raw data
* Reports present in public folders can be emailed to Salesforce users.
* Report display up to 2000 rows of data. Larger reports can be emailed to Excel.

**Report Type:**

A report type is like a template which makes reporting easier. The report type determines which fields and records are available for use when creating a report. This is based on the relationships between a primary object and its related objects. For example, with the ‘Contacts and Accounts’ report type, ‘Contacts’ is the primary object and ‘Accounts’ is the related object.

Reports display only records that meet the criteria defined in the report type. Out of the box, Salesforce provides a set of predefined standard report types

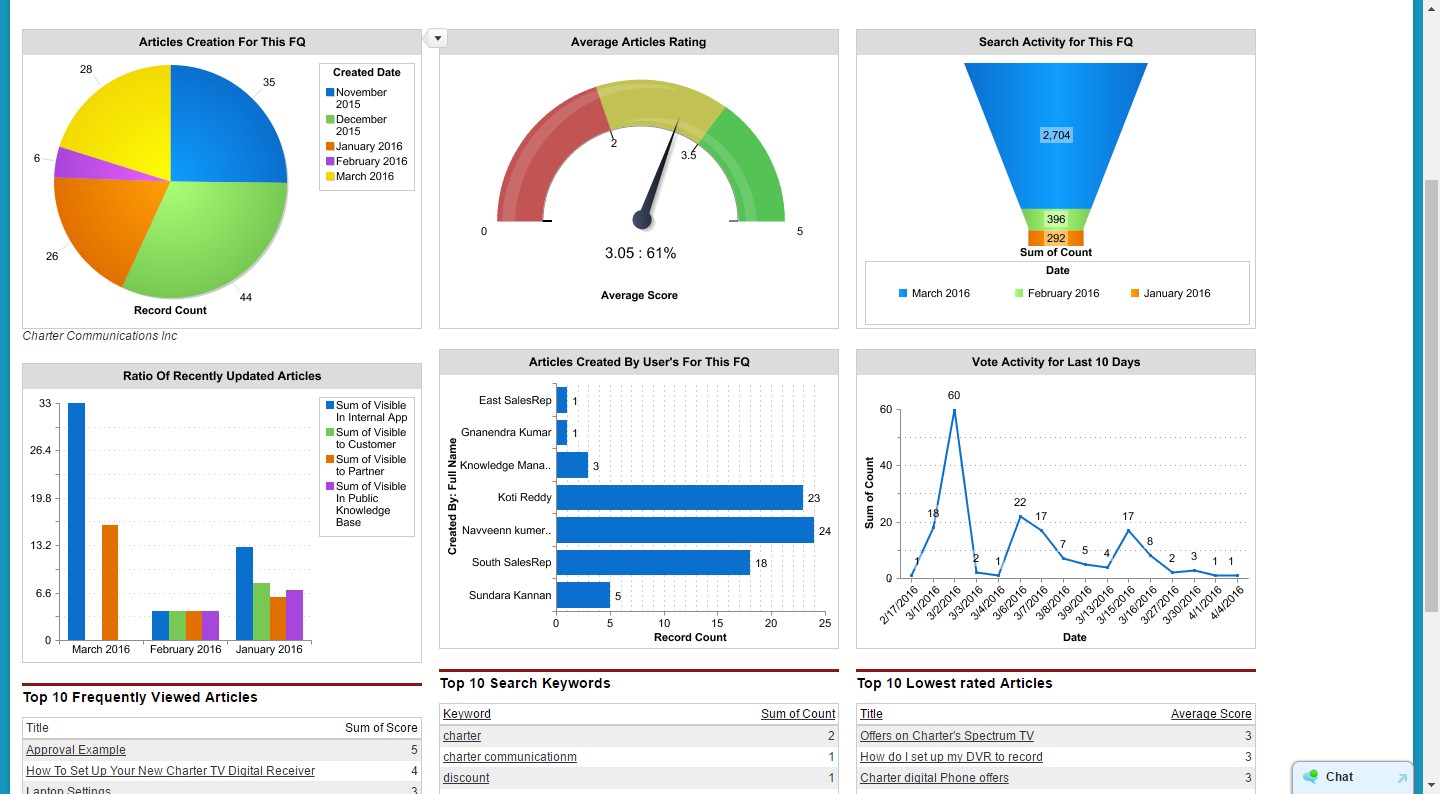
For example, an administrator can create a report type that shows only job applications that have an associated resume; applications without resumes won't show up in reports using that type. An administrator can also show records that may have related records—for example, applications with or without resumes. In this case, all applications, whether or not they have resumes, are available to reports using that type. An administrator can also add fields from a related object by creating a lookup relationship to that object, allowing for even more reporting possibilities.

**Dashboards:**

A dashboard is a visual display of key metrics and trends for records in your org. The relationship between a dashboard component and report is 1:1; for each dashboard component, there is a single underlying report. However, you can use the same report in multiple dashboard components on a single dashboard (e.g., use the same report in both a bar chart and pie chart).

Multiple dashboard components can be shown together on a single dashboard page layout, creating a powerful visual display and a way to consume multiple reports that often have a common theme, like sales performance, customer support, etc.

**Sample Dashboard (Pictorial representation of Report Data):**



Like reports, dashboards are stored in folders, which control who has access. If you have access to a folder, you can view its dashboards. However, to view the dashboard components, you need access to the underlying reports as well. You can also follow a dashboard in Chatter to get updates about the dashboard posted to your feed.

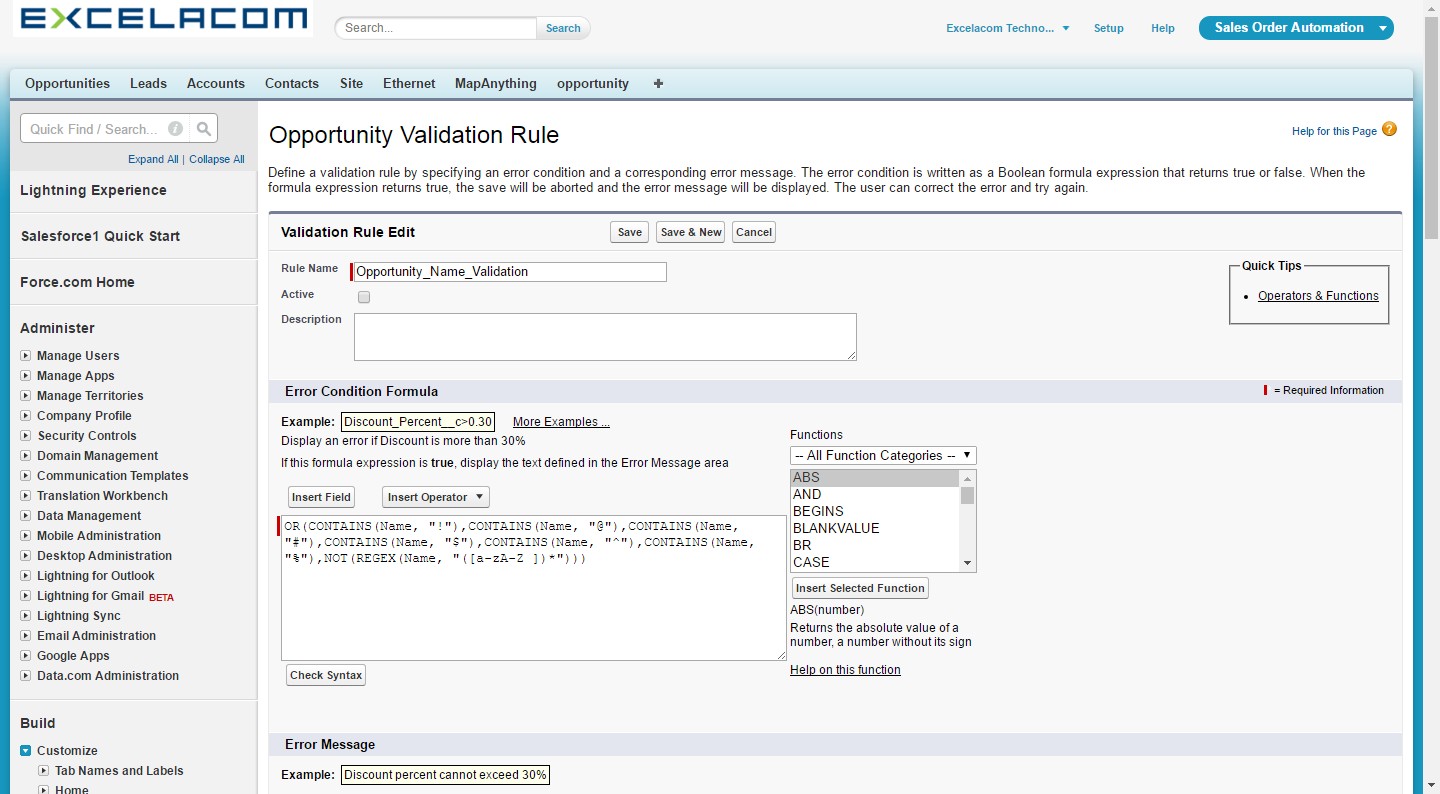
Each dashboard has a running user, whose security settings determine which data to display in a dashboard. If the running user is a specific user, all dashboard viewers see data based on the security settings of that user—regardless of their own personal security settings. For this reason, you’ll want to choose the running user wisely, so as not to open up too much visibility.

**Validation rules**:

**Validation rules** verify that the data a user enters in a record meets the standards you specify before the user can save the record. A **validation rule** can contain a formula or expression that evaluates the data in one or more fields and returns a value of “True” or “False”. ... All **validation rules** are verified.

You can specify the error message to display when a record fails validation and where to display it.

**Validation Rule:**



**Email Templates:**

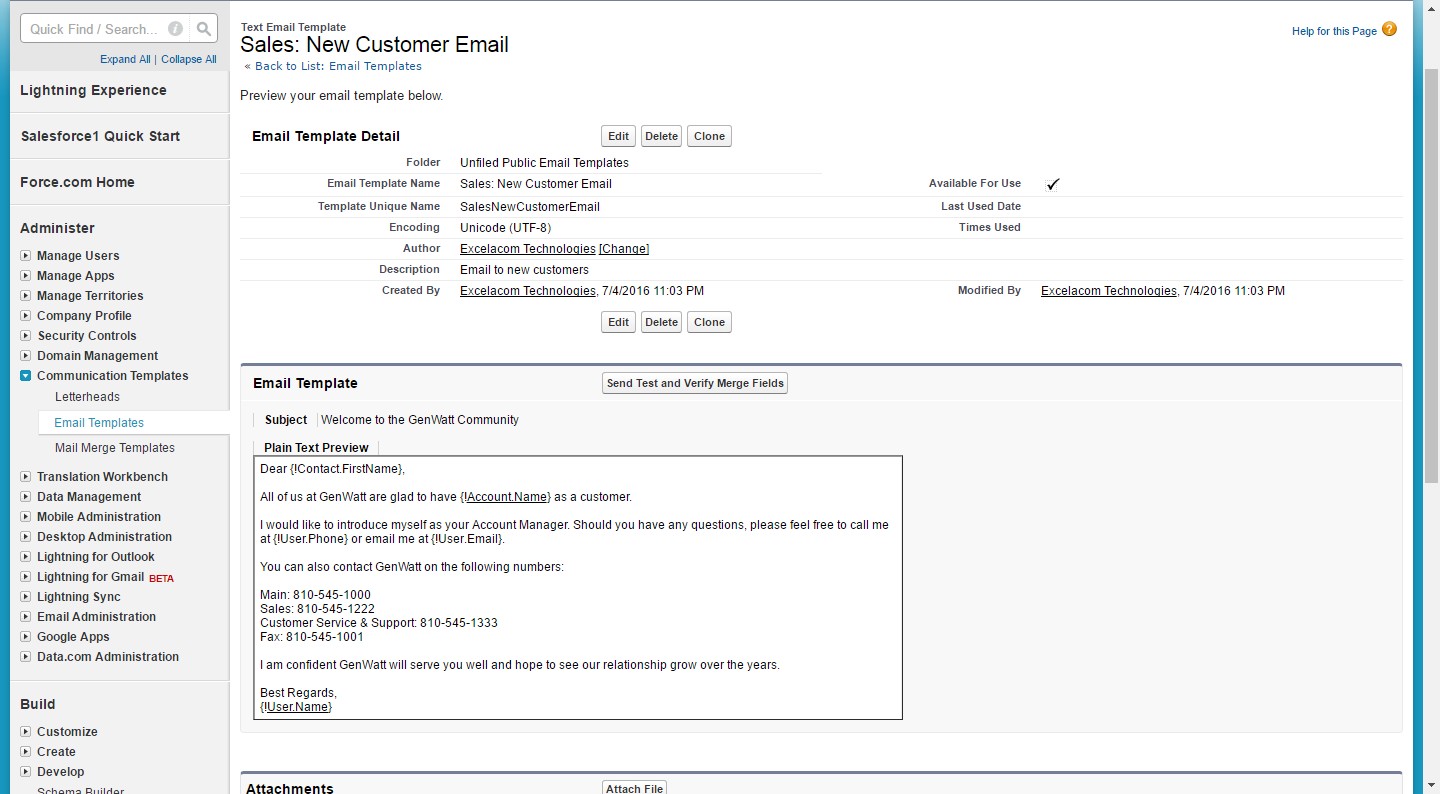
In Salesforce Classic, you can create four different types of email templates: text, HTML with letterhead, custom, and Visualforce. All of these email templates can include text, merge fields, and attached files. You can also include images on your HTML and Visualforce templates. You can use email templates when you send an email from the Activity History or HTML Email Status related list of a record.

Text and HTML templates can also be used when you send mass email. A button to check the spelling of your template is available for text templates and Visualforce templates without HTML tags.

**Text** - All users can create or change text email templates

**HTML with letterhead** - Administrators and users with the “Edit HTML Templates” permission can create HTML email templates based on a letterhead.

**Email Template:**

****

**Custom HTML** - Administrators and users with the “Edit HTML Templates” permission can create custom HTML email templates without using a letterhead. You must either know HTML or obtain the HTML code to insert in your email template.

**Visualforce** - Administrators and developers can create templates using Visualforce. Visualforce email templates allow for advanced merging with a recipient's data, where the content of a template can contain information from multiple records.

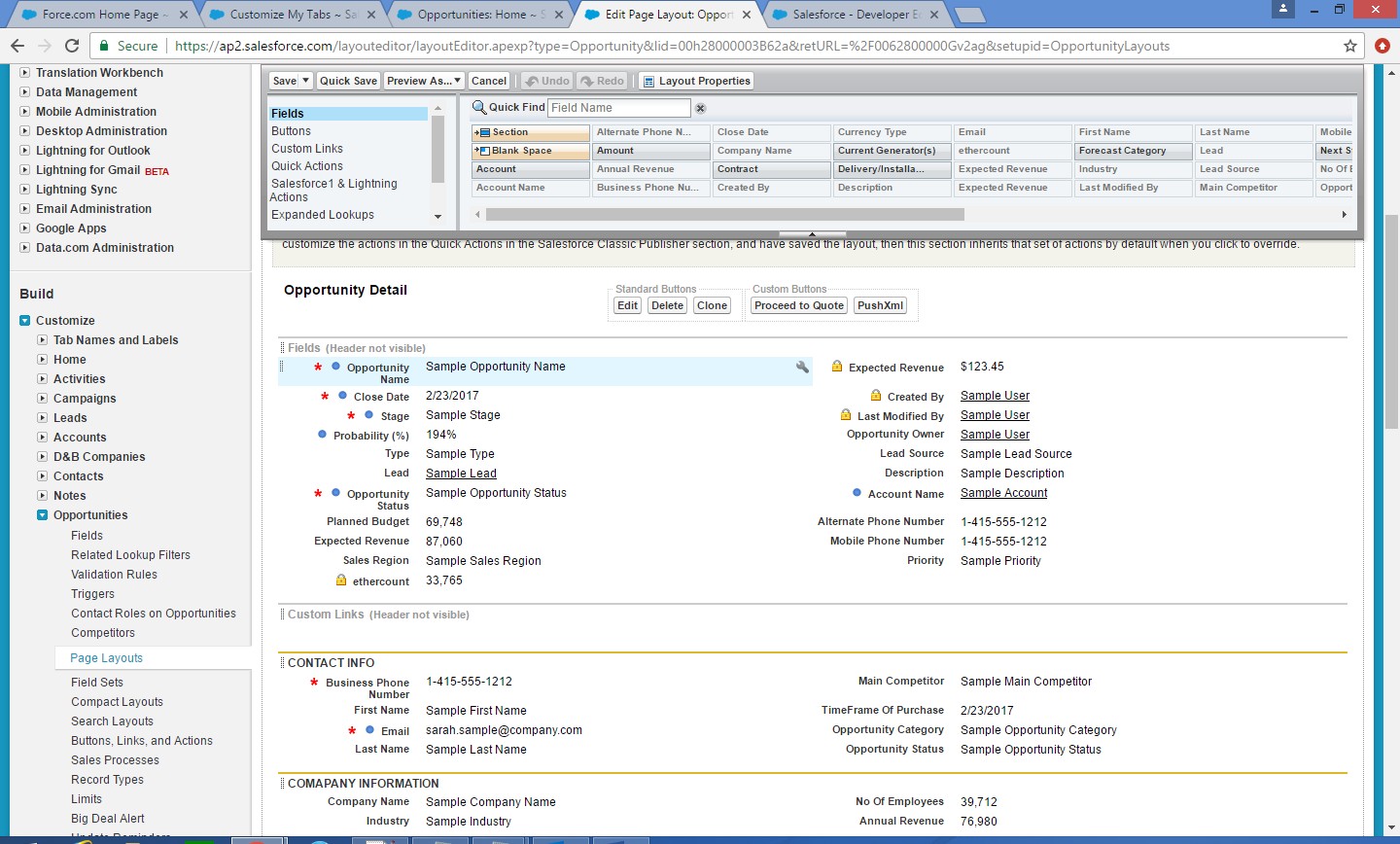
**Page Layout:**

In Salesforce**, Page Layouts** allow for customization of UI pages for users in your organization. And often times throughout Salesforce.com, you have pages of objects that might contain items that are not used by many users. Today, I'm going to explain how you can customize

A **Salesforce page layout with the fields, related lists, and custom links that suit your organization's use.**

In the example below, I'll change the layout of the page **Contracts.** Note that you can "Edit" the current page layout of that object, but it's best to create a new Contract Page Layout and then assign it the default one.

**Salesforce Standard Page Layout:**



Page Layout Assignment**:**

  Created the page layout, you can assign it to the appropriate user Profiles in your Salesforce org.

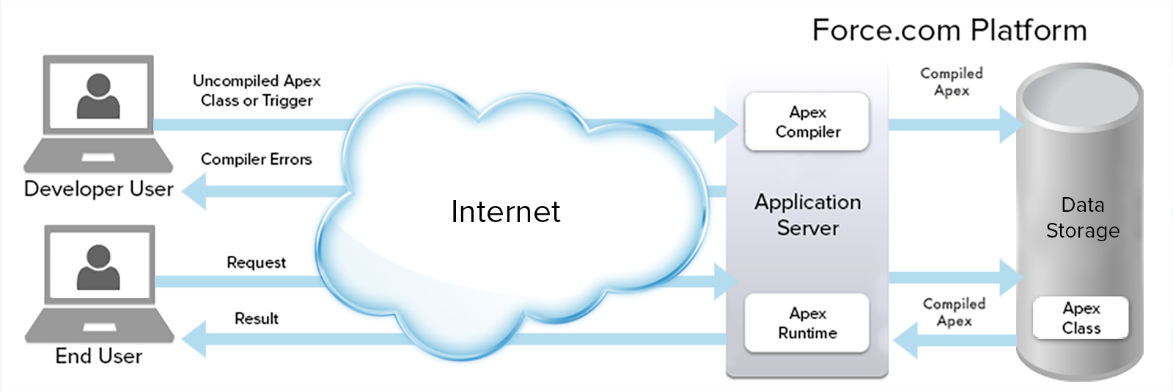
**Customizations in Salesforce:**

Apex Class:

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com​ API. Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

As a language, Apex is:

* Hosted— Apex is saved, compiled, and executed on the server—the Force.com platform.
* Automatically upgradeable— because compiled code is stored as metadata in the platform, Apex is automatically upgraded as part of Salesforce releases.
* Object oriented— Apex supports classes, interfaces, and inheritance.
* Strongly typed— Apex validates references to objects at compile time.
* Multitenant aware— Because Apex runs in a multitenant platform, it guards closely against runaway code by enforcing limits, which prevent code from monopolizing shared resources.
* Integrated with the database— It is straightforward to access and manipulate records. Apex provides direct access to records and their fields, and provides statements and query languages to manipulate those records.
* Data focused— Apex provides transactional access to the database, allowing you to roll back operations.



Apex Code :

public class MyHelloWorld {

public static void applyDiscount(Book\_\_c[] books) {

for (Book\_\_c b :books){

b.Price\_\_c \*= 0.9;

}

}

}

**Apex Language Highlights:**

Like other object-oriented programming languages, these are some of the language constructs that Apex supports:

* Classes, interfaces, properties, and collections (including arrays).
* Object and array notation.
* Expressions, variables, and constants.
* Conditional statements (if-then-else) and control flow statements (for loops and while loops).

Unlike other object-oriented programming languages, Apex supports:

* Cloud development as Apex is stored, compiled, and executed in the cloud.
* Triggers, which are similar to triggers in database systems.
* Database statements that allow you to make direct database calls and query languages to query and search data.
* Transactions and rollbacks.
* The global access modifier, which is more permissive than the public modifier and allows access across namespaces and applications.
* Versioning of custom code.

**Data Types Overview:**

Apex supports various data types, including a data type specific to Salesforce—the sObject data type.

Apex supports the following data types.

* A primitive, such as an Integer, Double, Long, Date, Date time, String, ID, Boolean,

Among others.

* An sObject, either as a generic sObject or as a specific sObject, such as an Account, Contact, or MyCustomObject\_\_c (you’ll learn more about sObjects in a later unit.)
* A list (or array) of primitives, sObjects, user defined objects, objects created from Apex classes, or collections

A set of primitives

* A map from a primitive to a primitive, sObject, or collection
* A typed list of values, also known as an enum
* User-defined Apex classes
* System-supplied Apex classes

**Apex Triggers:**

A trigger is Apex code that executes before or after specific data manipulation language (DML) events occur, such as before object records are inserted into the database, or after records have been deleted. Triggers are stored as metadata in Salesforce.

Trigger Sample Code :

trigger Customer\_After\_Insert on APEX\_Customer\_\_c (after update) {

List InvoiceList = new List();

for (APEX\_Customer\_\_c objCustomer: Trigger.new) {

if (objCustomer.APEX\_Customer\_Status\_\_c == 'Active') {

APEX\_Invoice\_\_c objInvoice = new APEX\_Invoice\_\_c();

objInvoice.APEX\_Status\_\_c = 'Pending';

InvoiceList.add(objInvoice);

}

}

//DML to insert the Invoice List in SFDC

insert InvoiceList;

}

Apex can be invoked by using *triggers*. Apex triggers enable you to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.

A trigger is Apex code that executes before or after the following types of operations:

* insert
* update
* delete
* merge
* Upsert
* undelete

For example, you can have a trigger run before an object's records are inserted into the database, after records have been deleted, or even after a record is restored from the Recycle Bin.

You can define triggers for top-level standard objects that support triggers, such as a Contact or an Account, some standard child objects, such as a Case Comment, and custom objects. To define a trigger, from the object management settings for the object whose triggers you want to access, go to Triggers.

There are two types of triggers:

* *Before triggers* are used to update or validate record values before they’re saved to the database.
* *After triggers* are used to access field values that are set by the system (such as a record's Id or LastModifiedDate field), and to affect changes in other records, such as logging into an audit table or firing asynchronous events with a queue. The records that fire the *after trigger* are read-only.

Triggers can also modify other records of the same type as the records that initially fired the trigger. For example, if a trigger fires after an update of contact *A*, the trigger can also modify contacts *B*, *C*, and *D*. Because triggers can cause other records to change, and because these changes can, in turn, fire more triggers, the Apex runtime engine considers all such operations a single unit of work and sets limits on the number of operations that can be performed to prevent infinite recursion.

Additionally, if you update or delete a record in its before trigger, or delete a record in its after trigger, you will receive a runtime error. This includes both direct and indirect operations. For example, if you update account A, and the before update trigger of account A inserts contact B, and the after insert trigger of contact B queries for account A and updates it using the DML update statement or database method, then you are indirectly updating account A in its before trigger, and you will receive a runtime error.

## **Implementation Considerations:**

Before creating triggers, consider the following:

* upsert triggers fire both before and after insert or before and after update triggers as appropriate.
* Merge triggers fire both before and after delete triggers for the losing records and before update triggers for the winning record only. See [Triggers and Merge Statements](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers_merge_statements.htm).
* Triggers that execute after a record has been undeleted only work with specific objects. See [Triggers and Recovered Records](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers_recovered_records.htm).
* Field history is not recorded until the end of a trigger. If you query field history in a trigger, you don’t see any history for the current transaction.
* Field history tracking honors the permissions of the current user. If the current user doesn't have permission to directly edit an object or field, but they activate a trigger that changes an object or field with history tracking enabled, no history of the change is recorded.
* Callouts must be made asynchronously from a trigger so that the trigger process isn’t blocked while waiting for the external service's response. The asynchronous callout is made in a background process, and the response is received when the external service returns it. To make an asynchronous callout, use asynchronous Apex such as a future method.
* In API version 20.0 and earlier, if a Bulk API request causes a trigger to fire, each chunk of 200 records for the trigger to process is split into chunks of 100 records. In Salesforce API version 21.0 and later, no further splits of API chunks occur. If a Bulk API request causes a trigger to fire multiple times for chunks of 200 records, governor limits are reset between these trigger invocations for the same HTTP request.

Bulk triggers can handle both single record updates and bulk operations like:

* Data import
* Force.com Bulk API calls.
* Mass actions, such as record owner changes and deletes
* Recursive Apex methods and triggers that invoke bulk DML statements

Trigger Events:

* before insert
* before update
* before delete
* after insert
* after update
* after delete
* after undelete

The code block of a trigger cannot contain the static keyword. Triggers can only contain keywords applicable to an inner class. In addition, you do not have to manually commit any database changes made by a trigger. If your Apex trigger completes successfully, any database changes are automatically committed. If your Apex trigger does not complete successfully, any changes made to the database are rolled back.

**Visual Force Pages:**

The Force.com platform offers separate tools for defining:

* The structure of the data—that is, the *data model*
* The rules that detail how that data can be manipulated—that is, the *business logic*
* The layouts that specify how that data should be displayed—that is, the *user interface*

While the tools for building the data model and business logic for applications are powerful solutions that run natively on Force.com platform servers, the existing tools for defining user interfaces have had certain limitations:

* **Page layouts**, the point-and-click tool that allows application developers to organize fields, buttons, and related lists on record detail pages, do not provide much flexibility in how sets of information are displayed. Fields must always appear above related lists, buttons must always appear above fields, and s-controls and custom links can only be placed in particular areas.

Example code:

<apex:page standardController="Account">

<apex:canvasApp applicationName="Test Inline Visualforce"

namespacePrefix="testorg"

height="400px" width="750px"/>

</apex:page>

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Force.com platform. The Visualforce framework includes a tag-based markup language, similar to HTML, and a set of server-side “standard controllers” that make basic database operations, such as queries and saves, very simple to perform.

In the Visualforce markup language, each Visualforce tag corresponds to a coarse or fine-grained user interface component, such as a section of a page, a related list, or a field. The behavior of Visualforce components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller class written in Apex.

Developers can use Visualforce to create a Visualforce page definition. A page definition consists of two primary elements:

* Visualforce markup
* A Visualforce controller

## **Visualforce Markup**

Visualforce markup consists of Visualforce tags, HTML, JavaScript, or any other Web-enabled code embedded within a single <apex:page> tag. The markup defines the user interface components that should be included on the page, and the way they should appear.

## **Visualforce Controllers**

A Visualforce controller is a set of instructions that specify what happens when a user interacts with the components specified in associated Visualforce markup, such as when a user clicks a button or link. Controllers also provide access to the data that should be displayed in a page, and can modify component behavior.

A developer can either use a standard controller provided by the Force.com platform, or add custom controller logic with a class written in Apex:

* A [standard controller](https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/pages_controller_std.htm) consists of the same functionality and logic that is used for a standard Salesforce page. For example, if you use the standard Accounts controller, clicking a **Save** button in a Visualforce page results in the same behavior as clicking **Save** on a standard Account edit page.

If you use a standard controller on a page and the user doesn't have access to the object, the page will display an insufficient privileges error message. You can avoid this by [checking the user's accessibility](https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/pages_controller_std_checking_accessibility.htm) for an object and displaying components appropriately.

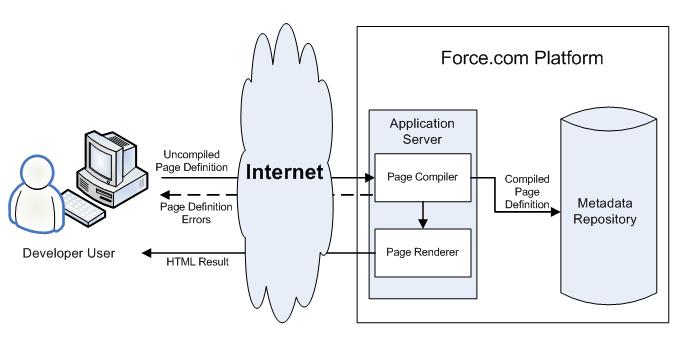
* A [standard list controller](https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/pages_controller_sosc_about.htm) enables you to create Visualforce pages that can display or act on a set of records. Examples of existing Salesforce pages that work with a set of records include list pages, related lists, and mass action pages.
* A custom controller is a class written in Apex that implements all of a page's logic, without leveraging a standard controller. If you use a custom controller, you can define new navigation elements or behaviors, but you must also implement any functionality that was already provided in a standard controller.

Like other Apex classes, custom controllers execute entirely in system mode, in which the object and field-level permissions of the current user are ignored. You can specify whether a user can execute methods in a custom controller based on the user's profile.

* A controller extension is a class written in Apex that adds to or overrides behavior in a standard or custom controller. Extensions allow you to leverage the functionality of another controller while adding your own custom logic.

Because standard controllers execute in user mode, in which the permissions, field-level security, and sharing rules of the current user are enforced, extending a standard controller allows you to build a Visualforce page that respects user permissions. Although the extension class executes in system mode, the standard controller executes in user mode. As with custom controllers, you can specify whether a user can execute methods in a controller extension based on the user's profile.

**Visualforce Architecture:**



**Visual Force Email Templates:**

Developers and administrators can use Visualforce to create email templates. The advantage of using Visualforce over standard HTML email templates is that Visualforce gives you the ability to perform advanced operations on data that is sent to a recipient.

Although Visualforce email templates use standard Visualforce components, they are not created in the same way. Visualforce email templates always use components that are prefaced with the messaging namespace. In addition:

* All Visualforce email templates must be contained within a single <messaging:emailTemplate> tag. This is analogous to regular Visualforce pages being defined within a single <apex:page> tag.
* The <messaging:emailTemplate> tag must contain either a single <messaging:htmlEmailBody> tag or a single <messaging:plainTextEmailBody> tag.
* Several standard Visualforce components are not available for use within <messaging:emailTemplate>. These include <apex:detail>, <apex:pageBlock> and all related pageBlock components, and all input components such as <apex:form>. If you attempt to save a Visualforce email template with these components, an error message displays.

**Integrating with third party system:**

Force.com provides a number of integration points ranging from off-the-shelf native ERP connectors to web services, email, syndication feeds and HTTP-based REST callouts. The platform also supports several APIs, including the Force.com SOAP and REST APIs, providing the basis for integration with other languages such as Java, .NET, Ruby, Objective-C and PHP.

## **Foundational Platform Integration Points**

The following sections examine the more fundamental integration building blocks:

* Creating and exposing web services using the Apex programming language
* Invoking external web services from Apex
* Outbound messaging for invoking external web services when data changes
* HTTP and REST integration
* Email integration for inbound and outbound messaging
* The Force.com SOAP APIs and associated toolkits, such as the Mobile SDK, AJAX Toolkit, Java, .NET, PHP and Adobe Flex integrations.
* Syndication feeds via Force.com Sites

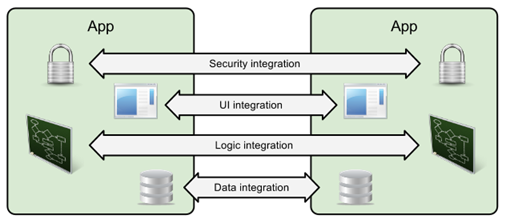
A few reasons that the external URLs would be secured are:

* The data obtained from them is not public information and only authorized clients should be able to access it e.g. someone’s personal details or things like bank account details.
* The URLs allow the data to be changed and only authorized clients should be allowed to make changes.
* There is a concern that the URLs might be overloaded with requests e.g. unexpected clients start to use the URLs because they available.

If none of these apply - the data is fine for anyone to see, the data is not updated, and the data is of no interest to others - you could leave the access unauthenticated.

If the access does need to be authenticated, the technique you use will depend on what is available on the server you access via the URLs - there are a variety of techniques

Most enterprise-level applications have a need to integrate with other applications used by the same organization. These integration usually cater to different layers, like Data, Business Logic, Presentation and Security, depending on the requirement. This helps organizations achieve greater levels of operational consistency, efficiency and quality.



**Batch Apex:**

A developer can now employ batch Apex to build complex, long-running processes that run on thousands of records on the Force.com platform. Batch Apex operates over small batches of records, covering your entire record set and breaking the processing down to manageable chunks. For example, a developer could build an archiving solution that runs on a nightly basis, looking for records past a certain date and adding them to an archive. Or a developer could build a data cleansing operation that goes through all Accounts and Opportunities on a nightly basis and updates them if necessary, based on custom criteria.

Batch Apex is exposed as an interface that must be implemented by the developer. Batch jobs can be programmatically invoked at runtime using Apex.

Batch jobs can also be programmatically scheduled to run at specific times using the Apex scheduler, or scheduled using the Schedule Apex page in the Salesforce user interface. For more information on the Schedule Apex page, see “Schedule Apex” in the Salesforce

**Start Method**:

To collect the records or objects to pass to the interface method execute, call the start method at the beginning of a batch Apex job. This method returns either a Database.QueryLocator object or an iterable that contains the records or objects passed to the job.

When you’re using a simple query (SELECT) to generate the scope of objects in the batch job, use the Database.QueryLocator object. If you use a QueryLocator object, the governor limit for the total number of records retrieved by SOQL queries is bypassed. For example, a batch Apex job for the Account object can return a QueryLocatorfor all account records (up to 50 million records) in an org. Another example is a sharing recalculation for the Contact object that returns a QueryLocator for all account records in an org.

Use the iterable to create a complex scope for the batch job. You can also use the iterable to create your own custom process for iterating through the list.

Syntax:

global (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext \*\*\*bc\*\*\*) {}

**Execute Method:**

This method takes the following:

* A reference to the Database.BatchableContext object.
* A list of sObjects, such as List<sObject>, or a list of parameterized types. If you are using a Database.QueryLocator, use the returned list.

Syntax:

global void execute(Database.BatchableContext \*\*\*BC\*\*\*, list<P>){}

Batches of records tend to execute in the order in which they’re received from the start method. However, the order in which batches of records execute depends on various factors. The order of execution isn’t guaranteed.

**Finish Method:**

* To send confirmation emails or execute post-processing operations, use the finish method. This method is called after all batches are processed.

Each execution of a batch Apex job is considered a discrete transaction. For example, a batch Apex job that contains 1,000 records and is executed without the optional scope parameter from Database.executeBatch is considered five transactions of 200 records each. The Apex governor limits are reset for each transaction. If the first transaction succeeds but the second fails, the database updates made in the first transaction are not rolled back.

global void finish(Database.BatchableContext \*\*\*BC\*\*\*){}

Schedule apex:

Salesforce schedules the class for execution at the specified time. Actual execution may be delayed based on service availability. You can only have 100 scheduled Apex jobs at one time. You can evaluate your current count by viewing the Scheduled Jobs page in Salesforce and creating a custom view with a type filter equal to “Scheduled Apex”.

You can also programmatically query the Trigger and Detail objects to get the count of Apex scheduled jobs. Use extreme care if you’re planning to schedule a class from a trigger. You must be able to guarantee that the trigger won’t add more scheduled classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

global class TestScheduledApexFromTestMethod implements Schedulable {

// This test runs a scheduled job at midnight Sept. 3rd. 2022

public static String CRON\_EXP = '0 0 0 3 9 ? 2022';

global void execute(SchedulableContext ctx) {

CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime

FROM CronTrigger WHERE Id = :ctx.getTriggerId()];

System.assertEquals(CRON\_EXP, ct.CronExpression);

System.assertEquals(0, ct.TimesTriggered);

System.assertEquals('2022-09-03 00:00:00', String.valueOf(ct.NextFireTime));

Account a = [SELECT Id, Name FROM Account WHERE Name =

'testScheduledApexFromTestMethod'];

a.name = 'testScheduledApexFromTestMethodUpdated';

update a;

}

If there are one or more active scheduled jobs for an Apex class,you cannot update the class or any classes referenced by this class through the Salesforce user interface. However, you can enable deployments to update the class with active scheduled jobs by using the Metadata API (for example, when using the Force.com IDE). See “Deployment Connections for Change Sets” in the Salesforce Help.