08_10 Using ColdFusion Components

In this exercise, you will use the components that you have created in the previous exercise.

In this exercise, you will

- Create an instance of the `newsService.cfc` component on the `news.cfm` page.
- Recreate the needed variables on the `news.cfm` page using the component's methods.
- Recreate the `comePlayWithUs.cfm` page using an instance of the `userService.cfc` component.
- Recreate the `profile.cfm` page using an instance of the `userService.cfc` component.

Create an instance of the newsService.cfc component on the news.cfm page

1. Close every open file and open the '`news.cfm`' page. If needed, you can also open the '`newsService.cfc`' component for reference while working on the page.

The '`news.cfm`' page is similar to the '`agenda.cfm`' page in that both pages accommodates different 'states'. That being said, the '`news.cfm`' page is a bit more complex than the '`agenda.cfm`' page because it has three 'states' :

- If no parameter exists in the URL, the page displays all the news.
- If a '`newsID`' parameter exists in the URL scope, the page has to display a single news.
- If a '`year`' parameter exists in the URL scope the page has to display a list of news for a that specific year.

Every state also displays the list of years with news in the side bar. You will now take care of each of those situations one by one.

2. At the very top of the code, create an instance of the '`newsService.cfc`' component. This instance will be used numerous times throughout the page.

3. Don't forget the commenting best practices and add some meaningful comments

Your code should look like the following

```
<!--- Create an instance of the 'newsSrvice.cfc' component --->
<cfset newsService = createObject('component',
  'cfTraining.components.newsService') />
  <!---Get news years--->
  code continues...
```

Recreate the needed variables on the news.cfm page using the component's methods

Now that you have created an instance of the '`newsService.cfc`' component, you can use the methods define in the component.

1. Locate the `<!---Get news years--->` comment.
2. On the very next line, use `<cfset />` to create the '`rsNewsYears`' local variable. Initialize it to the value returned by the '`getNewsYears`()' method of the '`newsService.cfc`' component.

Your code should look like the following

```
<!---Get news years—>
<cfset rsNewsYears = newsService.getNewsYears() />
```

3. Now, locate the `<!---Get all news--->` comment
4. Use `<cfset />` on the line that follows the comment to create the '`rsAllNews`' local variable. Initialize it to the value returned by the '`getLatestNews()`' method of the '`newsService.cfc`' component.

Your code should look like the following

```
<!---Get all news--->
<cfset rsAllNews = newsService.getLatestNews() />
```

After writing these 3 lines of code, the first 'state' of the page should be working. Save and run the page to check it out. Do not click on the '`Read More`' links or on the links displayed in the side bar, because you did not fix the remaining 2 states of the page yet.

To fix the second 'state' of the page (the one that displays a single news),

5. Locate the `<!—Output single news --->` comment. On the following line, use `<cfset />` to create the '`rsSingleNews`' local variable.
6. Make this variable equal to the data returned by the '`getNewsByID()`' method of the component. Remember that this method needs the '`newsID`' argument to work properly.
7. Pass the '`url.newsID`' URL parameter as the newsID argument of the method.

Your code should look like the following

```
<!---Output a single news--->
<cfset rsSingleNews= newsService.getNewsByID(url.newsID) />
```

Save and run the page. By adding this single line of code, all the '`Read More`' links should work again.

To fix the links on the side bar :

8. Locate the `<cfelseif isDefined('url.year') >` line of code.
9. On the next line, use `<cfset />` to create the '`rsNewsOfYear`' local variable.
10. Make this variable store the data returned by the '`getNewsForYears()`' method of the '`newsService.cfc`' component. Remember that this method needs the '`year`' as its only required argument to work properly.
11. Pass the '`url.year`' parameter to the method.

Your code should look like the following

```
<cfelseif isDefined('url.year') >
<cfset rsNewsOfYear = newsService.getNewsForYear(url.year) />
```

This single line of code should do the job! Save and run the page. Test every single link, the `news.cfm` page should be fully functional again.

If everything works as expected, close all the pages in ColdFusion Builder.

Recreate the comePlayWithUs.cfm page using an instance of the userService.cfc component.

The next page you will focus on is '`comePlayWithUs.cfm`'. The main page content should already be properly displayed, but this page has an extra-feature that requires your attention.
The side bar of the page has a form that the users can fill to submit their application to the band.

Create an instance of the userService component

You will now update the form processing code so it uses the methods of the '`userService.cfc`' component to process the form data.

1. Open the '`comePlayWithUs.cfm`' page and the '`userService.cfc`' component.
2. At the very top of the '`comePlayWithUs.cfm`' page, use `<cfset />` to create the '`userService`' local variable.
3. Make it equal to an instance of the '`userService.cfc`' component.
4. Don't forget to add a meaningful comment.

Your code should look like the following

```
<!---Create an instance of the userService component--->
<cfset userService =
  createObject("component",'cfTraining.components.userService') />
<!---Form processing script starts here--->
<cfif structKeyExists(form,'fld_newUserSubmit')>
```

Use the 'userService.cfc' component for the server side form validation.

The '`userService.cfc`' component, contains the '`validateUser()`' method. This method handles the server side form validation process. It returns an array and requires 6 arguments. One very important thing to keep in mind is that the '`validateUser()`' method will be used by the '`comePlayWithUs.cfm`' page, but also by the '`profile.cfm`' page.

1. Locate the `<!---generate the missing data--->` comment.
2. Select and cut the comment and the four `<cfset />` statement that follow.
3. Paste that code on the line before the `<!---Server side data validation--->` comment.

Your code should look like the following

```
<!---Form processing script starts here--->
<cfif structKeyExists(form,'fld_newUserSubmit')>
<!---generate the missing data--->
  <cfset form.fld_userPassword = generateSecretKey("AES") />
  <cfset form.fld_userRole = 1 />
  <cfset form.fld_userApproved = 0 />
  <cfset form.fld_userIsActive = 0 />
  <!---Server side data validation--->
```

4. Erase all the code that is between the `<!---Server side form validation—>` comment and the `<cfif ArrayisEmpty(aErrorMessages)>` line.
5. Replace the code you just erased by a `<cfset />` tag that creates the '`aErrorMessages`' variable.
6. Make that variable equal to the value returned by the '`validateUser()`' method.
7. Pass the required arguments to the method.

Your code should look like the following

```
<!---Server side data validation--->
<cfset aErrorMessages =
    userService.validateUser(form.fld_userFirstName,form.fld_userL
    astName,form.fld_userEmail,form.fld_userPassword,form.fld_user
    Password) />
<cfif ArrayIsEmpty(aErrorMessages)>
```

On the above code, notice the six arguments that are passed to the method. All these arguments are taken from the data entered by the user in the form, except for the '`fld_userPassword`' that is automatically generated when the form is submitted. Notice also that when calling the method, the '`fld_userPassword`' variable is used for both the '`userPassword`' and the '`userPasswordConfirm`' arguments.

Use the 'userService.cfc' component to save the new user in the database.

Inserting the user in the database is to be done only if the '`validateUser()`' method returns an empty array. Locate the `<!---Insert data in database if no error detected--->` comment and confirm it is in a `<cfif>` block that tests if the '`aErrorMessages`' array is empty.

1. Just below the comment, delete the `<cfquery>` block that inserts the new user in the database.

2. Replace the deleted query by a `<cfset />` tag that calls the '`addUser()`' method of the '`userService.cfc`' component.
3. Pass the 9 arguments required by the method in the parenthesis.

TIP : If you do not remember the 9 arguments of the method, run the component and look at the automatic documentation for reference.

Your code should look like the following

```
<cfif ArrayIsEmpty(aErrorMessages)>
<!---Insert data in database if no error detected--->
<cfset
    userService.addUser(form.fld_userFirstName,form.fld_userLastNa
    me,form.fld_userEmail,form.fld_userPassword,form.fld_userRole,
    form.fld_userInstrument,form.fld_userComment,form.fld_userAppr
    oved,form.fld_userIsActive) />
<cfset userIsInserted = true />
</cfif>
```

In the code above, notice how the `<cfset />` tag is used even though the query does not return any data.

Finalize the page by calling the 'getInstruments()' method

The last thing to do on the '`comePlayWithUs.cfm`' page is to replace the `<cfquery>` that retrieves the instruments from the database by a call to the '`getInstruments()`' method of the '`userService.cfc`' component.

1. First, delete the query located right below the `<!---Get the instruments list to fill the drop-down menu of the form--->` comment.
2. Replace the deleted query by a `<cfset />` tag that creates the `rsInstrumentsList` variable.
3. Make the variable equal to the data returned by the '`getInstruments()`' method.

Your code should look like the following

```
<!---Get the instruments list to fill the drop down menu of the
form--->
<cfset rsInstrumentsList = userService.getInstruments() />
```

When done, save and run the page. When the page shows in the browser, fill and submit the form. If you did not mistype anything, you should receive a message sating that the form has been correctly submitted.
The '`comePlayWithUs.cfm`' page is now finished. Take some time to look at its code. Thanks to the code-reuse techniques that you have implemented it is much shorter and easier to read than it was before. Notice also that you have instantiated the '`userService.cfc`' component only once at the top of the page, but you have used it multiple times across the page.

Return to ColdFusion Builder and close all the open files.

Update the 'profile.cfm' page so that it uses the 'userService.cfc' component.

The '`profile.cfm`' page is the last one to update in this exercise.

1. Open the '`profile.cfm`' page in ColdFusion Builder.
2. At the very top of the page, use `<cfset />` to create the '`userService`' local variable.
3. Make it equal to an instance of the '`userService.cfc`' component.
4. Don't forget to add a meaningful comment.

Your code should look like the following

```
<!---Create an instance of the user Service component--->
<cfset userService =
  createObject("component",'cfTraining.components.userService') />
<!---Form processing begins here--->
<cfif structKeyExists(form,'fld_editUserSubmit')>
```

5. Just below the `<!---Server side form validation—>` comment, use `<cfset />` to create the '`aErrorMessages`' local variable.
6. Make it equal to the value returned by the '`validateUser()`' method of the '`userService.cfc`' component.
7. Use the values submitted by the user in the form to pass the required 6 arguments to the method.

Your code should look like the following

```
<!---Server side form validation--->
<cfset aErrorMessages =
    userService.validateUser(form.fld_userFirstName,form.fld_userL
    astName,form.fld_userEmail,form.fld_userPassword,form.fld_user
    PasswordConfirm) />
<!---Continue form processing if the aErrorMessages array is
empty--->
```

There are four pieces of data that the user should not be able to modify, so they do not appear in the form. That being said, these four pieces of data are needed by the '`updateUser()`' method of the '`userService.cfc`' component and by the database. These four pieces of data are the '`fld_userID`', the '`fld_userRole`', the '`fld_userIsActive`' and the '`fld_userIsApproved`' fields. You will now add those pieces of data to the form as hidden form fields. This will ensure that the data will not be changed by the user, but will be present in the submitted form anyway.

8. Scroll down the '`profile.cfm`' page and locate the `<!---Submit button--->` comment.
9. Right above the comment, you should find the hidden form field of the '`fld_userID`'.
10. Add three more hidden fields for the remaining pieces of data.

Your code should look like the following

```
<cfinput name="fld_userID" value="#rsUserToUpdate.FLD_USERID#"
    type="hidden" />
<cfinput name="fld_userRole" value="1" type="hidden" />
<cfinput name="fld_userIsActive" value="1" type="hidden" />
<cfinput name="fld_userIsApproved" value="1" type="hidden" />
<!---Submit button--->
```

11. Scroll back up and, locate the `<!---Continue form processing if the aErrorMessages array is empty--->` comment.
12. In the `<cfif>` block that follows, use `<cfset />` to call the '`updateUser()`' method of the '`userService.cfc`' component.
13. Use the data stored in the '`form`' scope to pass the ten required arguments to the method.

Your code should look like the following

```
<!---Continue form processing if the aErrorMessages array is
    empty--->
<cfif arrayIsEmpty(aErrorMessages)>
  <cfset
      userService.updateUser(form.fld_userFirstName,form.fld_userLa
      stName,form.fld_userEmail,form.fld_userPassword,form.fld_user
      Role,form.fld_userInstrument,form.fld_userComment,form.fld_us
      erIsApproved,form.fld_userIsActive,form.fld_userID) />
  <cfset variables.formSubmitComplete = true />
</cfif>
```

The form processing script is now finished, but there are two more lines of code to write for the '`profile.cfm`' page to run smoothly.
The first one retrieves the user to update from the database. Currently, you will hard-code the value of the '`userID`' you want to update. Later in the course, we will update this code to retrieve the data of the currently logged-in user.

14. Locate the `<!---Get user to update--->` comment.
15. Just below the comment, use a `<cfset />` tag to create the '`rsUserToUpdate`' variable.
16. Make that variable equal to the value returned by the '`getUserByID()`' method.
17. Pass user ID n°2 to the method.

Your code should look like the following

```
<!---Get user to update--->
<cfset rsUserToUpdate = userService.getUserByID(2) />
```

The last thing to do is to retrieve the list of instruments from the database to fill the drop-down list of the form with options.

18. Just below the `<!--Get instruments to feed the form's Drop-Down list-->`comment, use a `<cfset />` tag to create the '`rsInstrumentsList`' variable.
19. Make that variable equal to the value returned by the '`getInstruments()`' method of the '`userService`' component.

Your code should look like the following

```
<!---Get instruments to feed the form's Drop-Down list--->
<cfset rsInstrumentsList = userService.getInstruments() />
```

Save and test the page. Try to change some data in the form and, submit it. It should modify the user in the database without generating any error.

If everything works as expected, save and close all the files in ColdFusion Builder.