Custom Controller

Custom Controllers

→As the name suggests customer controller does all the hard work by itself and does not use the Standard controller logic

→We saw how standard controller helps in performing standard actions like *Save*, *Edit*, *New* etc. but if you need new custom actions, custom navigations, HTTP callouts etc. we can go for custom controller

→If you decide to use a custom controller, you need to write the controller class in Apex unlike standard controller where the platform takes care of all the actions and functionality

→Creation of custom controller involves the following steps:

- » Create a custom controller VF page
- » Create a custom controller Apex class
- » Create a method inside the controller class to retrieve data and display on your VF page - Getter
- » Create a method inside the controller class to save the data (that responds to the action performed by user on the VF page) - Setter
- » Create navigation methods
- » Create methods for sorting action

→Following methods are available in controllers:

- » Getter and Setter
- » Action methods

Custom Controller labs

→Let's build a simple custom controller using Apex and a VF page that will consume this controller

→Create a new Apex class called " studcustomcontroller " from Developer Console or Eclipse

→This class helps in retrieving data of students from " studentmaster " object. Please ensure you have some records and also use LIMIT statement or WHERE clause to reduce number of records

→Please note that this does not have exception handling. This is just a demo about custom controller. There is a small bug in the code below. Try to troubleshoot.

```
public class studcustomcontroller {

public List<studentmaster__c> getstudents() {
    //getstudents is the getter method. In the VF page refer to this as students dont use getstudents
    List<studentmaster__c> students  = Database.query(
'SELECT Id, Department__c,,NumofSubjectsPassed__c,NumSubjectFailed__c,AverageMarek__c FROM studentmaster__c
    return students;
}
}
```

→Important point to understand is how the customer controller written in apex and the VF page are linked

→First point of linkage happens in the <apex:page controller="studcustomcontroller" > in the VF page studcustomcontroller

→Second point of linkage happens in the line# 3 where you are using {!stduents}" which refers to the getter method defined in the class, "get students"

```
1  <apex:page controller="studcustomcontroller">
2  <apex:pageblock title="Welcome {!$User.FirstName} ">
3          <apex:pageBlockTable value="{! students}" var="studmast">
4                  <apex:column value="{!studmast.Department__c}"/>
5                 <apex:column value="{!studmast.NumofSubjectsPassed__c}"/>
6                      <apex:column value="{!studmast.NumSubjectFailed__c}"/>
7                      <apex:column value="{!studmast.AverageMarsk__c}"/>
8          </apex:pageBlockTable>
9  </apex:pageblock>
10 </apex:page>
```

Controller extensions

→Controller extensions help extend functionality provided by standard and custom controllers

→You can selectively override one or more actions of standard controller such as Edit, View, Save or Delete

→Add new actions apart from Edit, View, Save, Delete

→Controller extensions can be used if there is a requirement to implement custom functionality

→To include a controller extension in a standard controller use the following syntax in the VF page

```
<apex:page standardController="studentmaster__c" extensions="studmastextension">
```

→To include a controller extension in a custom controller use the following syntax in the VF page

```
<apex:page coontroller="customcontroller extensions="studmastextension">
```

→You can have more than one controller extensions :

```
<apex:page standardController="studentmaster__c" extensions="studmastextension1",studmastextension2>
```

Sharing in controllers

	System Mode	System Mode
Standard Controller	Yes	Yes(Controller)
Custom Controller	Yes	Yes
Controller Extensions	Yes	No

→You want to build a Visualforce page that respects user permissions. Although a controller extension class executes in system mode, if a controller extension extends a standard controller, the logic from the standard controller does not execute in system mode. Instead, it executes in user mode, in which permissions, field-level security, and sharing rules of the current user apply

→Although custom controllers and controller extension classes execute in system mode and thereby ignore user permissions and field-level security, you can choose whether they respect a user's organization-wide defaults, role hierarchy, and sharing rules by "Using the with sharing keywords" in the class definition. For information, see "Using the with sharing or without sharing Keywords" in the Force.com Apex Code Developer's Guide

Extensions :Labs

→Create a new class " studmastextension " as shown below and define a method called "getWelcomeMsg"

→This will be the custom action or message that will be displayed on the page

→After making changes mentioned below, associate the VIEW button to the Buttons, Links, and Actions of the studentmaster__c object

→Make a copy of the custom controller we built in the previous module and name it as studentstandarCplusextension" and make the below change

```
<apex:page standardController="studentmaster__c" extensions="studmastextension">
```

```
public class studmastextension {
    private final studentmaster__c studmast;
    public studmastextension(ApexPages.StandardController stdController)
    {
    this.studmast = (studentmaster__c)stdController.getRecord();
    }
    public string getWelcomeMsg ()
    {
        return 'Welcome' + studmast.name + 'to the Cloud University' ;
    }
}
```

Error handling in vf pages

* **APEX:MESSAGE**
  It is used to display an error on only a very specific field.
* <apex:outputLabel value="Test String" /><apex:inputField value="{!test.Name}" id="test"/>
*
<apex:message for="test"/>
* :all adderror messages are displayed
* **<apex:messages /> :**All errors are displayed,including valdation errors.
* <**apex:pageMessages** /> Custom error handling-see line below
* **ApexPages.Message**
  `myMsg = new ApexPages.Message(ApexPages.severity.ERROR,'text'`
* <**apex:pageMessage** summary="This is a pageMessage" severity="error" strength="1"/>:Always appears on the screen,permanent message

External resource in VF

* Add as static resource first
* Single file :<apex:includeScript value="{!$Resource.MyJavascriptFile}"/>
* Multiple file :<apex:includeScript value="{!URLFOR($Resource.LibraryJS, '/base/subdir/file.js')}"/>
* Apex:stylesheet
* Directly add CDN
* <Script src="cdn link"/> example jquery

Pagination using controllers

* Salesforce provides ApexPages.StandardSetController Class to get controllable record views
* apcontactset = new ApexPages.StandardSetController(Database.getquerylocator(str));
* apcontactset.setpagesize(xx);
* contactlist =new List<contact>();
* contactlist = (List<contact>)apcontactset.getRecords();
* Using Limit and offset to control queries

Calling apex from javascript button

* {!REQUIRESCRIPT("/soap/ajax/14.0/connection.js")} // declare the Js connection

* {!REQUIRESCRIPT("/soap/ajax/14.0/apex.js")} // declare the Js apex

* {!REQUIRESCRIPT("/js/dojo/0.4.1/dojo.js")} // declare the Js dojo
* var returnFlag = ''; var ids='';
* returnFlag = sforce.apex.execute("CheckClass" , "methode ", {ids:ids});

Calling apex using js remoteaction

* Remote action function in salesforce allows user to access any method from any class through javasrcipt methods, and get the result as a javascript object for further manipulation.
* Points to remember while implementing remote action function:
* Remote action method should have **@RemoteAction** annotation.
* The method should also be **Global** and **Static**

Action Function

* Embedding script in vf pages and use action function
* Action function is a component that provides support for invoking controller action methods directly from JavaScript code using an AJAX request. An <apex:actionFunction> component must be a child of an <apex:form> component
* Can bypass validation rules if immediate attribute set to true,not recommended

AJAX support in VF

* Action support is added for event handling of fields in side a block
* Used along with actionregion/block render attributes
* Used to refresh only section of pages or blocks,improves performance

View State and performance

* Add transient keyword before variables, like lists used just for printing the tables.
* Declare some variable as static if possible, create static code block to recreate them in each request.
* Check your SOQL queries, if you are querying additional fields, never used or required on visualforce page.

View State

* All non-transient data members in the associated controller (either standard or custom) and the controller extensions.
* The component tree for that page, which represents the page's component structure and the associated state, which are the values applied to those components.
* A small amount of data for Visualforce to do housekeeping and to maintain state of stateless http request.
* View state data is encrypted and cannot be viewed with tools like Firebug/wireshark. The view state inspector described below lets you look at the contents of view state.

Concept of test class and code coverage

→Unit test methods help determine if the code works as expected on a standalone basis

→Test methods do not take arguments and do not commit anything to the database

→Test methods do not usually access existing data in the database or make changes do the existing data

→Test methods create their own data, perform the tests and automatically the data is cleaned up or not saved!

→Apex requires that you have at least 75% code coverage of all your code before you can deploy code from one environment to another

→Testing DML and Triggers need some understanding of few more methods available as part of the testing framework

→Test.Start and Test.Stop help in ensuring the test methods get separate Governor limits (you will be learning about Governor limits soon)

→We have a trigger on the student_master__c object when you add a new student or update existing student that posts a Welcome message on the field 'welcome__c';

→We define a class that is called by the Trigger

→Test class is shown in the next slide that simulates an insert of a new student record and then verifies if the trigger has set the welcome field – 'welcome__c'

```
1  public class WelcomeStudent2Class {
2  //Create the class first
3
4     //ensure your custom object name is mentioned instead of student_master__c
5
6     public static void welcome2class(List<student_master__c> students){
7     //ensure your custom object name is mentioned instead of student_master__c
8
9        for (student_master__c stud:students){
10
11    //ensure your custom field name is mentioned instead of welcome__c
12          stud.welcome__c = 'Hello Welcome';
13
14       }
15    }
16 }
```

```
1trigger welcomestudent on student_master__c (before insert, before update)
2 {
3    //ensure your custom object name is mentioned instead of student_master__c
4//Create this trigger after creating the class WelcomeStudent2Class
5//declare a List
6List<student_master__c> studentlist = Trigger.new;
7
8WelcomeStudent2Class.welcome2class (studentlist);
9}
```

Sample test class

```
@isTest
private class TestWelcomeTrigger
{
  @isTest static void TestWelcome()
  {
  //Insert a new student master record
  //Using Test.startTest and stoptest ensure you get separate governor limits for the lines of code between them
  string foundstudent ='DIDNOTFIND';
student_master__c stdmast = new student_master__c (Name ='Test Student', studentEmail__c = 'test@test.com', StudentPhone__c = '9999999999',StudentCity__c='NewYork');
  Test.startTest();
Insert stdmast ;
student_master__c studentres = [SELECT Id, Name, Department__c,studentEmail__c,welcome__c FROM student_master__c WHERE  studentEmail__c = 'test@test.com' and name='Te
    Test.stopTest();
  //Check if the record was inserted and the trigger set the field welcome__c to Hello welcome as per expectations
   if (studentres.name == 'Test Student' && studentres.studentemail__c == 'test@test.com' && studentres.welcome__c=='Hello Welcome')
   //if (studentres.welcome__c=='Hello Welcome')
   {
    foundstudent = 'FOUNDSTUDENT';
   }

   system.assertequals ('FOUNDSTUDENT',foundstudent);


  }

}
```

@istest options

* @isTest static void TestSomething() { ... }
* //instead of this: static void testMethod TestSomething() { ... }

* Only method can be annotated as @istest,that class can have
  Other normal non test methods

Don’t mix testclass and apex controllers for best practise

Keep both logically separate for better control over project

Testing extension controller

* Test.startTest();
* ApexPages.StandardController std = new ApexPages.
* standardController(acct);
* StandardControllerExtension sce = new
* StandardControllerExtension(std);
* // Change rating, then save
* sce.acct.Rating = ‘Hot’;
* sce.save();
* Test.stopTest();

Testing Custom Controllers

* Test.startTest();
* PageReference pr = Page.MyVisualforcePage;
* Test.setCurrentPageReference(pr);
* pr.getParameters().put('id', acct.Id);
* CustomController cc = new CustomController();
* // Change rating then save
* cc.acct.Rating = 'Hot';
* cc.save();
* Test.stopTest();

Apex:components

* Build resuable UI components in your VF page
* Apex:component is the enclosing tag
* Used as <c:componentname attributes =“value”>
* Component can define attributes to get data from main vf page
* Can have controllers/extension- no standard controllers
* Used mainly for reusable ui components with branding and style class similar across org
* All vf tags work inside components

SeeAlldata annotation

* @istest(SeeAlldata=True) opens data access of org to test class ,to be avoided.
* If a test class is defined with the @isTest(SeeAllData=true) annotation, this annotation applies to all its test methods whether the test methods are defined with the @isTestannotation or the (deprecated) testMethod keyword.
* The @isTest(SeeAllData=true) annotation is used to open up data access when applied at the class or method level. However, if the containing class has been annotated with@isTest(SeeAllData=true), annotating a method with @isTest(SeeAllData=false) is ignored for that method. In this case, that method still has access to all the data in the organization. Annotating a method with @isTest(SeeAllData=true) overrides, for that method, an @isTest(SeeAllData=false) annotation on the class.
* @istest(SeeAlldata=True)

Wrapper classes in vf pages to render unrelated object

*

```
*       public static List<User> createUsers( Integer numToCreate, String profileName, String employeeNumber, String userName, Boolean doInsert )
*         {
*           String profileId = [ SELECT Id FROM Profile WHERE Name = :profileName LIMIT 1 ].get(0).Id;

*           List<User> testUsers = new List<User>();
*           for( Integer i = 0; i < numToCreate; i++ )
*           {
*             User aUser = new User();
*             aUser.TimeZoneSidKey = 'America/New_York';
*             aUser.LocaleSidKey = 'en_US';
*             aUser.EmailEncodingKey = 'ISO-8859-1';
*             aUser.ProfileId = profileId;
*             aUser.LanguageLocaleKey  = 'en_US';
*             aUser.Department = '741';
*             aUser.IsActive = true;

*             String uName = userName + '(' + i + ')';
*             String uEmail = uName + '@apexunittest.com';
*             aUser.CommunityNickname = uName + i;
*             aUser.LastName = uName;
*             aUser.Alias = 't' + i;
*             aUser.Email = uEmail;
*             aUser.Username = uEmail;
*             aUser.EmployeeNumber = employeeNumber;
*             testUsers.add( aUser );
*           }

*           if( doInsert )
*             insert testUsers;

*           return testUsers;
*         }
```