12_05 The Comments administration

The comments administration is the fourth area of the admin site that you will implement. Basically, the steps to follow are more or less the same as for the previous areas, but there are some major differences also.

- The '`Create`' operation has already been implemented on the public site with the commenting form that appears on the '`news.cfm`' page.
- The administrator must be able to approve a comment submitted by a user, thus creating some kind of a 'Workflow'.
- The administrator must also be able to reject (in other words to delete) a comment, and to modify a submitted comment before or after approval.

Create the necessary pages in the 'admin' folder.

Refer to the following table for details about the pages to create. Remember that each of these two pages has the following basic code.

```
<cf_admin title="title Goes Here">
  <div id="pageBody">
      <h1>main page title goes here</h1>
  </div>
</cf_admin>
```

Page Name	Title attribute	<h1> content
comments.cfm	HD street band - Comments administration	Comments Administration
commentEdit.cfm	HD street band - Comments administration	Update a comment

Once done, test the new pages in the browser. If you named your files correctly, the '`Comments`' link on the main navigation bar of the admin site should now be working.

Add the necessary methods in the 'commentsService.cfc' component

Open the '`commentsService.cfc`' component. It currently contains 3 methods used to implement the commenting feature on the public site. You will now add the methods needed to implement the comments administration.

1. Open the '`helpers/commentsComponent.txt`' file.
2. Copy/paste all of its content in the '`commentsService.cfc`' component after the existing three methods.
3. Save a run the component when done.

You should see six additional methods in the component

- The '`getUnapprovedComments()`' method does not take any argument and returns a query. It is used to display a list of comments that requires the administrator's immediate attention as they have been submitted, but not yet moderated.
- The '`getAllComments()`' method does not take any argument and also returns a query. It is used to display a list of all the exiting approved comments ordered by their related news.
- The '`getCommentByID()`' method takes the '`commentID`' as its only argument and returns a query object. It is used to retrieve the data of a single comment in order to pre-populate the '`edit comment`' form.
- The '`updateComment()`' method takes four arguments and does not return any value. It is used to save the changes made to a comment in the '`commentEdit.cfm`' file.
- The '`approveComment()`' method is a variant of the previous one. It takes the '`commentID`' as its only argument and uses an update query to turn the '`fld_commentApproved`' boolean flag on. This method does not return any value.
- The '`deleteComment()`' method is used to delete a comment. It takes the '`commentID`' of the comment to delete as its only argument and does not return any value. This method is also used to reject a submitted comment.

Carefully review the code of these six methods. Make sure you fully understand it, then, save and run the component. If you can access its automatic documentation, it means there is no syntax errors in the component.

Refresh the ‘commentsService.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.

Display all the comments to approve on the ‘comments.cfm’ page

The ‘`admin/comments.cfm`’ page will display a list of all the comments present in the database grouped by news. It will also show a list of unapproved comments if there are some to display.

1. At the very top of this code, use `<cfset />` to create the ‘`commentsToApprove`‘ variable.
2. Make its value equal to the query returned by the ‘`getUnapprovedComments()`’ method of the ‘`commentsService.cfc`’ component.

```
<!---Get the list of comments to approve--->
<cfset commentsToApprove =
    application.commentsService.getUnapprovedComments() />
```

3. Just below the `<h1>` title, create an <h2> title that reads, ‘`Comments awaiting approval`’.
4. Below the `<h2>`, write a `<cfif> <cfelse>` block that tests if the ‘`commentsToApprove`’ query contains at least one row of data.
5. If it does not contain any data, create a meaningful message to the user in an HTML paragraph.
6. If it contains data, create an HTML table of 7 columns by 2 rows.
7. Wrap the second row in a `<cfoutput>` block that loops over the ‘`commentsToApprove`’ query.
8. In the first row of the table, write the following headers : ‘`Date`’, ‘`Author`’, ‘`Comment`’, ’`News`’, ‘`&nbsp;`’, ‘`&nbsp;`’ a second time and finally, ’`&nbsp;`’ a third time.
9. In the second row of the table, output the corresponding data of the ‘`commentsToApprove`’ query.

10. To enable the administrator to see the news and all its comment in the right context, create a link to the '`news.cfm`' page of the public site in the fourth cell of the table.
11. Pass the '`newsID`' url parameter to display the target page in its '`single news`' state.
12. Use the '`target="_blank"`' HTML attribute to open the link in a new tab/window of the browser.
13. In the last 3 cells of the table, create 3 links that read, '`Approve`', '`Modify`' and '`Reject`'.

The final code is

```
<h2>Comments awaiting approval</h2>
<cfif commentsToApprove.recordCount EQ 0>
  <p>There are no comments awaiting approval at this time.</p>
<cfelse>
  <table>
    <tr>
      <th>Date</th>
      <th>Author</th>
      <th>Comment</th>
      <th>News</th>
      <th>&nbsp;</th>
      <th>&nbsp;</th>
      <th>&nbsp;</th>
    </tr>
    <cfoutput query="commentsToApprove">
      <tr>
      <td>#dateFormat(fld_commentDate, 'mmm-dd-yyyy')#</td>
      <td>#fld_commentAuthor#</td>
      <td>#fld_commentContent#</td>
      <td><a href="../news.cfm?newsID=#fld_newsID#"
        target="_blank">#fld_newsTitle#</a></td>
      <td>Approve</td>
      <td>Modify</td>
      <td>Reject</td>
      </tr>
  </cfoutput>
  </table>
</cfif>
```

Save and run the page. A list of comments awaiting approval should be displayed. If there is no comments to display, return to the public site and submit a comment for the news of your choice. Back to the site admin, don't forget to test the link to the news page of the public site. If everything works as expected, you can safely move on to the next step.

Code the 'Approve' link

In the 'comments.cfm' page, locate the 'Approve' link in the fifth cell of the table.

1. Set the target page of the link to the current page
2. Add the 'approve' url parameter to the link.
3. Set the value of the parameter to the 'commentID' of the current comment

The code is

```
<td><a href="comments.cfm?approve=#fld_commentID#">Approve</a></td>
```

4. Return to the very top of the code.
5. Use <cfif> and 'structKeyExists()' to check if the 'approve' url parameter is present in the URL.
6. If the parameter is detected, use <cfset /> to call the 'approveComment()' method of the 'commentsService.cfc' component.

The code is

```
<!---Approve comments--->
<cfif structKeyExists(URL,'approve')>
    <cfset application.commentsService.approveComment(url.approve) />
</cfif>
```

7. Scroll down the page and place your cursor between the <h1> and the <h2> titles.
8. Use <cfif> and 'structKeyExists()' to check if the 'approve' url parameter is present in the URL.
9. If it is, output a meaningful feedback message to the administrator acknowledging that a comment has been approved.

The code is

```
<cfif structKeyExists(URL,'approve')>
    <p class="feedback">The comment has been approved!</p>
</cfif>
```

Save and run the page. Try to approve a comment. When the page reloads, the comment should disappear from the table and the feedback message should be displayed. If this was the only comment of the table, a message saying that there are no more comments to approve should also be displayed.

Code the ‘Reject’ link

Rejecting a comment is the same thing as deleting a comment. There are three main steps to follow in order to implement this feature :

- Create dynamic links in the ‘`Delete`’ column of the table.
- Create a confirmation mechanism in JavaScript.
- Call the ‘`deleteComment()`’ method of the ‘`commentsService.cfc`’ component to proceed with the actual deletion of the rejected comment.

First, you should create the JavaScript function right below the opening `<cf_admin>` tag. It is the same JavaScript function as for the ‘`News`’ and the ‘`events`’ sections.

```
<script>
function confirmDelete(commentID)
{
  if(window.confirm('Are you sure you want to delete this
      comment?'))
  {
    window.location.href = '/cfTraining/admin/comments.cfm?
        delete='+commentID;
  }
  else
  {
    null;
  }
}
</script>
```

Next, Locate the ‘`Reject`’ link in the last cell of the table. Make it triggers the ‘`confirmDelete()`’ function you just created. Pass the ‘`commentID`’ variable to the function.

```
<td><a href="javascript:confirmDelete(#fld_commentID#);">Reject</
a></td>
```

Finally, at the very top of the code, create a `<cfif>` block to test for the existence of the ‘`delete`’ parameter in the URL. If the parameter is detected, call the ‘`deleteComment()`’ method of the ‘`commentService.cfc`’ component.

```
<!---Delete a comment--->
<cfif structKeyExists(URL,'delete')>
  <cfset application.commentsService.deleteComment(url.delete) />
</cfif>
```

Save and run the page to perform the following tests.

- If needed, go to the public site and submit a comment on the news of your choice.
- Return to the admin site and click on the '`Reject`' link of the new comment.
- Confirm your intention to delete the comment in the JavaScript alert.
- When the page refreshes, the comment should be removed from the table.

Create the 'Modify' dynamic link

Locate the '`Modify`' link in the sixth cell of the table. Make it a link to the '`commentEdit.cfm`' file. Pass the '`commentID`' parameter to the target page.

```
<td><a href="commentEdit.cfm?commentID=#fld_commentID#">Modify</a></td>
```

Save and test the page. If needed, submit a new comment, then click on the '`modify`' link. The '`commentEdit.cfm`' page should load and the '`commentID`' parameter should be present in the URL.

Copy / paste the form on the 'commentEdit.cfm' page.

1. Return to ColdFusion Builder and open the '`commentEdit.cfm`' page.
2. Open the '`helpers/commentsForm.txt`' file and copy/paste all of its content in the '`commentEdit.cfm'` page, right below the `<h1>` tag block.

When done, save and run the page to take a look at the form as displayed in the browser. Then, return to ColdFusion Builder and carefully inspect the code of the form.

Prepare the Edit Comment form on the 'commentEdit.cfm' page

Return to the '`commentEdit.cfm`' page in ColdFusion Builder.

1. At the very top of the page, use `<cfif>` to check if a '`commentID`' parameter exists in the URL.
2. If it does not exist, redirect the user to the '`comments.cfm`' page.

```
<cfif NOT structKeyExists(URL,'commentID')>
  <cflocation url="comments.cfm" />
</cfif>
```

Just below the above <cfif> block, use <cfset /> to create the 'commentToEdit' variable. Make it equal to the query returned by the 'getCommentByID()' method of the 'commentsService.cfc' component.

```
<!--- Get the current data of the comment to edit. --->
<cfset commentToEdit =
    application.commentsService.getCommentByID(url.commentID) />
```

Now, you can populate the form with the data stored in the 'commentToEdit' query. Locate the <!---display form fields---> comment. Just below the comment, there are two form fields to populate.

- The 'fld_commentAuthor' is a standard text field. Use the 'value' attribute of the <cfinput /> tag to populate that field.
- The 'fld_commentContent' is a textarea. Use a <cfoutput> tag block in the body of the <cftextarea> to populate that field.

Finally, just above the <input /> tag that creates the 'fld_commentUpdateSubmit', button, use <cfinput /> to create a hidden form field. Name the field 'fld_commentID' and give it the corresponding value in the 'commentToEdit' query.

Save and run the page. Because the 'commentID' parameter is not present in the URL, you should be redirected to the 'comments.cfm' page. On that page, click on the 'modify' link next to the comment of your choice. The 'commentEdit.cfm' page should reload and display the form filled with the current data of the chosen comment.

If there is no comments to approve, return to the public site a submit a comment for the news of your choice.

Write the form processing script.

Return to the 'commentEdit.cfm' page in ColdFusion Builder.

1. Go back at the very top of the code.
2. Use `<cfif>` to detect if the form has been submitted.
3. Use `<cfset />` to create the 'aErrorMessages' variable.
4. Make it equal to the value returned by the 'validateComment()' method of the 'commentService.cfc' component.

On the public site, one of the validation checks performed by the 'validateComment()' method is to compare the random string displayed in the captcha image to the string entered by the user. In the admin section, this check is irrelevant. There are many approaches to solve this issue. The easiest one is to send dummy data to the method to make sure that this particular test always succeeds.

```
<!---Form processing script begins here--->
<cfif structKeyExists(form, 'fld_commentUpdateSubmit')>
  <cfset aErrorMessages =
      application.commentsService.validateComment(form.fld_commentA
      uthor,form.fld_commentContent,'dummyCode','dummyCode') />
</cfif>
```

Notice the 'dummyCode' data that is used for both the 'securityCodeSystem' and the 'securityCodeUser' arguments. This is how you ensure that this validation step always succeeds.

5. Next, test if the 'aErrorMessages' array is empty or not.
6. If it is empty, call to the 'updateComment()' method of the 'commentService.cfc' component.
7. Use the data gathered by the form as the arguments of the method. The 'commentApprove' argument should have a value of 1.
8. Finally, use `<cflocation />` to redirect the user to the 'comments.cfm?update=true' page.
9. Save the page when done.

```
<cfif ArrayIsEmpty(aErrorMessages)>
  <cfset
      application.commentsService.updateComment(form.fld_commentAut
      hor,form.fld_commentContent,1,Form.fld_commentID) />
  <cflocation url="comments.cfm?update=true" />
</cfif>
```

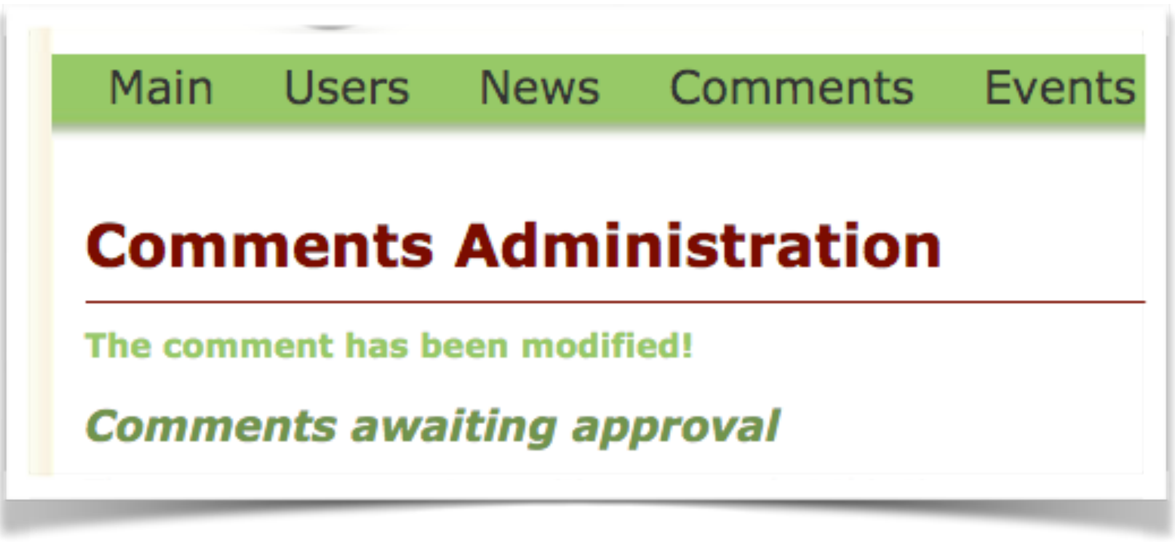
Provide a feedback message on the 'comments.cfm' page

Return to the 'comments.cfm' page and locate the feedback message that acknowledges the approval of a comment.

1. Create another similar <cfif> block to test if the 'update' parameter exists in the URL.
2. If the parameter is detected, display a meaningful feedback message.

```
<cfif structKeyExists(URL,'update')>
  <p class="feedback">The comment has been modified!</p>
</cfif>
```

Save and run the page. Click on the 'modify' link for the comment of your choice. (If there is no comment on the page, go to the public site and submit a new comment for the news of your choice). The 'commentEdit.cfm' page should load and display the current data of the chosen comment. Make some changes to the comment and submit the form. You should be redirected to the 'comments.cfm' page with the feedback message displayed.



Comments Administration

The comment has been modified!

Comments awaiting approval

Display the list of exiting comments

To properly display the list of existing comments, you need to nest two `<cfoutput>` blocks. Since this is a new technique that was not previously covered in the course, displaying the list of existing comments will be covers in a video.

Watch video : Nesting <cfoutput>