Getting Started with SalesForce CRM

VisualForce Controllers in SalesForce CRM

Description:

BISP is committed to provide BEST learning material to the beginners and advance learners. In the same series, we have prepared a complete end-to end Hands-on Beginner's Guide for SalesForce. The document focuses on Visualforce interface. Join our professional training program and learn from experts.

History:

Version	Description Change	Author	Publish Date
0.1	Initial Draft	Chandra Prakash Sharma	20th Dec 2013
0.1	Review#1	Amit Sharma	20th Dec 2013

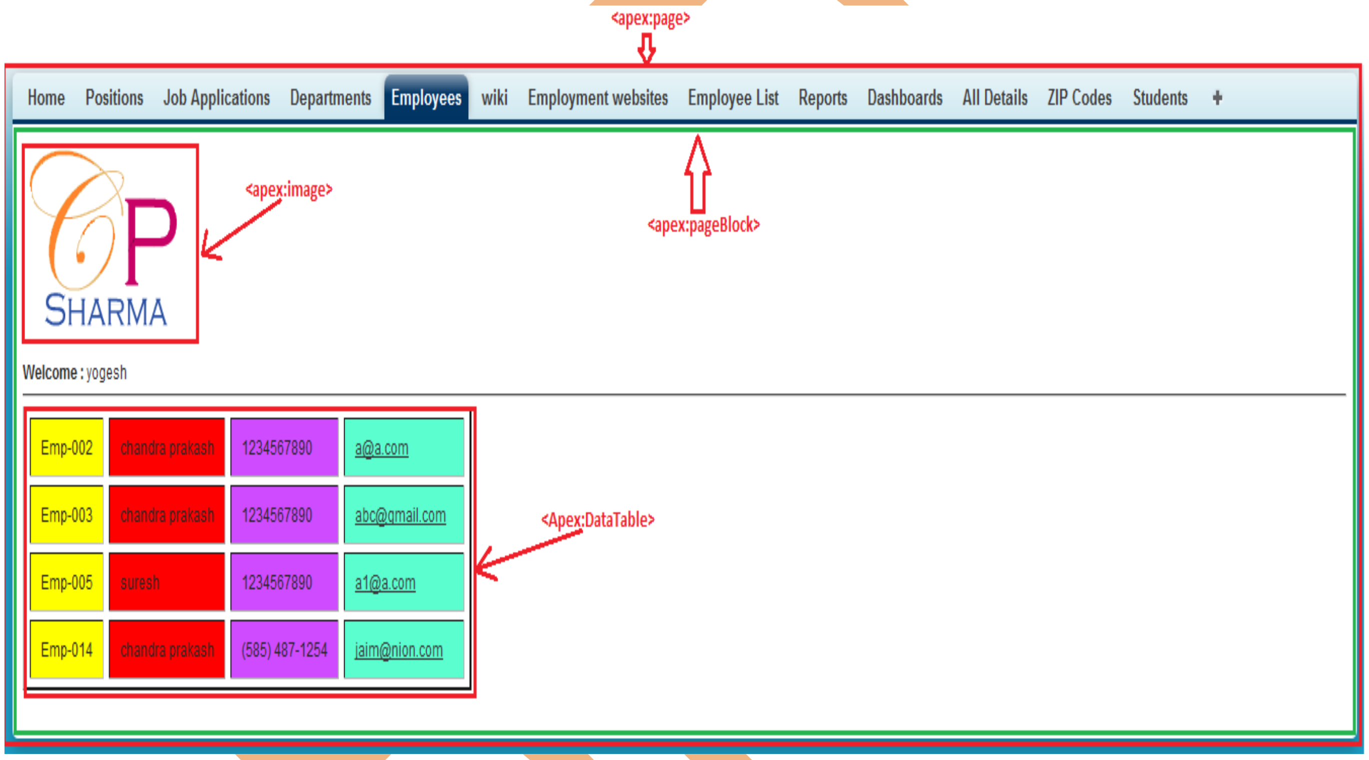
Contents

Understand the VisualForce framework, including its Advantages

Understand the VisualForce framework :

What is VisualForce ?

What is VisualForce?
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.



Welcome : yogesh

Emp-002	chandra prakash	1234567890	a@a.com
Emp-003	chandra prakash	1234567890	abc@gmail.com
Emp-005	suresh	1234567890	a1@a.com
Emp-014	chandra prakash	(585) 487-1254	jaim@nion.com

What is a VisualForce Page ?
Developers can use VisualForce to create a VisualForce page definition. A page definition consists of two primary elements:

- VisualForce markup
- A VisualForce controller

Where Can VisualForce Pages Be Used ?
Similar to s-controls, developers can use VisualForce pages to:
• Override standard buttons, such as the New button for accounts, or the Save button for contacts
• Override tab overview pages, such as the Accounts tab home page
• Define custom tabs
• Embed components in detail page layouts, similar to the way inline s-controls can be embedded

Advantage of VisualForce :

As a markup language, VisualForce provides the following benefits :
User-friendly development :
Developers can edit their VisualForce markup in the same window that displays the resulting page. Consequently, developers can instantly verify the result of an edit just by saving their code. The VisualForce editor pane also include auto-completion and syntax highlighting.

Integration with other Web-based user interface technologies :
Because VisualForce markup is ultimately rendered into HTML, designers can use VisualForce tags alongside standard HTML, JavaScript, Flash, or any other code that can execute within an HTML page on the platform, including Force.com platform merge fields and expressions.

Model-View-Controller (MVC) style development :
VisualForce conforms to the Model-View-Controller (MVC) development pattern by providing a clear division between the view of an application (the user interface, defined by VisualForce markup), and the
controller and the controller that determines how the application works(the business logic, defined by a VisualForce controller written in Apex).

Concise syntax :
VisualForce pages can implement the same functionality as s-controls but with approximately 90% fewer lines of code.

Data-driven defaults :
VisualForce components are rendered intelligently by the platform. For example, rather than forcing page designers to use different component tags for different types of editable fields (such as email addresses or calendar dates), designer scan simply use a generic <apex:inputField> tag for all fields. The VisualForce renderer displays the appropriate edit interface for each field.

Hosted platform :
VisualForce pages are compiled and rendered entirely by the Force.com platform. Because they are so tightly integrated, they display the same performance as standard SalesForce pages, regardless of the amount of data being displayed or edited.

Automatically upgradeable :
VisualForce pages do not need to be rewritten when other parts of the Force.com platform are upgraded. Because the pages are stored as metadata, they are automatically upgraded with the rest of the system.

Use expressions to bind data and actions on a page to a controller

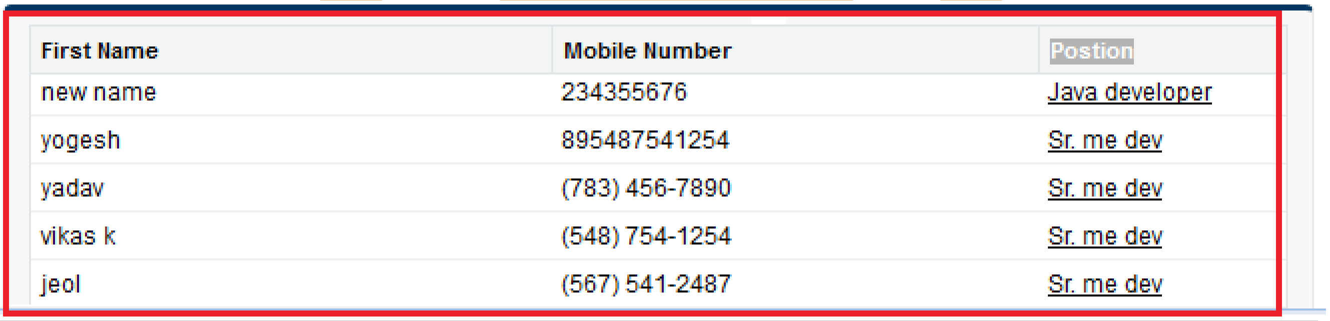
Dynamic VisualForce Bindings :

In SalesForce Dynamic VisualForce bindings are a way of writing generic VisualForce pages that display information about records without necessarily knowing which fields to show. In other words, fields on the page are determined at run time, rather than compile time. This allows a developer to design a single page that renders differently for various audiences, based on their permissions or preferences. Dynamic bindings are useful for VisualForce pages included in managed packages since they allow for the presentation of data specific to each subscriber with very little coding.

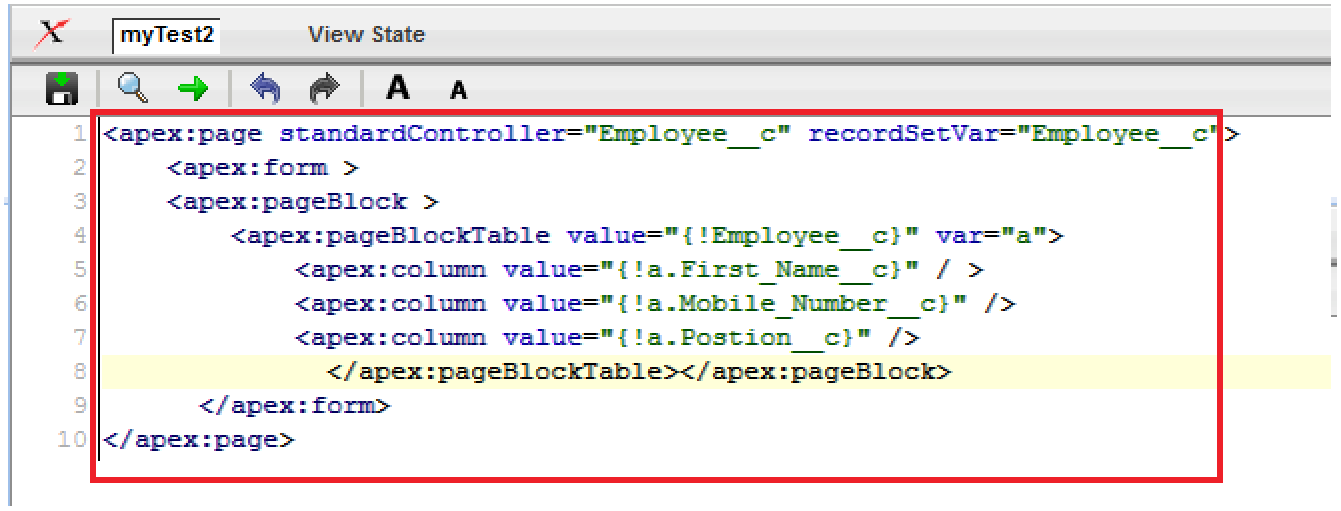
Dynamic VisualForce binding is supported for standard and custom objects :
Syntax : reference[expression]

Dynamic bindings can be used anywhere formula expressions are valid :
Syntax : {!reference[expression]}

Enters Dynamic VisualForce Binding.
Let's just quickly see what a regular VisualForce page looks like. As an example I want to display the Employee first Name, Mobile Number, Position. To do that, all you need to do is a very simple VisualForce page like this :



First Name	Mobile Number	Postion
new name	234355676	Java developer
yogesh	895487541254	Sr. me dev
yadav	(783) 456-7890	Sr. me dev
vikas k	(548) 754-1254	Sr. me dev
jeol	(567) 541-2487	Sr. me dev



```
<apex:page standardController="Employee__c" recordSetVar="Employee__c">
    <apex:form >
    <apex:pageBlock >
        <apex:pageBlockTable value="{!Employee__c}" var="a">
            <apex:column value="{!a.First_Name__c}" / >
            <apex:column value="{!a.Mobile_Number__c}" />
            <apex:column value="{!a.Postion__c}" />
              </apex:pageBlockTable></apex:pageBlock>
      </apex:form>
</apex:page>
```

Actions on a page to a controller :

Using Standard List Controller Actions :

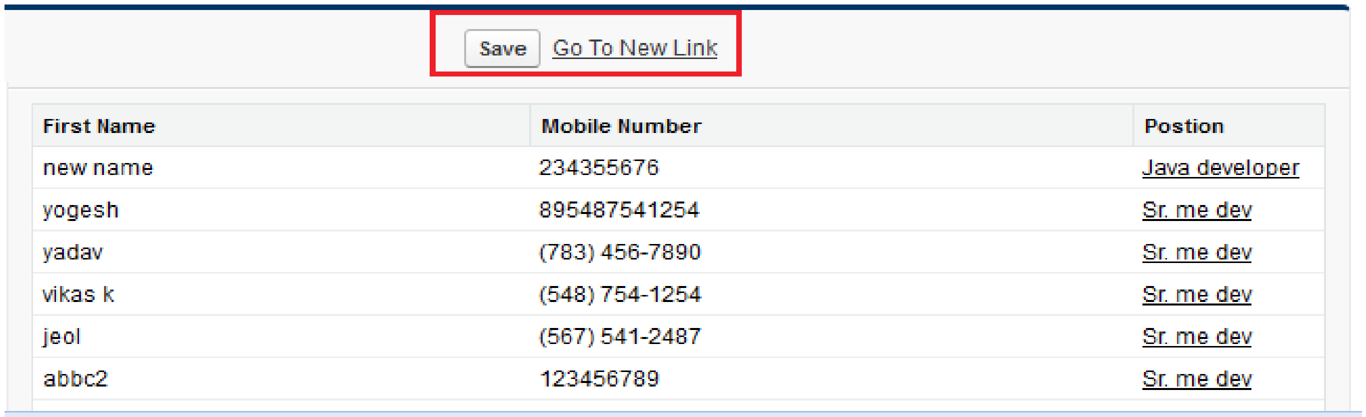
Action methods perform logic or navigation when a page event occurs, such as when a user clicks a button, or hovers over an area of the page.

Syntax	Description
<apex:commandButton>	creates a button that calls an action
<apex:commandLink>	creates a link that calls an action
<apex:actionPoller>	periodically calls an action
<apex:actionSupport>	makes an event (Ex: "onclick", "onmouseover", and so on) on another, named component, call an action
<apex:actionFunction>	defines a new JavaScript function that calls an action
<apex:page>	calls an action when the page is loaded

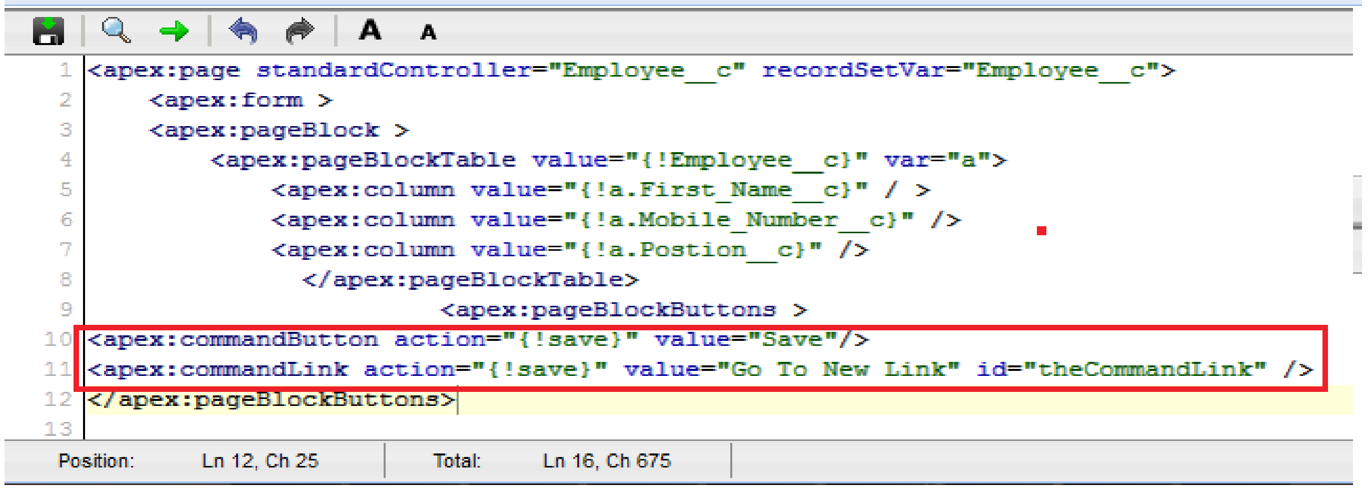
Action Methods :

Action	Description
Save	Inserts new records or updates existing records that have been changed.
quicksave	Inserts new records or updates existing records that have been changed.
list	Returns a PageReference object of the standard list page, based on the most recently used list filter for that object when the filterId is not specified by the user.
cancel	Aborts an edit operation.
`first`	Displays the first page of records in the set.
last	Displays the last page of records in the set.
next	Displays the next page of records in the set.
previous	Displays the previous page of records in the set.

You can see below Example :



First Name	Mobile Number	Postion
new name	234355676	Java developer
yogesh	895487541254	Sr. me dev
yadav	(783) 456-7890	Sr. me dev
vikas k	(548) 754-1254	Sr. me dev
jeol	(567) 541-2487	Sr. me dev
abbc2	123456789	Sr. me dev



```
<apex:page standardController="Employee__c" recordSetVar="Employee__c">
    <apex:form >
    <apex:pageBlock >
        <apex:pageBlockTable value="{!Employee__c}" var="a">
            <apex:column value="{!a.First_Name__c}" / >
            <apex:column value="{!a.Mobile_Number__c}" />
            <apex:column value="{!a.Postion__c}" />
              </apex:pageBlockTable>
                       <apex:pageBlockButtons >
<apex:commandButton action="{!save}" value="Save"/>
<apex:commandLink action="{!save}" value="Go To New Link" id="theCommandLink" />
</apex:pageBlockButtons>
```

Create custom controllers and standard controller extensions to incorporate new data and actions into a page

Custom Controllers :

A custom controller is an Apex class that implements all of the logic for a page without leveraging a standard controller.

- You want to leverage the built-in functionality of a standard controller but override one or more actions, such as edit, view, save, or delete.
- You want to add new actions.
- You want to build a VisualForce page that respects user permissions.

Standard controllers can provide all the functionality you need for a VisualForce page because they include the same logic that is used for a standard 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.

However, if you want to override existing functionality, customize the navigation through an application, use callouts or Web services, or if you need finer control for how information is accessed for your page, you can write a custom controller or a controller extension using Apex:

What are Custom Controllers and Controller Extensions?

In VisualForce custom control is an apex class that implement all of the logic for a page without leaving a standard controller. Custom control use in VisualForce page run entirely in system mode they does not enforce the permissions and field-level security of the current user.

How To Create Page By Using Custom Controllers :

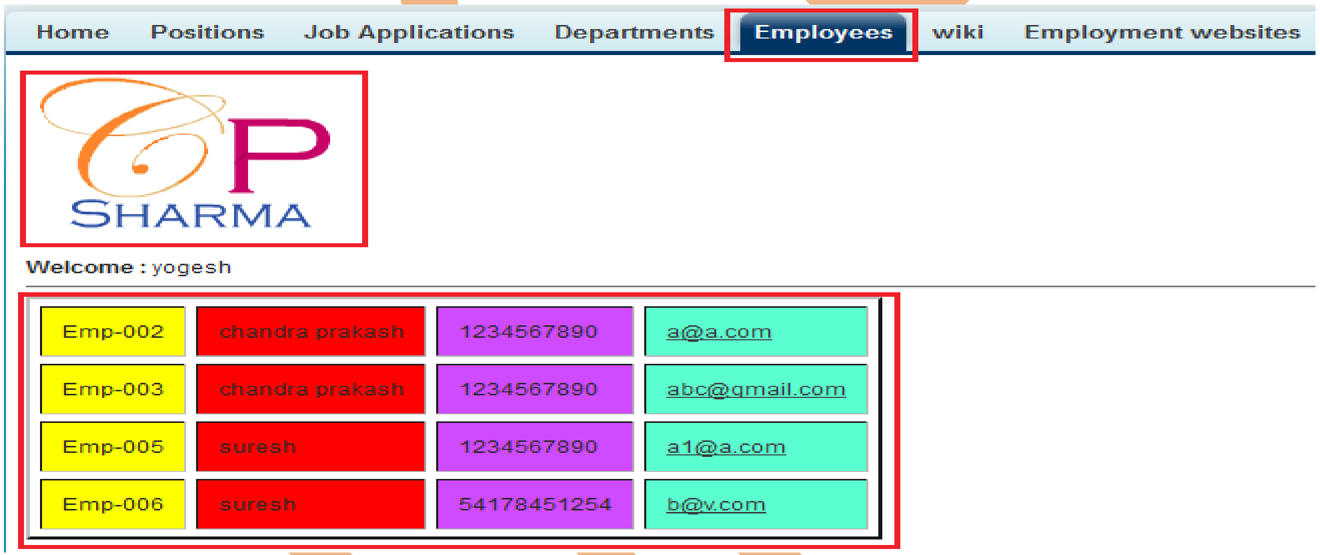
Click on **Setup** > **develop** > **Pages** Then Click on New button.

OR

Directly write in address bar after login SalesForce site then add given address in url /apex/Newpage1
for example you see here https://c.ap1.visual.force.com/apex/Newpage1

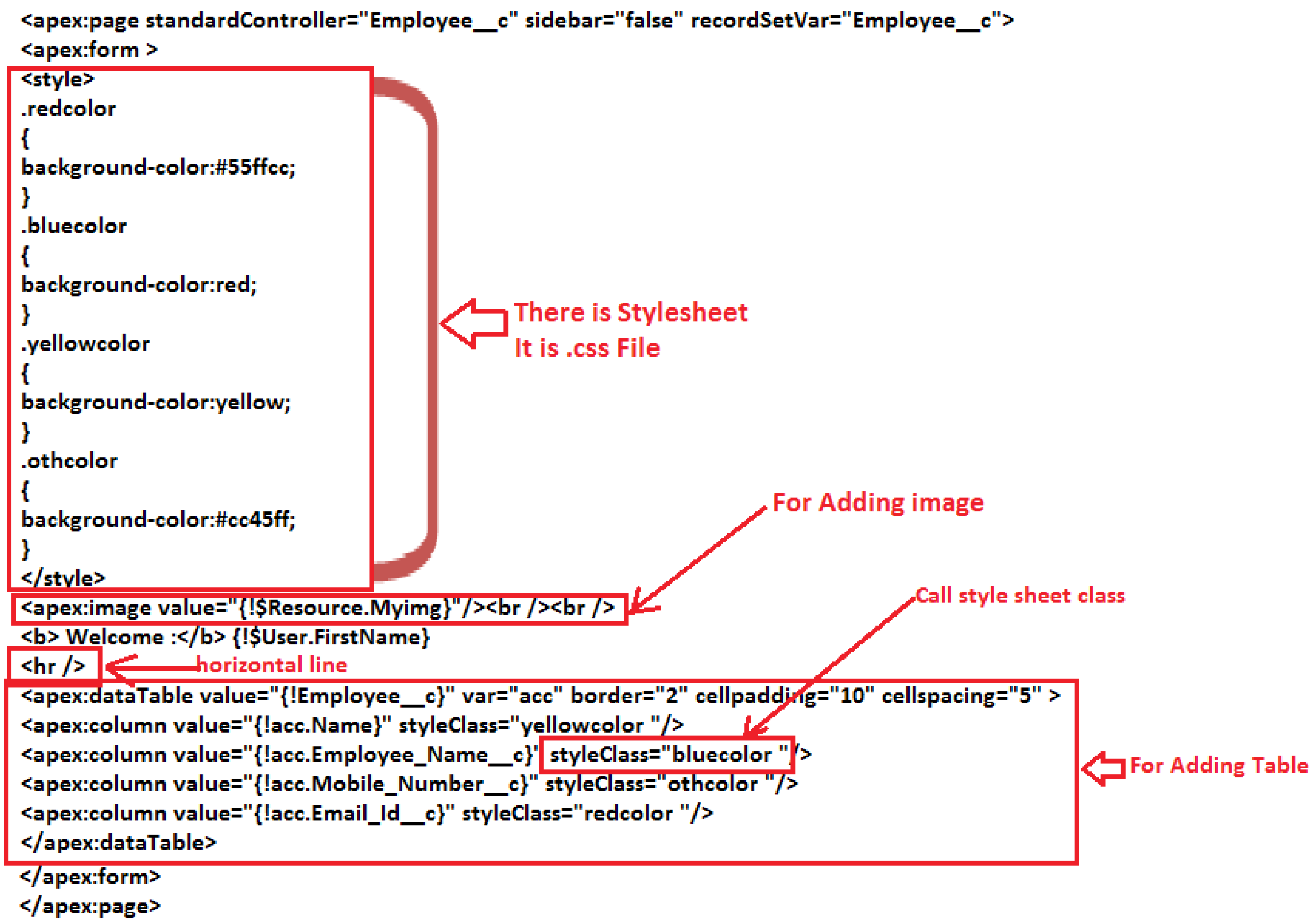
Note :- 1.Make Sure in User Setting Development Mode is enable.
2. You can Custom Control you in Standard Object and Custom object.

Example 1:- In this example create table with style sheet and how to call image by using custom object Controller.
Step 1 :- Create new page /apex/Newpage1



Welcome : yogesh

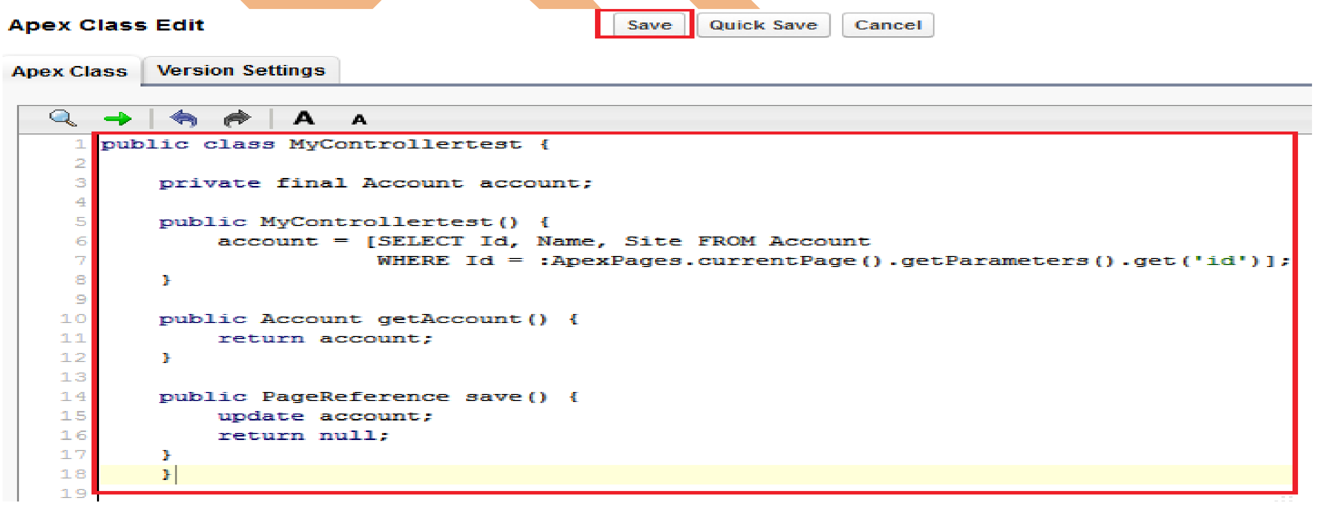
Emp-002	chandra prakash	1234567890	a@a.com
Emp-003	chandra prakash	1234567890	abc@gmail.com
Emp-005	suresh	1234567890	a1@a.com
Emp-006	suresh	54178451254	b@v.com


```
<apex:page standardController="Employee__c" sidebar="false" recordSetVar="Employee__c">
<apex:form >
<style>
.redcolor
{
background-color:#55ffcc;
}
.bluecolor
{
background-color:red;
}
.yellowcolor
{
background-color:yellow;
}
.othcolor
{
background-color:#cc45ff;
}
</style>
<apex:image value="{!$Resource.Myimg}"/><br /><br />
<b> Welcome :</b> {!$User.FirstName}
<hr />
<apex:dataTable value="{!Employee__c}" var="acc" border="2" cellpadding="10" cellspacing="5" >
<apex:column value="{!acc.Name}" styleClass="yellowcolor "/>
<apex:column value="{!acc.Employee_Name__c}" styleClass="bluecolor "/>
<apex:column value="{!acc.Mobile_Number__c}" styleClass="othcolor "/>
<apex:column value="{!acc.Email_Id__c}" styleClass="redcolor "/>
</apex:dataTable>
</apex:form>
</apex:page>
```
There is Stylesheet
It is .css File
For Adding image
Call style sheet class
horizontal line
For Adding Table

Note :- Before using this code <apex:image value="{!$Resource.imagename}"/> make sure add image file in Static Resources.

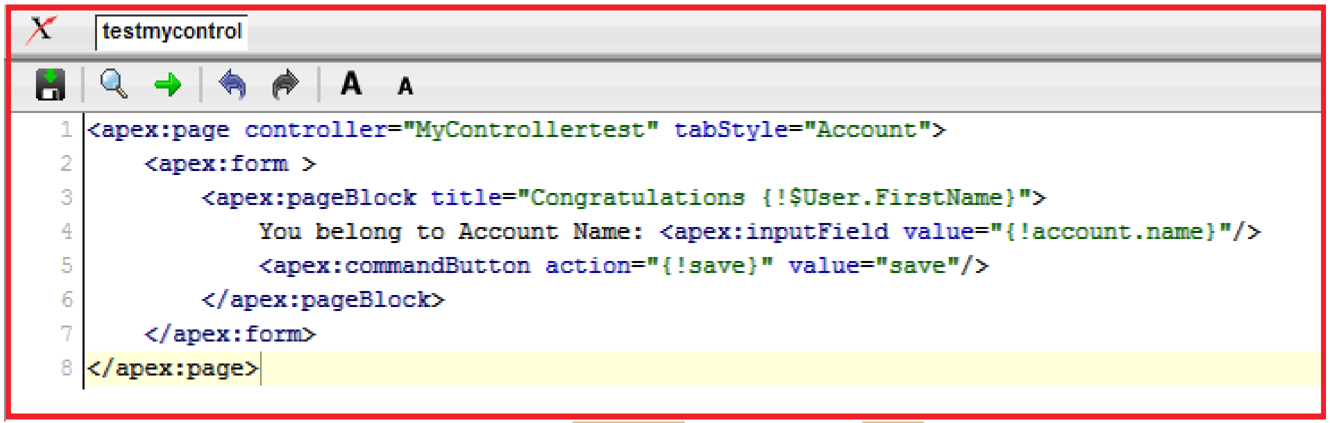
How to Building a Custom Controller :

From **Setup** > **Develop** > **Apex Classes** then click **New** button.



```
public class MyControllertest {

    private final Account account;

    public MyControllertest() {
        account = [SELECT Id, Name, Site FROM Account
                   WHERE Id = :ApexPages.currentPage().getParameters().get('id')];
    }

    public Account getAccount() {
        return account;
    }

    public PageReference save() {
        update account;
        return null;
    }
}
```

After create apex class then you can create VisualForce page you can see below.



```
<apex:page controller="MyControllertest" tabStyle="Account">
    <apex:form >
        <apex:pageBlock title="Congratulations {!$User.FirstName}">
            You belong to Account Name: <apex:inputField value="{!account.name}"/>
            <apex:commandButton action="{!save}" value="save"/>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

Controller Methods :

Action Control :

<apex:commandButton> creates a button that calls an action
<apex:commandLink> creates a link that calls an action
<apex:actionPoller> periodically calls an action
<apex:actionSupport> makes an event (such as "onclick", "onmouseover", and so on) on another, named component, call an action
<apex:actionFunction> defines a new JavaScript function that calls an action
<apex:page> calls an action when the page is loaded

Standard Controllers :

In Salesforce.com 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.

Associating a Standard Controller with a VisualForce Page :

Use the `standardController` attribute on the `<apex:page> </apex:page>` tag and assign it the name of any SalesForce object that can be queried using the Force.com API.

Syntax :

```
<apex:page standardController="Account">

</apex:page>
```

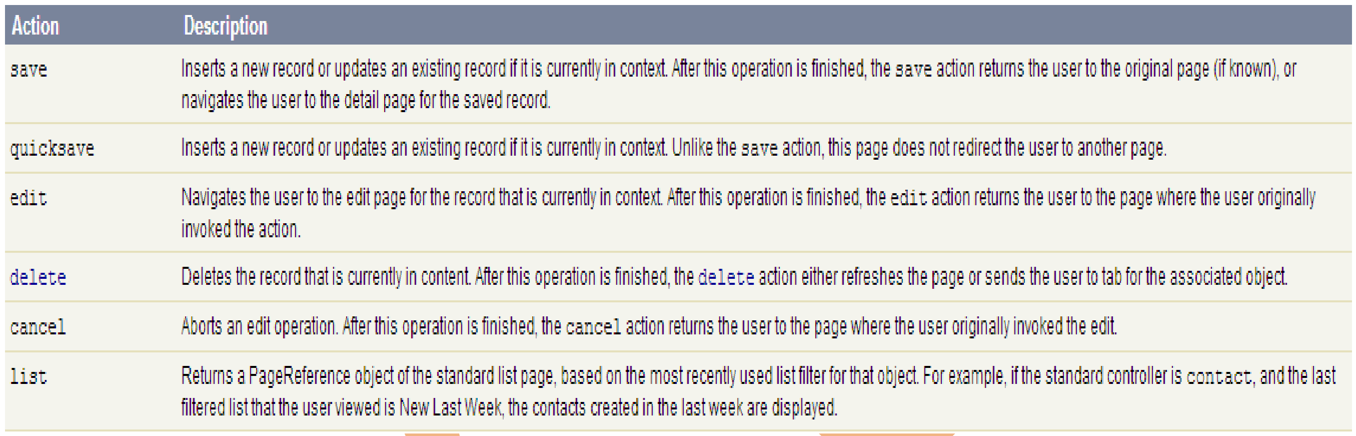
Accessing Data with a Standard Controller :

Every standard controller includes a getter method that returns the record specified by the id query string parameter in the page URL.
- You can traverse up to five levels of child-to-parent relationship.
- You can traverse one level of parent-to-child relationships.

Standard Controller Actions :

<apex:commandButton> creates a button that calls an action

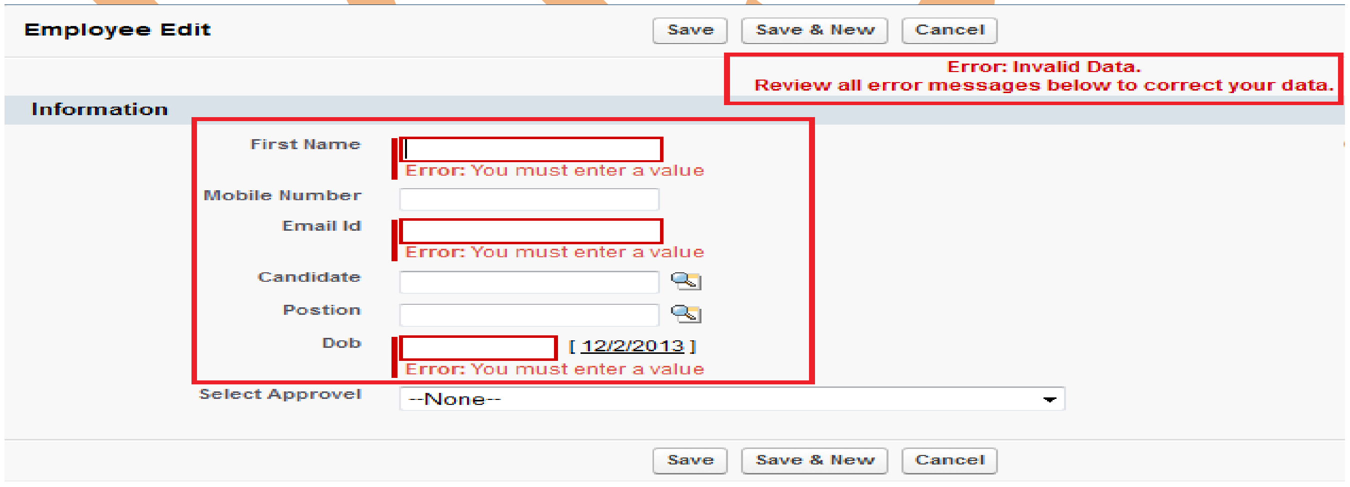
<apex:commandLink> creates a link that calls an action
<apex:actionPoller> periodically calls an action
<apex:actionSupport> makes an event (such as “onclick”, “onmouseover”, and so on) on another, named component, call an action
<apex:actionFunction> defines a new JavaScript function that calls an action
<apex:page> calls an action when the page is loaded



Action	Description
`save`	Inserts a new record or updates an existing record if it is currently in context. After this operation is finished, the `save` action returns the user to the original page (if known), or navigates the user to the detail page for the saved record.
`quicksave`	Inserts a new record or updates an existing record if it is currently in context. Unlike the `save` action, this page does not redirect the user to another page.
`edit`	Navigates the user to the edit page for the record that is currently in context. After this operation is finished, the `edit` action returns the user to the page where the user originally invoked the action.
`delete`	Deletes the record that is currently in content. After this operation is finished, the `delete` action either refreshes the page or sends the user to tab for the associated object.
`cancel`	Aborts an edit operation. After this operation is finished, the `cancel` action returns the user to the page where the user originally invoked the edit.
`list`	Returns a PageReference object of the standard list page, based on the most recently used list filter for that object. For example, if the standard controller is `contact`, and the last filtered list that the user viewed is New Last Week, the contacts created in the last week are displayed.

Validation Rules and Standard Controllers :

If a user enters data on a SalesForce page that uses a standard controller, and that data causes a validation rule error, the error can be displayed on the SalesForce page. If the validation rule error location is a field associated with an <apex:inputField> component, the error displays there. If the validation rule error location is set to the top of the page, use the <apex:pageMessages> or <apex:messages> component within the <apex:page> to display the error.
you can see below



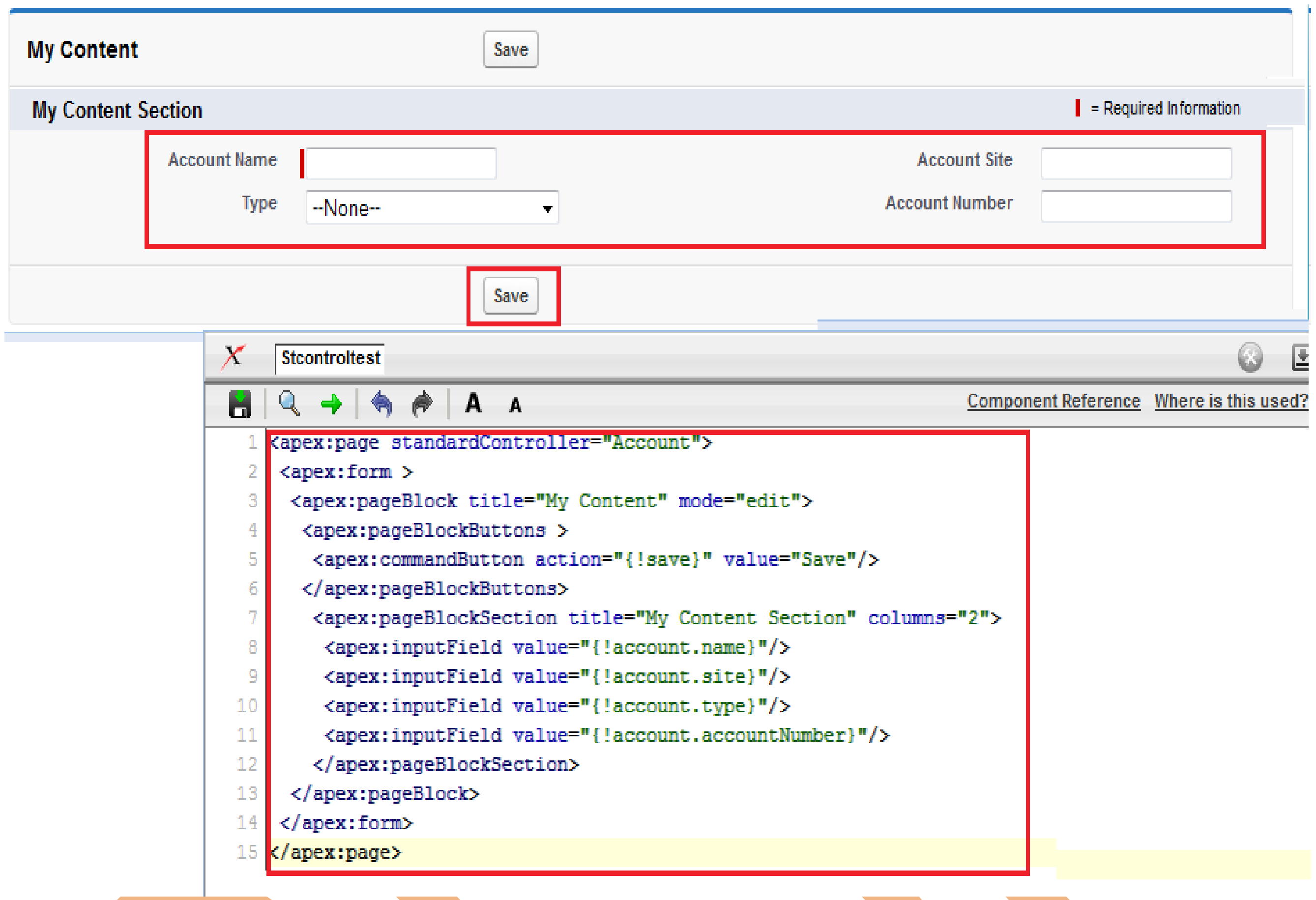
Employee Edit

Error: Invalid Data.
Review all error messages below to correct your data.

Information

First Name	Error: You must enter a value
Mobile Number	
Email Id	Error: You must enter a value
Candidate	
Postion	
Dob	[12/2/2013] Error: You must enter a value
Select Approvel	--None--

Example : The following page allows you to update an account after will all fields then click on **Save** button.



```
<apex:page standardController="Account">
 <apex:form >
  <apex:pageBlock title="My Content" mode="edit">
   <apex:pageBlockButtons >
    <apex:commandButton action="{!save}" value="Save"/>
   </apex:pageBlockButtons>
    <apex:pageBlockSection title="My Content Section" columns="2">
     <apex:inputField value="{!account.name}"/>
     <apex:inputField value="{!account.site}"/>
     <apex:inputField value="{!account.type}"/>
     <apex:inputField value="{!account.accountNumber}"/>
    </apex:pageBlockSection>
  </apex:pageBlock>
 </apex:form>
</apex:page>
```

Understand the security implications of using custom vs. standard controllers

Custom Control :

Standard controllers can provide all the functionality you need for a VisualForce page because they include the same logic that is used for a standard 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.

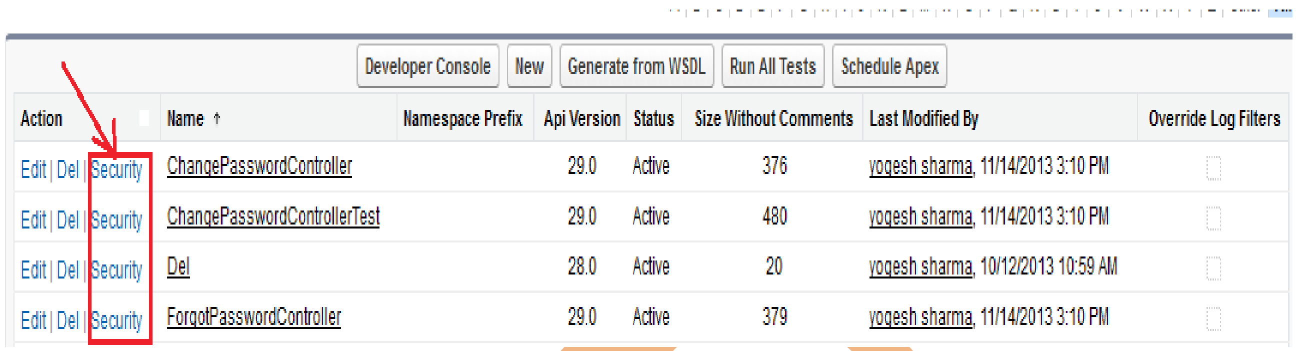
Standard Control :

In Salesforce.com 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.
contain the same functionality and logic that are used for standardSalesforce pages.
Can be used with standard objects and custom objects.

Controller Class Security :

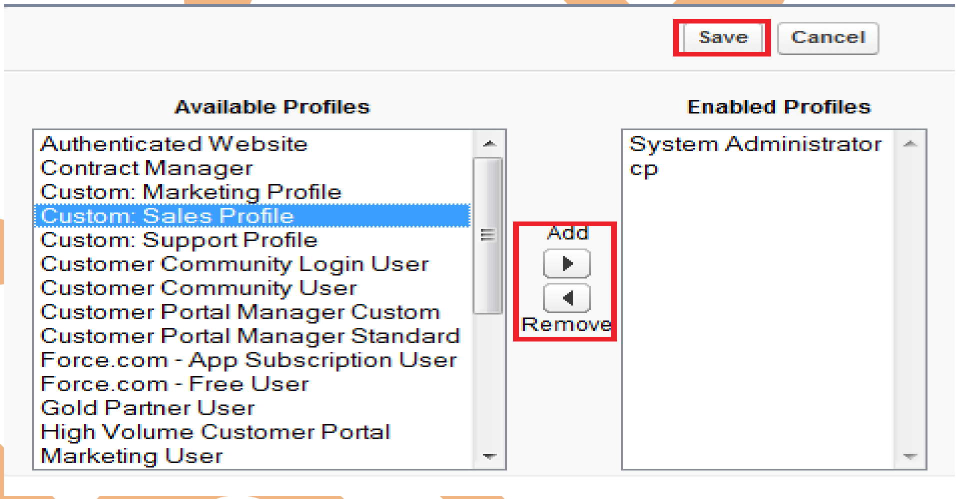
To set Apex class security from the class list page:

Setup > **Develop > Apex Classes**
then you can see security link on given page you can see below.



Action	Name ↑	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Override Log Filters
Edit | Del | Security	ChangePasswordController		29.0	Active	376	yogesh sharma, 11/14/2013 3:10 PM	☐
Edit | Del | Security	ChangePasswordControllerTest		29.0	Active	480	yogesh sharma, 11/14/2013 3:10 PM	☐
Edit | Del | Security	Del		28.0	Active	20	yogesh sharma, 10/12/2013 10:59 AM	☐
Edit | Del | Security	ForgotPasswordController		29.0	Active	379	yogesh sharma, 11/14/2013 3:10 PM	☐

Select the profiles that you want to enable from the Available Profiles list and click **Add**, or select the profiles that you want to disable from the Enabled Profiles list and click **Remove**.
Then click **Save.**
You can see below.



Available Profiles	Enabled Profiles
Authenticated Website	System Administrator
Contract Manager	cp
Custom: Marketing Profile	
Custom: Sales Profile	
Custom: Support Profile	
Customer Community Login User	
Customer Community User	
Customer Portal Manager Custom	
Customer Portal Manager Standard	
Force.com - App Subscription User	
Force.com - Free User	
Gold Partner User	
High Volume Customer Portal	
Marketing User	

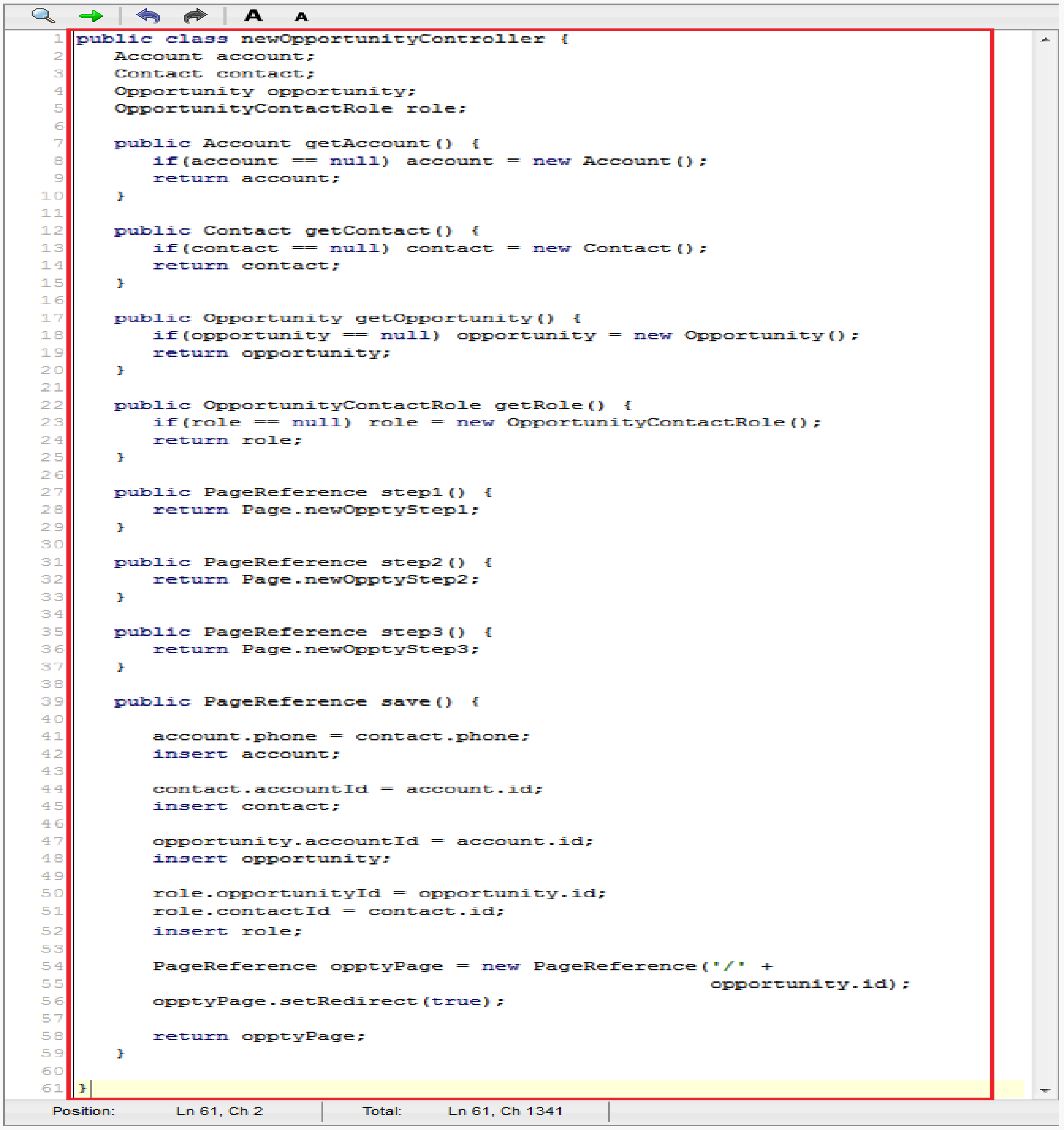
Implement wizards using custom controllers to handle the state and operations

Example : If you want to create a three-step opportunity wizard that allows users to create an opportunity at the same time as a related contact, account, and contact role:

- The first step captures information related to the account and contact.
- The second step captures information related to the opportunity.
- The final step shows which records will be created and allows the user to save or cancel .

Solution :
1. Navigate to the URL for the Create VisualForce first page:
https://<host>.salesforce.com/apex/opptyStep1
2. Click Create Page newOpptyStep1.

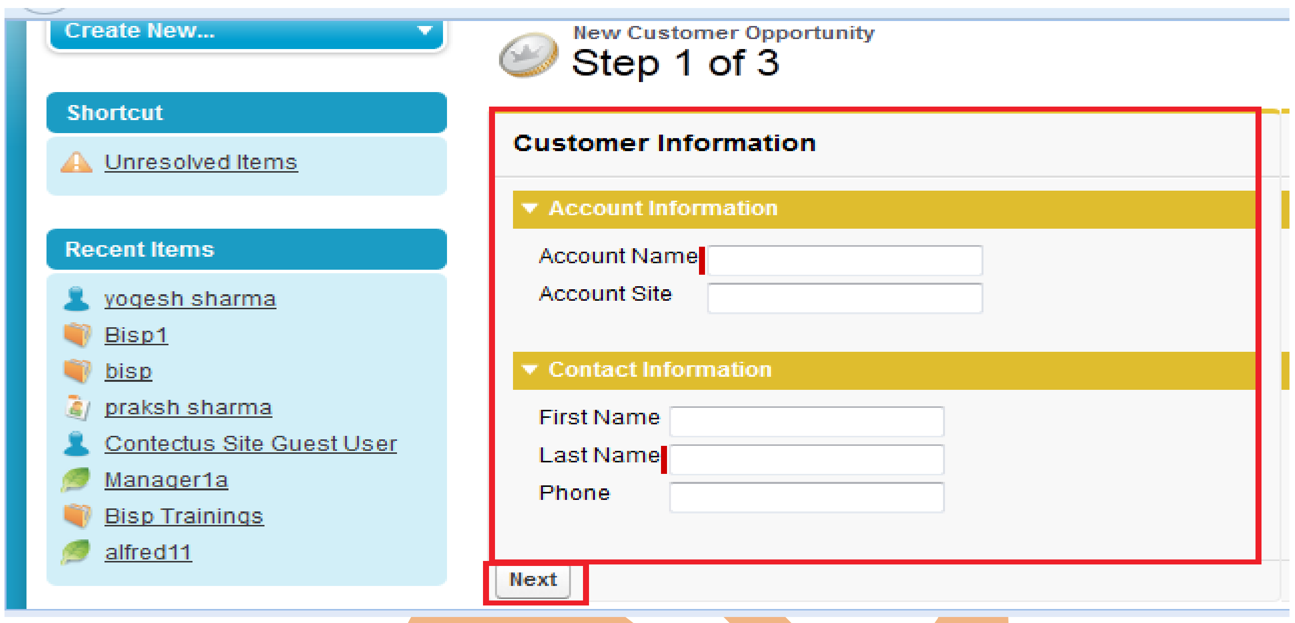
3. Repeat the Step (Step 1 or Step 2) and give VisualForce page Name opptyStep2 and opptyStep3.
4. Create the newOpportunityController controller it's create on Apex class.
you can see below.



```
public class newOpportunityController {
    Account account;
    Contact contact;
    Opportunity opportunity;
    OpportunityContactRole role;

    public Account getAccount() {
        if(account == null) account = new Account();
        return account;
    }

    public Contact getContact() {
        if(contact == null) contact = new Contact();
        return contact;
    }

    public Opportunity getOpportunity() {
        if(opportunity == null) opportunity = new Opportunity();
        return opportunity;
    }

    public OpportunityContactRole getRole() {
        if(role == null) role = new OpportunityContactRole();
        return role;
    }

    public PageReference step1() {
        return Page.newOpptyStep1;
    }

    public PageReference step2() {
        return Page.newOpptyStep2;
    }

    public PageReference step3() {
        return Page.newOpptyStep3;
    }

    public PageReference save() {

        account.phone = contact.phone;
        insert account;

        contact.accountId = account.id;
        insert contact;

        opportunity.accountId = account.id;
        insert opportunity;

        role.opportunityId = opportunity.id;
        role.contactId = contact.id;
        insert role;

        PageReference opptyPage = new PageReference('/' +
                                                    opportunity.id);
        opptyPage.setRedirect(true);

        return opptyPage;
    }

}
```

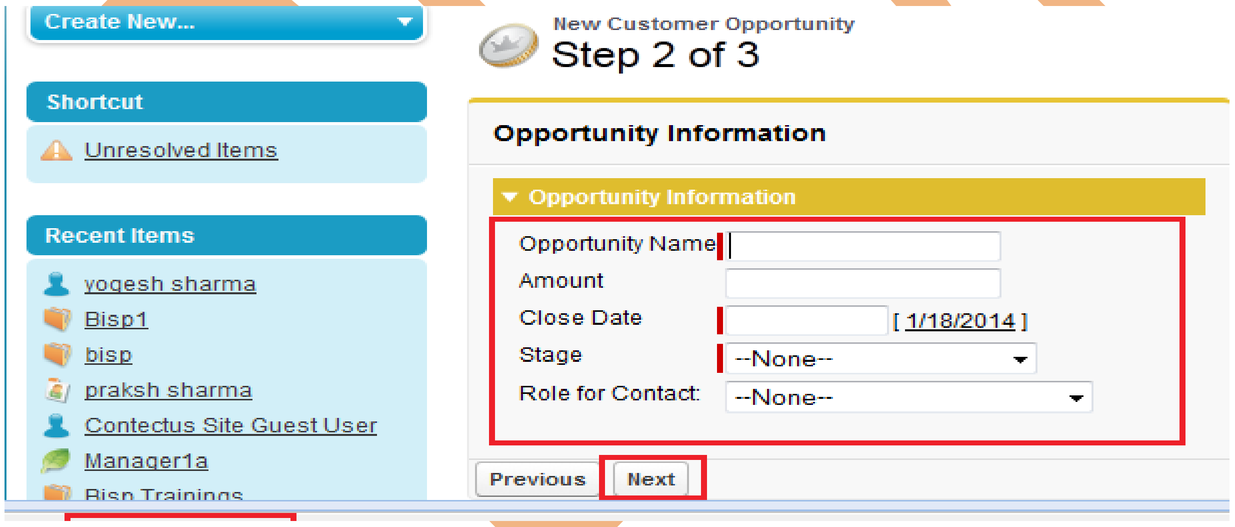
```
Position:    Ln 61, Ch 2    Total:    Ln 61, Ch 1341
```

(
Force page. You can see below.

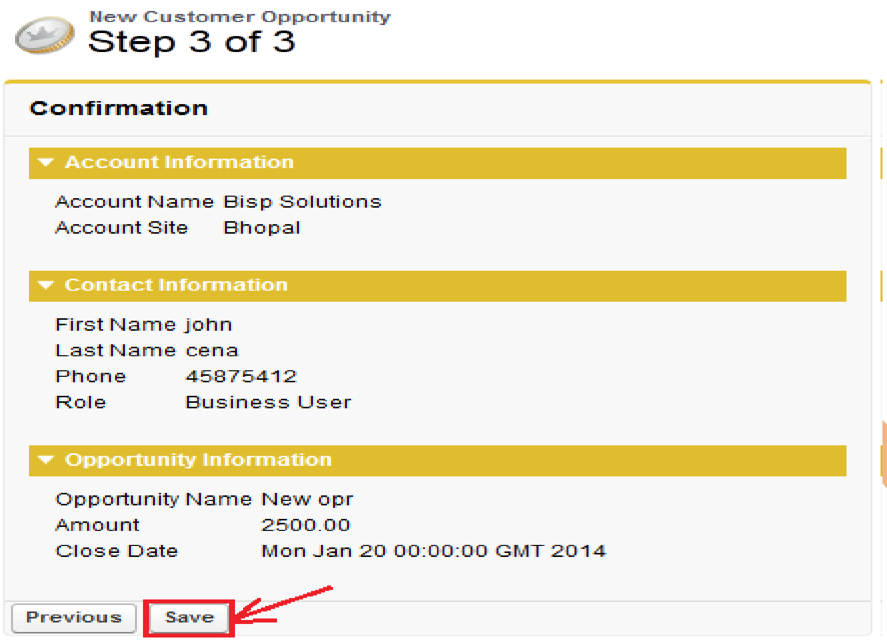


```
<apex:page controller="newOpportunityController"
       tabStyle="Opportunity">
  <apex:sectionHeader title="New Customer Opportunity"
                  subtitle="Step 1 of 3"/>
  <apex:form >
   <apex:pageBlock title="Customer Information">

     <!-- This facet tag defines the "Next" button that appears
        in the footer of the pageBlock. It calls the step2()
        controller method, which returns a pageReference to
        the next step of the wizard. -->

     <apex:facet name="footer">
      <apex:commandButton action="{!step2}" value="Next"
                   styleClass="btn"/>
     </apex:facet>
     <apex:pageBlockSection title="Account Information">

     <!-- <apex:panelGrid> tags organize data in the same way as
         a table. It places all child elements in successive cells,
         in left-to-right, top-to-bottom order -->

     <!-- <apex:outputLabel > and <apex:inputField > tags can be
         bound together with the for and id attribute values,
         respectively. -->

      <apex:panelGrid columns="2">
       <apex:outputLabel value="Account Name" for="accountName"/>
       <apex:inputField id="accountName" value="{!account.name}"/>
      <apex:outputLabel value="Account Site" for="accountSite"/>
      <apex:inputField id="accountSite" value="{!account.site}"/>
     </apex:panelGrid>
    </apex:pageBlockSection>
    <apex:pageBlockSection title="Contact Information">
     <apex:panelGrid columns="2">
       <apex:outputLabel value="First Name"
                         for="contactFirstName"/>
       <apex:inputField id="contactFirstName"
                        value="{!contact.firstName}"/>
       <apex:outputLabel value="Last Name" for="contactLastName"/>
       <apex:inputField id="contactLastName"
                        value="{!contact.lastName}"/>
       <apex:outputLabel value="Phone" for="contactPhone"/>
       <apex:inputField id="contactPhone"
                        value="{!contact.phone}"/>
     </apex:panelGrid>
    </apex:pageBlockSection>
   </apex:pageBlock>
  </apex:form>
</apex:page>
```

Create newOpptyStep1, then create newOpptyStep2 VisualForce page.



```
<apex:page controller="newOpportunityController"
           tabStyle="Opportunity">
  <apex:sectionHeader title="New Customer Opportunity"
                      subtitle="Step 2 of 3"/>
  <apex:form >
    <apex:pageBlock title="Opportunity Information">
    <apex:facet name="footer">
     <apex:outputPanel >
      <apex:commandButton action="{!step1}" value="Previous"
                          styleClass="btn"/>
      <apex:commandButton action="{!step3}" value="Next"
                          styleClass="btn"/>
     </apex:outputPanel>
    </apex:facet>
    <apex:pageBlockSection title="Opportunity Information">
     <apex:panelGrid columns="2">
      <apex:outputLabel value="Opportunity Name"
                       for="opportunityName"/>
      <apex:inputField id="opportunityName"
                      value="{!opportunity.name}"/>
      <apex:outputLabel value="Amount"
                       for="opportunityAmount"/>
      <apex:inputField id="opportunityAmount"
                      value="{!opportunity.amount}"/>
      <apex:outputLabel value="Close Date"
                       for="opportunityCloseDate"/>
      <apex:inputField id="opportunityCloseDate"
                      value="{!opportunity.closeDate}"/>
      <apex:outputLabel value="Stage"
                       for="opportunityStageName"/>
      <apex:inputField id="opportunityStageName"
                      value="{!opportunity.stageName}"/>
      <apex:outputLabel value="Role for Contact:
                              {!contact.firstName}
                              {!contact.lastName}"
                       for="contactRole"/>
      <apex:inputField id="contactRole"
                      value="{!role.role}"/>
     </apex:panelGrid>
    </apex:pageBlockSection>
   </apex:pageBlock>
  </apex:form>
</apex:page>
```
Create newOpptyStep2, then create newOpptyStep3 VisualForce page.


New Customer Opportunity

Step 3 of 3

Confirmation

Account Information

Account Name	Bisp Solutions
Account Site	Bhopal

Contact Information

First Name	john
Last Name	cena
Phone	45875412
Role	Business User

Opportunity Information

Opportunity Name	New opr
Amount	2500.00
Close Date	Mon Jan 20 00:00:00 GMT 2014

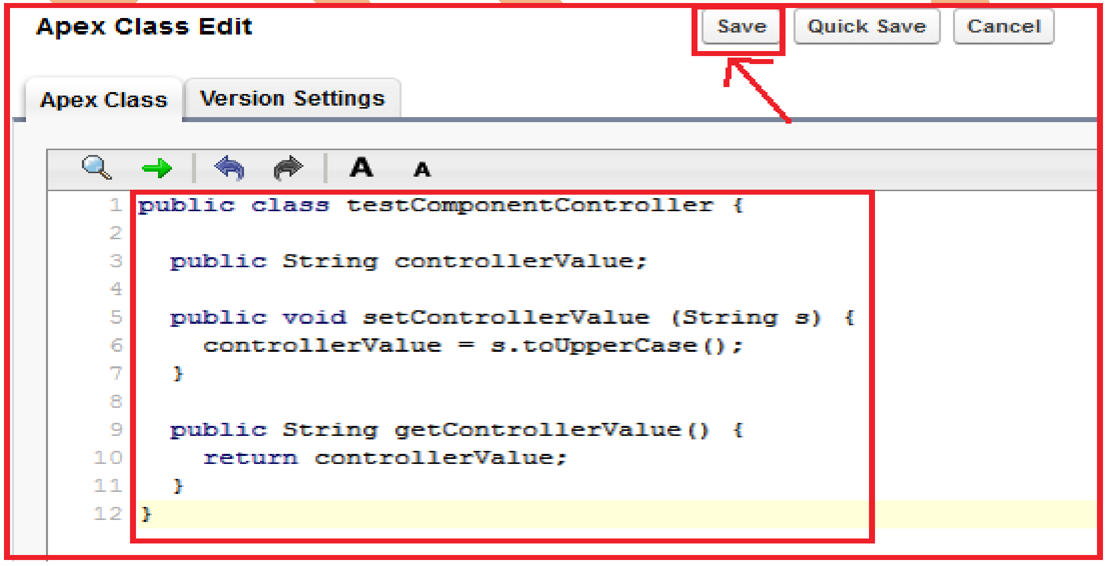
```
<apex:page controller="newOpportunityController"
        tabStyle="Opportunity">
 <apex:sectionHeader title="New Customer Opportunity"
                  subtitle="Step 3 of 3"/>
 <apex:form >
  <apex:pageBlock title="Confirmation">
   <apex:facet name="footer">
    <apex:outputPanel >
     <apex:commandButton action="{!step2}" value="Previous"
                     styleClass="btn"/>
     <apex:commandButton action="{!save}" value="Save"
                     styleClass="btn"/>
    </apex:outputPanel>
   </apex:facet>
   <apex:pageBlockSection title="Account Information">
    <apex:panelGrid columns="2">
     <apex:outputText value="Account Name"/>
     <apex:outputText value="{!account.name}"/>
     <apex:outputText value="Account Site"/>
     <apex:outputText value="{!account.site}"/>
    </apex:panelGrid>
   </apex:pageBlockSection>
   <apex:pageBlockSection title="Contact Information">
    <apex:panelGrid columns="2">
     <apex:outputText value="First Name"/>
     <apex:outputText value="{!contact.firstName}"/>
     <apex:outputText value="Last Name"/>
     <apex:outputText value="{!contact.lastName}"/>
     <apex:outputText value="Phone"/>
```

```
        <apex:outputText value="{!contact.phone}"/>
        <apex:outputText value="Role"/>
        <apex:outputText value="{!role.role}"/>
      </apex:panelGrid>
    </apex:pageBlockSection>
    <apex:pageBlockSection title="Opportunity Information">
      <apex:panelGrid columns="2">
        <apex:outputText value="Opportunity Name"/>
        <apex:outputText value="{!opportunity.name}"/>
        <apex:outputText value="Amount"/>
        <apex:outputText value="{!opportunity.amount}"/>
        <apex:outputText value="Close Date"/>
        <apex:outputText value="{!opportunity.closeDate}"/>
      </apex:panelGrid>
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:form>
</apex:page>
```

Create custom components that use custom controllers

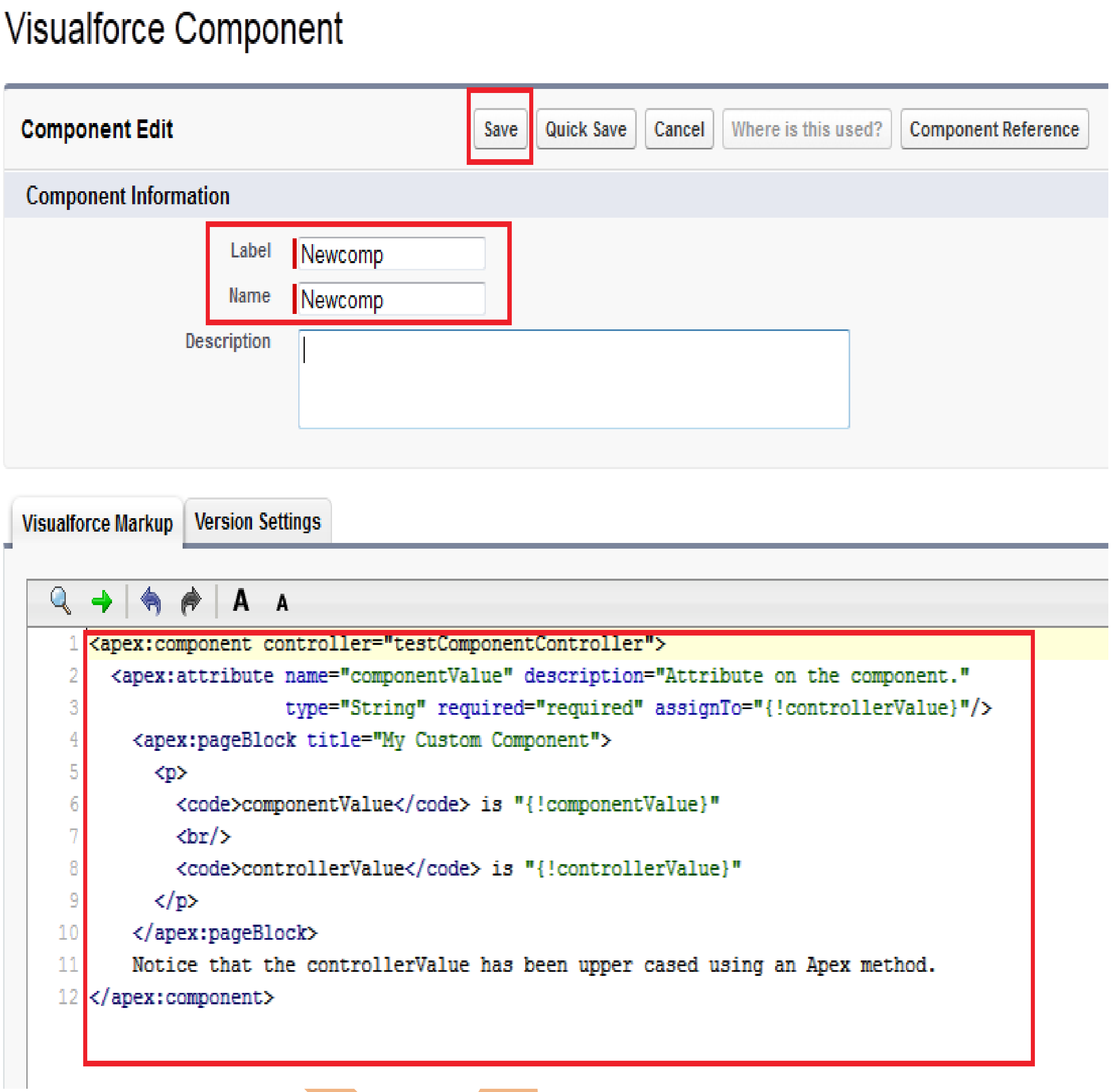
In SalesForce you can build your own custom components to augment this library.

Accessing Custom Component Attributes in a Controller :
Step 1:
Create Apex class :



```
public class testComponentController {

  public String controllerValue;

  public void setControllerValue (String s) {
    controllerValue = s.toUpperCase();
  }

  public String getControllerValue() {
    return controllerValue;
  }
}
```

Step 2 :
Create Component, how to create Component in SalesForce.com.
Setup > **Develop** > **Components**, Then click **New** button.



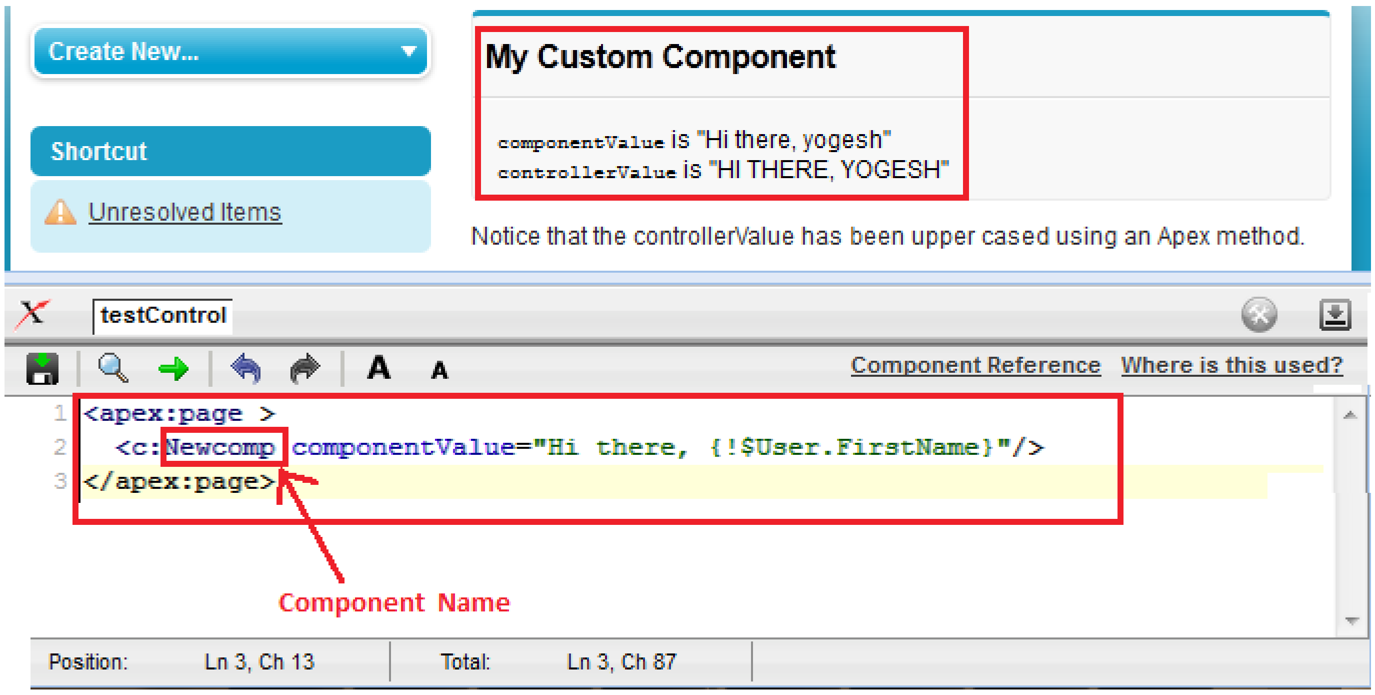
Label: Newcomp

Name: Newcomp

Description:

```
<apex:component controller="testComponentController">
  <apex:attribute name="componentValue" description="Attribute on the component."
                  type="String" required="required" assignTo="{!controllerValue}"/>
    <apex:pageBlock title="My Custom Component">
      <p>
        <code>componentValue</code> is "{!componentValue}"
        <br/>
        <code>controllerValue</code> is "{!controllerValue}"
      </p>
    </apex:pageBlock>
    Notice that the controllerValue has been upper cased using an Apex method.
</apex:component>
```

Step 3 :
Create VisualForce page.



My Custom Component

componentValue is "Hi there, yogesh"
controllerValue is "HI THERE, YOGESH"

Notice that the controllerValue has been upper cased using an Apex method.

```
<apex:page >
  <c:Newcomp componentValue="Hi there, {!$User.FirstName}"/>
</apex:page>
```

Position: Ln 3, Ch 13 Total: Ln 3, Ch 87