12_02 The Pages Administration

In this exercise, you will develop the page administration section of your site. This is the most basic area of the admin site because it has to manage only two of the four CRUD operations. In the pages section, the site administrator must be able to

- Read the pages from the database
- Update the existing pages

The Create and the Delete operations are not used in the pages section of the admin site.

Create the necessary '.cfm' pages

The first step of the exercise is to create the necessary pages in the '`admin`' folder of the project.

1. Right click on the '`Admin`' folder and choose '`New->ColdFusion Page`'.
2. In the dialog box, name the page '`pages`' and click on '`Finish`'.
3. Right click on the '`Admin`' folder a second time and choose '`New->ColdFusion Page`' again.
4. Create the '`pageEdit.cfm`' page.
5. When done, you should see two additional pages in the '`admin`' folder.

Now, you will write the starting code of those two new pages using the following steps.

6. In ColdFusion Builder, return to the '`pages.cfm`' file.
7. Call the '`admin.cfm`' custom tag created earlier in the course.
8. Use the '`title`' attribute of the `<cf_admin>` custom tag to create a meaningful page title.
9. In the body of the custom tag, create a `<div id="pageBody">` tag block.
10. In the `<div id="pageBody">` block, Create the title of the page in an `<h1>` tag block.

Your code should look like

```
<cf_admin title="HD-Street band - Pages administration">
    <div id="pageBody">
        <h1>Pages administration</h1>
    </div>
</cf_admin>
```

11. Copy - paste the entire code to the '`pageEdit.cfm`' page.
12. On the '`pageEdit.cfm`' page, change the content of the `<h1>`title.

The code of the '`pageEdit.cfm`' file should look like

```
<cf_admin title="HD-Street band - Pages administration">
    <div id="pageBody">
          <h1>Edit a page</h1>
    </div>
</cf_admin>
```

Your '`.cfm`' pages are now ready. Save and run them to make sure they work as expected. Remember that you must be logged-in as an administrator to access those pages.

Update the 'pageService.cfc' component.

To support the additional operations required by the pages administration, you have to add a few methods to the '`pageService.cfc`' component.

1. Open the '`components/pageService.cfc'` file. There should be only one method defined in that component.
2. To speed up your work and avoid unnecessary typing, open the '`helpers/pageComponent.txt'` file.
3. Copy/paste the entire content of the file to the '`pageService.cfc`' component after the '`getPageByID()`' method, but before the closing `</cfcomponent>` tag.
4. When done, save and run the component to access its automatic documentation.

There should be three additional methods in the component.

- The '`getAllPages()`' method does not take any argument and returns a query containing all the pages present in the database. This method handles the 'Read' operation of the CRUD.
- The '`updatePage()`' method takes three arguments and does not return any value. It handles the 'Update' operation of the CRUD.
- The '`validatePageForm()`' method takes two arguments and handles the server side data validation. It returns an array of error messages.

Carefully review the component's code in ColdFusion Builder. Make sure you understand it before moving on to the next step.

Implement method-level security in the 'pageService.cfc' component.

The '`pageService.cfc`' component should now contains four methods. You will now take advantage of another goodie of the `<cflogin>` and the `<cfloginuser/>` tags. Back in chapter 10, we used those two tags to log the user into the ColdFusion server. Thanks to the `<cfloginuser />` tag, we could define the role for the logged-in user.

Out of the four methods of the '`pageService.cfc`' component, only one (the '`getPageByID()`' method) can be called by any user. The use of the three other methods is reserved for administrative tasks. These methods can be executed only by an administrator.

The `<cffunction>` tag supports the '`roles`' attribute. This '`roles`' attribute is used to define a comma separated list of roles. Only users who are logged in with one of the specified roles can execute the method. If the '`Roles`' attribute is omitted, any user can consume the method.

1. Add the '`roles`' attribute to the opening `<cffunction>` tag that defines the '`getAllPages()`' methods
2. Make it equal to '`Administrator`' so it only the administrators of the site can use the method.

The code is

```
<!--- Get all pages Method --->
<cffunction name="getAllPages" returntype="Query"
    roles="Administrator" >
```

3. Do the same with the '`validatePageForm()`' and with the '`updatePage()`' methods, but do not change the '`getPageByID()`' method as every user must be able to use it.
4. When done, save and run the component. The automatic documentation should reflect your changes.

Cache the new version of the pageService.cfc component in the Application scope

To do so, browse any page of the site, add the '`?restartApp`' url parameter and reload the page. When the page reloads, confirm that the '`Execution Time`' section of the debug output has an entry for the '`onApplicationStart()`' method of the '`Application.cfc`' file.

Develop the READ operation of the pages administration

This operation retrieves all the pages from the database and displays them in a table. There should be an '`Edit this page`' link next to each page.

1. In ColdFusion Builder, return to the '`pages.cfm`' page of the admin site.
2. At the very top of the page, create the '`allPages`' local variable. Make it equal to the query returned by the '`getAllPages()`' method of the '`pageService.cfc`' component cached in the application scope.
3. To make future application maintenance easier, add a meaningful comment.

The code should be the same as

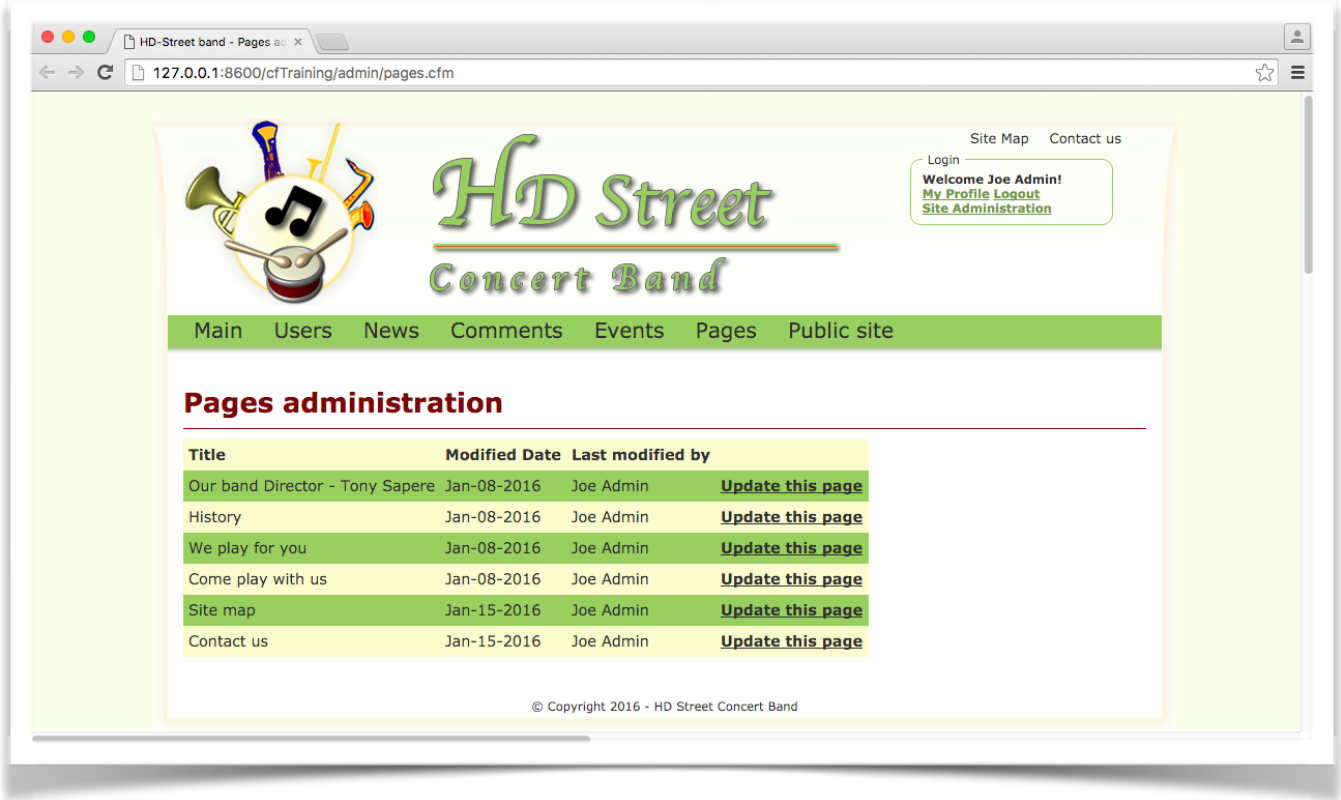
```
<!--- Get all pages from the database. --->
<cfset allPages = application.pageService.getAllPages() />
```

4. Right under the `<h1>` tag block that defines the main page title, add another meaningful comment to announce that you will output all the news in a table.
5. On the next line, define an HTML table of two rows by four columns. Use the `<th>` tag of html to define the cells of the first row and the `<td>` tag for the cells of the second row.
6. Wrap the second row of the table in a `<cfoutput>` block that loops over the '`allPages`' query.
7. Write the column headers in the first row of the table. The headers are '`Title`', '`Modified date`', '`Last modified by`' and '` `'.
8. Output the corresponding query data in the second row of the table.
9. In the last cell of the table, create a link that reads, '`Update this page`'. The link should pass the '`pageID`' url parameter to the '`pageEdit.cfm`' page.

The final code is

```
<!--- Display the list of all pages in an html table --->
<table>
  <tr>
    <th>Title</th>
    <th>Modified Date</th>
    <th>Last modified by</th>
    <th> </th>
  </tr>
    <cfoutput query="allPages">
      <tr>
        <td>#allPages.fld_pageTitle#</td>
        <td>#dateFormat(allPages.fld_pageModificationDate, 'mmm-dd-
          yyyy')#</td>
        <td>#allPages.fld_modifierFirstName#
          #allPages.fld_modifierLastName#</td>
        <td><a href="pageEdit.cfm?pageID=#allPages.fld_pageID#">Update
          this page</a></td>
      </tr>
    </cfoutput>
</table>
```

Save and run the page. It should display a table with a list of all the pages of the site. The last cell of each row should provide a link to the '`pageEdit.cfm`' page. Confirm the link works and the '`pageID`' url parameter is correctly passed to the target page.



Create the update page form on the 'pageEdit.cfm' page

You will now concentrate on the `Update` operation. The '`pageEdit.cfm`' page shall display a form that is pre-populated with the current data of the page whose ID is passed in the URL from the '`pages.cfm`' page. Consequently, a '`pageID`' url parameter is required for the '`pageEdit.cfm`' page to work.

Your first task is to make sure a '`pageID`' url parameter exists. If it does not exist, you should redirect the user to the '`pages.cfm`' file, where he will be able to choose a page to update.

1. In ColdFusion Builder, open the '`pageEdit.cfm`' page.
2. Add a meaningful comment at the very top of the code announcing that you will check if the '`url.pageID`' parameter is correctly passed.
3. Use `<cfif>` and '`structKeyExists()`' to check if a '`url.pageID`' variable exists.
4. In the body of the `<cfif>`, use `<cflocation />` to redirect the user to the '`pages.cfm`' page if '`url.pageID`' does not exist.

The code should be

```
<!--- Confirm url.pageID is passed --->
<cfif NOT structKeyExists(url, 'pageID')>
    <cflocation url="pages.cfm" />
</cfif>
```

Save and run the page. As no '`pageID`' parameter exists in the URL, you should be redirected to the '`pages.cfm`' page. On the '`pages.cfm`' page, click on any '`Update this page`' link and confirm the '`pageEdit.cfm`' page now displays properly.

Your next task is to insert the pages update form below the `<h1>` title of the page.

5. Return to ColdFusion Builder and open the '`helpers/pageForm.txt`' file.
6. Copy/paste the entire content of this file right under the `<h1>` tag block of the '`pageEdit.cfm`' file.

Save and run the page. Confirm that it displays a form with three fields : the '`Page Title`' text filed, the '`Page content`' textarea that displays the Rich Text editor and the '`Update Page`' submit button .

Fill the page update form with data

Now that the form exists on the page, you will pre-fill it with the current data of the chosen page.

First of all, you will call the '`getPageByID()`' method of the '`pageService.cfc`' component to get the current page data and to store it in the '`pageToUpdate`' local variable.

1. Return to the very top of the code, just before the opening `<cf_admin>` tag, but after the `<cfif>` block that checks if the '`pageID`' URL parameter exists.
2. Use a `<cfset />` tag to create the '`pageToUpdate`' variable.
3. Make it equal to the data returned by the '`getPageByID()`' method of the '`pageService.cfc`' component.
4. Pass the '`url.pageID`' parameter as the only argument of the method.

The code is

```
<!---Get current page data--->
<cfset pageToUpdate =
    application.pageService.getPageByID(url.pageID) />
```

Next, you will use the data of the '`pageToUpdate`' variable to pre-populate the form fields.

5. Locate the `<cfinput type="text" />` tag that creates the '`fld_pageTitle`' form field.
6. Add the '`value`' attribute to this tag and make it equal to the '`fld_pageTitle`' field of the '`pageToUpdate`' query.

```
<cfinput name="fld_pageTitle" id="fld_pageTitle"
value="#pageToUpdate.fld_pageTitle#" required="true"
message="Please enter a valid page title" validateAt="onSubmit" />
```

7. Two lines below, locate the `<cftextarea>` tag block.
8. In the body of the `<cftextarea>`block, create a `<cfoutput>` tag block.
9. Output the '`fld_pageContent`' field of the '`pageToEdit`' query in the body of the `<cfoutput>` block.

```
<cftextarea name="fld_pageContent" id="fld_pageContent"
    required="true" message="Please enter a valid page content"
    validateAt="onSubmit" richtext="true" height="500" >
        <cfoutput>#pageToUpdate.fld_pageContent#</cfoutput>
</cftextarea>
```

10. Locate the `<!---Create hidden form field to store page ID--->` comment.
11. On the next line, use `<cfinput type="hidden" />` to create the '`fld_pageID`' form field. Set the value of this field equal to the '`pageID`' url parameter.

```
<!---Create hidden form field to store page ID--->
<cfinput type="hidden" name="fld_pageID" value="#url.pageID#" />
```

Save and run the page. Confirm that the form displays the current data of the page you chose to edit.

Write the form processing script on the 'pageEdit.cfm' file

The last thing to do is to write the form processing script on the '`pageEdit.cfm`' file. As usual, this script will follow these two basic steps :

- Server side form validation.
- Updating the page in the database if the validation succeeds.

1. In ColdFusion Builder, return to the '`pageEdit.cfm` page'.
2. At the very top of the page, just before the `<!--- Get current page data --->` comment, add two meaningful comments to wrap the form processing code.

```
<!--- Begin form processing --->
<!--- End Form Processing --->
```

3. Between those two comments, create a `<cfif>` block that checks if the '`fld_editPageSubmit`' button has been clicked.
4. Inside the `<cfif>` block, create the '`aErrorMessages`' variable.
5. Make it equal to the data returned by the '`validatePageForm()`' method of the '`pageService.cfc`' component.
6. Use the data gathered by the form as the arguments of the method.
7. On the next line, create a `<cfif>` block that checks if the '`aErrorMessages`' array is empty.

8. If the array is empty, call the '`updatePage()`' method of the '`pageService.cfc`' component to save the updated data in the database.
9. When done, redirect the user to the '`pages.cfm?update=true`' page.
10. Save the '`pageEdit.cfm`' page when done.

The final code is

```
<!--- Begin form processing --->
<cfif structKeyExists(form, 'fld_editPageSubmit')>
  <cfset aErrorMessages =
      application.pageService.validatePageForm(form.fld_pageTitle,f
      orm.fld_pageContent) />
  <cfif ArrayIsEmpty(aErrorMessages)>
      <cfset
          application.pageService.updatePage(form.fld_pageTitle,for
          m.fld_pageContent,form.fld_pageID) />
      <cflocation url="pages.cfm?update=true" />
  </cfif>
</cfif>
<!--- End Form Processing --->
```

Notice the '`update=true`' url parameter passed to the '`pages.cfm`' page by the `<cflocation />` tag. It will be used to display a feedback message to the user.

Display a feedback message to the user on the 'pages.cfm' page

This message will be displayed to acknowledge the successful completion of the page update process only if the '`update`' variable is detected in the URL scope.

1. In ColdFusion Builder, return to the '`admin/pages.cfm`' page.
2. Right above the `<!– Display the list of all pages in an html table --->` comment, create a `<cfif>` tag block.
3. Use '`structKeyExists()`' to check if the '`update`' variable is defined in the url scope.
4. In the body of the `<cfif>` block, create a `<p>` tag block.
5. Apply the '`feedback`' css class to the paragraph and write a meaningful feedback message.
6. Save the '`pages.cfm`' file when done.

The code is

```
<!---Display a feedback message to the user--->
<cfif structKeyExists(url, 'update')>
    <p class="feedback">The page has been successfully updated.</p>
</cfif>
```

Test the page administration.

Make sure every file is saved, then, open and run the '`pages.cfm`' page. Make sure you are logged-in as an administrator.

1. Click on any '`Update this page`' link.
2. The '`pageEdit.cfm`' page should open and display the current data of the chosen page.
3. Change the page title and submit the form.
4. You should be redirected to the '`pages.cfm`' page with the feedback message showing.
5. Visit the updated page on the public site to make sure it reflects the changes you just made.

If no error shows up, the page administration if finished!

Next step : 12_03-NewsAdministration.pdf