Contents

Hello world with external css and js

Go to quick find and search for visualforce. Go to visualforce pages.

Create new page.

```
<apex:page>
  <apex:includeScript value="{! $Resource.jquery331 }"/>
  <apex:stylesheet value="{!URLFOR($Resource.AgentDemoCSS)}"/>
  <apex:includeScript value="{! $Resource.AgentDemoJS }"/>
  <script>
  alert('here');
  </script>
</apex:page>
```

Show list of records

To show list of records you need to use a controller.

Notice the additional code. We have defined record set var in apex:page. This var is used in the loop apex:repeat.

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

  <apex:includeScript value="{! $Resource.jquery331 }"/>
  <apex:stylesheet value="{!URLFOR($Resource.AgentDemoCSS)}"/>
  <apex:includeScript value="{! $Resource.AgentDemoJS }"/>

  <apex:pageblock>
    <apex:repeat var="a" value="{!AccountsList}" rendered="true"  id="account_list">
      <li>
        <apex:outputLink value="/{!a.ID}" >
          <apex:outputText value="{!a.Name}"/>
        </apex:outputLink>
      </li>
    </apex:repeat>
  </apex:pageblock>

</apex:page>
```

Save your code and preview to check your work.

Show one record detail view

Now if you click on the account name in above page, it takes you to standard detail view of account record. We will create a custom detail view for account record. This page will get the record id from URL, pull its details and display in visualforce.

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

  <apex:includeScript value="{! $Resource.jquery331 }"/>
  <apex:stylesheet value="{!URLFOR($Resource.AgentDemoCSS)}"/>
  <apex:includeScript value="{! $Resource.AgentDemoJS }"/>

  <apex:pageBlock title="Record Detail Page">
  <apex:detail subject="{!Account.Id}" relatedList="false" title="false"/>
```

```
    </apex:pageBlock>

</apex:page>
```

Above code will display detail view using standard layout. To implement your own layout replace apex:detail tag with your code such as <apex:outputText>{!Account.Name}</apex:outputText>

Record edit view

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

    <apex:includeScript value="{! $Resource.jquery331 }"/>
    <apex:stylesheet value="{!URLFOR($Resource.AgentDemoCSS)}"/>
    <apex:includeScript value="{! $Resource.AgentDemoJS }"/>

    <apex:pageBlock title="Record Detail Page">
    <apex:form>
    <apex:inputField value="{!Account.Name}" />
    <apex:commandButton action="{!Save}" value="Save and return to standard detail page" />
    </apex:form>
    </apex:pageBlock>

</apex:page>
```

Above submit button will save and return to standard detail page. Ideally you would want to return to your custom detail page. To do so change submit button and add extension controller.

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

    <apex:includeScript value="{! $Resource.jquery331 }"/>
    <apex:stylesheet value="{!URLFOR($Resource.AgentDemoCSS)}"/>
    <apex:includeScript value="{! $Resource.AgentDemoJS }"/>

    <apex:pageBlock title="Record Detail Page">
    <apex:form >
    <apex:inputField value="{!Account.Name}" />
    <apex:commandButton action="{!autoRun}" value="Save and return to custom detail page" />

    <apex:selectlist size="1">
      <apex:selectoptions value="{!CustomerPriorities}" />
    </apex:selectlist>

    </apex:form>
    </apex:pageBlock>

</apex:page>
```

CustomSave class:

```
public class CustomSave {

    private ApexPages.StandardController stdController;

    public CustomSave(ApexPages.StandardController stdController) {

        this.stdController = stdController;
```

```
    }

    public List<Selectoption> getCustomerPriorities(){
      List<SelectOption> options = new List<SelectOption>();
      List<Schema.Picklistentry> fieldResult = Account.CustomerPriority__c.getDescribe().getPicklistValues();
      options.add(new SelectOption('', '-- select -- '));
      for(Schema.PicklistEntry f : fieldResult) {
         options.add(new SelectOption(f.getValue(), f.getLabel()));
      }
      return options;
    }

    public PageReference autoRun() {
       stdController.save(); // This takes care of the details for you.
        PageReference v = Page.Account_Detail_View;
        v.getParameters().put('id',stdController.getId());
        v.setRedirect(true);
        return v;
    }
}
```

To save and return to same page use quicksave in submit button action

```
<apex:commandButton action="{!quicksave}" value="Save and return to same page" />
```

Create new record – normal save, ajax based insert

Refer to edit record section above. Both create new record and edit record have same html.

VF components

Component is a code block to accomplish particular functionality. The component can be used in multiple pages.

Creating component: quick find visualforce and click on components. Create new.

```
<apex:component>
   <p>some HTML here</p>
</apex:component>
```

Paste above code and save new component as mycomponent

Using a component in page:

```
<apex:page>

 <c:mycomponent />

</apex:page>
```

Controllers – standard, custom, extensions

Consider records listing page. It shows how we define standardController and use it to list records in a loop.

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

  <apex:pageblock>
    <apex:repeat var="a" value="{!AccountsList}" rendered="true"  id="account_list">
      <li>
        <apex:outputLink value="/{!a.ID}" >
          <apex:outputText value="{!a.Name}"/>
        </apex:outputLink>
      </li>
    </apex:repeat>
  </apex:pageblock>

</apex:page>
```

Similarly you can use a custom object here like <apex:page standardController="Employee__c" recordSetVar="EmpList">

Custom controller – it is called by using controller attribute like
<apex:page controller="GetEmployees" recordSetVar="EmpList">
Note that the GetEmployees is an apex class here.

Extension – these are used to override standard functionality of the controller object like instead of using standard {!save} feature in command button you can define {!mysave} by wiring an extension class e.g.

<apex:page standardController="Account" extensions="CustomSave">
Here CustomSave is apex class name and mysave is the method inside this class.

Search functionality

<apex:page controller="theController">
  <apex:form >
  <apex:pageBlock mode="edit" id="block">
 	
  <apex:pageBlockSection >
  <apex:pageBlockSectionItem >
	public class theController {
	
	  String searchText;
	  List<Lead> results;
	
	  public String getSearchText() {

```
          <apex:outputLabel for="searchText">Search
Text</apex:outputLabel>
          <apex:panelGroup >
            <apex:inputText id="searchText"
value="{!searchText}"/>
            <apex:commandButton value="Go!"
action="{!doSearch}" rerender="block" status="status"/>
          </apex:panelGroup>
        </apex:pageBlockSectionItem>
      </apex:pageBlockSection>

      <apex:actionStatus id="status"
startText="requesting..."/>
      <apex:pageBlockSection title="Results" id="results"
columns="1">
        <apex:pageBlockTable value="{!results}" var="l"
rendered="{!NOT(ISNULL(results))}">
          <apex:column value="{!l.name}"/>
          <apex:column value="{!l.email}"/>
          <apex:column value="{!l.phone}"/>
        </apex:pageBlockTable>
      </apex:pageBlockSection>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

```
    return searchText;
  }

  public void setSearchText(String s) {
    searchText = s;
  }

  public List<Lead> getResults() {
    return results;
  }

  public PageReference doSearch() {
    results = (List<Lead>)[FIND :searchText RETURNING
Lead(Name, Email, Phone)][0];
    return null;
  }

}
```