08_09 Creating ColdFusion Components

In this long exercise, you will create the components that you will need in your application. You will also transfer all the `<cfquery>` tags you have used so far in the methods of those components.

In this exercise, you will

- Define the methods of the `newsService.cfc` component.
- Create the `eventsService.cfc` component and define its methods.
- Create the `userService.cfc` component and define its methods.

Define the methods of the newsService.cfc component

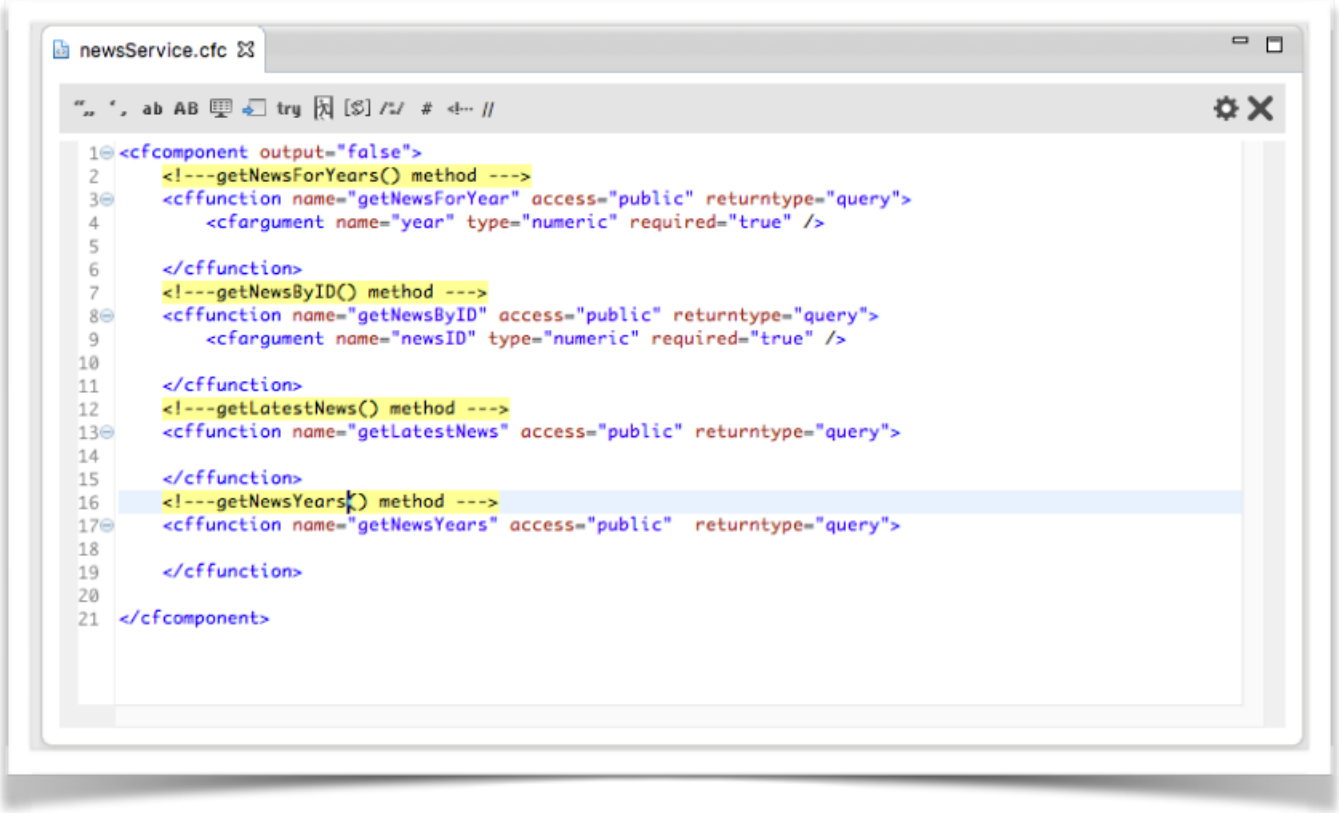
The `newsService.cfc` component has been created in the previous video and should already contain four methods signatures.

> TIP
> It is a good idea to add some meaningful comments in your code.
> I strongly recommend you take the necessary time to add a comment just before each method of the `newsService` component. This will help you spot the various functions of the component later, when more code makes it more difficult to find your way around.

Define the 'getNewsForYears'() method of the 'newsService.cfc' component.

The *New ColdFusion Component* dialog box helped you as much as it could, but now, you have to take over and write some code on your own. Hopefully, most of the code this component requires has already been written earlier in the '`news.cfm`' page. Basically, what you will do in the next few pages is transferring the ColdFusion code present in '`news.cfm`' into your newly created component and made the necessary adjustments.

1. Open the '`news.cfm`' page.



2. Just below the `<!---Get news years--->` comment, select and cut the entire `<cfquery>` block (5 lines of code), but leave the comment as it is.
3. Next, return to the '`newsService.cfc`' component and locate the '`getNewsYears()`' method.
4. Paste the `<cfquery>` block in the body of the '`getNewsYears`' method .
5. Just before the `<cfquery>` tag, use `<cfset />` to create the '`rsNewsYears`' variable. Make this variable local to the function (In other words, *var scope* the variable) and initialize it to an empty string.

Because this variable has the same name as the query, it will store whatever data comes back from the database. Because it is a *var-scoped* variable, it will be discarded as soon as the function will be executed.

6. At the very end of the function, use `<cfreturn />` to return the '`rsNewsYears`' local variable back to the method's caller.

Your code should look like the following

```
<cffunction name="getNewsYears" access="public" output="false"
  returntype="query">
    <cfset var rsNewsYears = '' />
    <cfquery datasource="hdStreet" name="rsNewsYears">
      SELECT YEAR(TBL_NEWS.FLD_NEWSCREATIONDATE) AS fld_newsYear
      FROM TBL_NEWS
```

```
      ORDER BY TBL_NEWS.FLD_NEWSCREATIONDATE DESC
    </cfquery>
    <cfreturn rsNewsYears />
</cffunction>
```

7. Save and run the component. Confirm the automatic documentation displays correctly before moving on to the next step.

Define the 'getLatestNews()' method of the 'newsService.cfc' component.

1. Return to the '`news.cfm`' page. Scroll down to locate the '`rsAllNews`' `<cfquery>` block that follows the `<!---Get all news--->` comment.
2. When spotted, select and cut the entire `<cfquery name="rsAllNews">` (5 lines of code) but leave the comment as it is. The `<cfquery>` block disappears from the page.
3. In the '`newsService.cfc`' component, locate the '`getLatestNews()`' method and paste the `<cfquery>` in the method's body.
4. Just before the opening `<cfquery>` tag, use `<cfset>` to create the '`rsAllNews`' variable. Make this variable local to the function and initialize it to an empty string.
5. Don't forget to add the `<cfreturn />` tag at the very end of the method to return the '`rsAllNews`' query to the function's caller.

Make sure your code looks like the following

```
<!--- getLatestNews() method --->
<cffunction name="getLatestNews" access="public" output="false"
  returntype="query">
    <cfset var rsAllNews = '' />
    <cfquery datasource="hdStreet" name="rsAllNews">
        SELECT FLD_NEWSTITLE, FLD_NEWSID, FLD_NEWSCREATIONDATE
        FROM TBL_NEWS
        ORDER BY FLD_NEWSCREATIONDATE DESC
    </cfquery>
    <cfreturn rsAllNews />
</cffunction>
```

Save and run the '`newsService.cfc`' component. Confirm the automatic documentation displays correctly before moving on to the next step.

Define the 'getNewsByID()' method of the 'newsService.cfc' component.

1. Return to the '`news.cfm`' page and locate the `<!---Output a single news--->` comment. It should be followed by the '`rsSingleNews`' `<cfquery>` tag block that retrieves a single news from the database based on the '`newsID`' parameter present in the URL.
2. Select and cut the entire `<cfquery name="rsSingleNews">`, but leave the comment as it is. The `<cfquery>` block disappears from the page.
3. Return to the '`newsService.cfc`' component and locate the '`getNewsByID()`' method.
4. Paste the `<cfquery>` in this method's body, right after the `<cfargument />` tag.
5. Just before the `<cfquery>` tag, but after the `<cfargument />`, use `<cfset />` to create the '`rsSingleNews`' variable. Make this variable local to the function and initialize it to an empty string.

There is one extra step to take for this one! This function takes '`newsID`' as its only required argument. This argument is used in the `WHERE` clause of the query to uniquely identify the one news to return to the caller.

6. To implement it in the component, simple replace '`url.newsID`' by '`arguments.newsID`' in the `WHERE` clause of the query.
7. At the very end of the function, don't forget to use `<cfreturn />` to return the '`rsSingleNews`' query.

Make sure your code looks like the following

```
<!--- getNewsByID() method --->
<cffunction name="getNewsByID" access="public"
  output="false"returntype="query">
    <cfargument name="newsID" type="numeric" required="true" />
    <cfset var rsSingleNews = '' />
    <cfquery datasource="hdStreet" name="rsSingleNews">
        SELECT TBL_NEWS.FLD_NEWSID,
            TBL_NEWS.FLD_NEWSTITLE,TBL_NEWS.FLD_NEWSCONTENT,
            TBL_NEWS.FLD_NEWSCREATIONDATE,TBL_USERS.FLD_USERFIRSTNA
            ME, TBL_USERS.FLD_USERLASTNAME
        FROM TBL_NEWS INNER JOIN TBL_USERS ON
            TBL_NEWS.FLD_NEWSAUTHOR =TBL_USERS.FLD_USERID
        WHERE FLD_NEWSID = #arguments.newsID#
    </cfquery>
    <cfreturn rsSingleNews />
</cffunction>
```

Save and run the 'newsService.cfc' component. Confirm the automatic documentation displays correctly before moving on to the next step.

Define the 'getNewsForYear()' method of the 'newsService.cfc' component.

This is the last method (for now!) that you will define in this component. This method will retrieve all news from a given year from the database. The year will be passed as the only required argument of the method.

1. Return to the 'news.cfm' page and locate the <cfquery name='rsNewsOfYear'> query.
2. Select and cut the entire query. This removes the query from the page.
3. Return to the 'newsService.cfc' component and locate the 'getNewsForYear()' method.
4. Once spotted, paste the query in the body of the method, just below the <cfargument /> tag.
5. Just before the <cfquery> tag, but after the <cfargument />, use <cfset /> to create the 'rsNewsOfYear' variable. Make this variable local to the function and initialize it to an empty string.
6. Next, update the WHERE clause of the query so it uses 'arguments.year' instead of 'url.year'.
7. At the very end of the function, don't forget to use <cfreturn /> to return the 'rsNewsForYear' query to the function's caller.

Your code should look like the following

```
<!--- getNewsForYears() method --->
<cffunction name="getNewsForYear" access="public"
  output="false"returntype="query">
    <cfargument name="year" type="numeric" required="true" />
    <cfset var rsNewsOfYear = '' />
    <cfquery datasource="hdStreet" name="rsNewsOfYear">
          SELECT FLD_NEWSTITLE, FLD_NEWSID, FLD_NEWSCREATIONDATE
          FROM TBL_NEWS
          WHERE YEAR(FLD_NEWSCREATIONDATE) = #arguments.year#
          ORDER BY FLD_NEWSCREATIONDATE DESC
    </cfquery>
    <cfreturn rsNewsOfYear />
</cffunction>
```

Save and run the 'newsService.cfc' component. Confirm the automatic documentation displays correctly before moving on to the next step.

The 'newsService.cfc' component is finished, for now. If the automatic documentation displays correctly, you can assume that the component's code has no major issues.

8. Save and close every open file.

Create the eventsService.cfc component and define its methods

In this part of the exercise, you will create the eventsService.cfc component. This component will contain the methods you will use to manage the events present in the band's agenda. All the queries that are currently located in the agenda.cfm page must be transferred to this component.

Create the eventsService.cfc component

First, close all the open files in ColdFusion Builder. .

1. In the 'navigator' view of ColdFusion Builder, right-click on the 'components' folder and choose 'New->ColdFusion Component'.
2. In the box that open, name your component 'eventsService'.
3. Set its 'output' to 'false' and click on 'Next->'.
4. Add 2 functions to the component according to the following table

Function name : getCurrentEvents	**Access Type** : public **Output** : false **ReturnType** : query No Argument
Function Name : getEventByID	**Access type** : public **Output** : false **ReturnType** : query **One Argument** • **Name** = eventID • **Type** = numeric • **Required** = Checked

5. Click on the 'Finish'.button when done.

ColdFusion Builder generates the 'eventsService.cfc' file in the 'components' folder and the component should be open in ColdFusion Builder. Make sure you

understand the generated code and don't hesitate to add some meaningful comments before you move on to the next step.

Define the 'getCurrentEvents()' method of the 'eventsService.cfc' component.

1. In ColdFusion Builder, open the '`agenda.cfm`' page and locate the `<cfquery>` tag that follows the `<!---Output the upcoming event table if url.eventID not defined-->` comment. This query retrieves all the future events from the database.
2. Select and cut the entire `<cfquery>` block (normally, 6 lines of code), but leave the comment unchanged.
3. Return to the '`eventsService.cfc`' component and paste the query in the body of the '`getCurrentEvents()`' method.
4. Right before the `<cfquery>` that you just pasted, use `<cfset />` to create the '`rsCurrentEvents`' variable. Make it local to the function and initialize it to an empty string.
5. Don't forget to use `<cfreturn />` at the very end of the function to return the '`rsCurrentEvents`' query to the function's caller.

Make sure your code looks like the following :

```
<!--- getCurrentEvents() method --->
<cffunction name="getCurrentEvents" access="public"
  output="false">
    <cfset var rsCurrentEvents = '' />
    <cfquery datasource="hdStreet" name="rsCurrentEvents">
        SELECT FLD_EVENTID, FLD_EVENTNAME,
            FLD_EVENTDATETIME,FLD_EVENTLOCATION, FLD_EVENTVENUE
        FROM TBL_EVENTS
        WHERE FLD_EVENTDATETIME >= #now()#
        ORDER BY FLD_EVENTDATETIME ASC
    </cfquery>
    <cfreturn rsCurrentEvents />
</cffunction>
```

Save and run the '`eventsService.cfc`' component to confirm there are no errors before moving on to the next step.

Define the 'getEventByID'() method of the 'eventsService.cfc' component.

This second method of the '`eventsService.cfc`' component retrieves a single event from the database. The event to retrieve is the event whose ID matches the '`eventID`' required argument passed to the method.

1. Return to the '`agenda.cfm`' page and locate the `<cfquery>` that follows the `<!---Output a single agenda if url.eventID is defined--->` comment.
2. When spotted, select and cut the entire `<cfquery>` block to remove it from the page, but leave the comment unchanged.
3. Return to the `eventsService.cfc` component.
4. Paste the query in the body of the '`getEventByID()`' function, right below the `<cfargument />` tag.
5. Just before the pasted `<cfquery>`, but after the `<cfargument />` tag, use `<cfset />` to create the '`rsSingleEvent`' variable. Make it local to the function and initialize it to an empty string.
6. Change the `WHERE` clause of the query so it uses '`arguments.eventID`' and not '`url.eventID`'.
7. Don't forget to add `<cfreturn />` at the very end of the function to return the '`rsSingleEvent`' query to the function's caller.

Your code is now the following :

```
<!--- getEventsByID() method --->
<cffunction name="getEventByID" access="public" output="false">
      <cfargument name="eventID" type="numeric" required="true" />
      <cfset var rsSingleEvent = '' />
      <cfquery datasource="hdStreet" name="rsSingleEvent">
          SELECT FLD_EVENTID, FLD_EVENTNAME, FLD_EVENTDATETIME,
              FLD_EVENTLOCATION, FLD_EVENTVENUE,
              FLD_EVENTDESCRIPTION
          FROM TBL_EVENTS
          WHERE FLD_EVENTID = #arguments.eventID#
      </cfquery>
      <cfreturn rsSingleEvent />
</cffunction>
```

The '`eventsService.cfc`' component is finished for now. Save and run it. Confirm that it does not generate any error and that the automatic documentation displays properly before moving on to the next step.

Create the userService.cfc component and define its methods

It is time to create the final component of this chapter. It is going to be a somewhat more complex component, as it will contain more methods and these methods require more arguments to run properly. Technically though, it will not be anything more than the components you have made so far, but you'll have more code to deal with. In such a situation, code commenting is even more important than usual. I strongly recommend you take the necessary time to comment your code during the remaining steps of this exercise.

Create the 'userService.cfc' component.

First, save and close all the open files in ColdFusion Builder.

1. Right click on the '`components`' folder and choose '`New->ColdFusion Component`' in the menu.
2. In the box that open, name your component '`userService`' and set the '`output`' attribute to '`False`'.
3. When done, click 'Next->' to define 5 methods for this component according to the following table.

function name : getUserByID	**Access Type** : Public **Return Type** : Query **Output** : False **Argument 1 : userID** • **Type** = Numeric • **Required** = checked
function Name : getInstruments	**Access Type** : Public **ReturnType** : Query **Output** : False No Arguments

function Name : addUser	**Access Type** : Public **Return Type** : void **Output** : False Argument 1 : **userFirstName** • **Type** = string • **Required** = checked Argument 2 : **userLastName** • **Type** = string • **Required** = checked Argument 3 : **userEmail** • **Type** = string • **Required** = checked Argument 4 : **userPassword** • **Type** = string • **Required** = checked Argument 5 : **userRole** • **Type** = numeric • **Required** = checked Argument 6 : **userInstrument** • **Type** = numeric • **Required** = checked Argument 7 : **userComment** • **Type** = string • **Required** = checked Argument 8 : **userIsApproved** • **Type** = numeric • **Required** = checked Argument 9 : **userIsActive** • **Type** = numeric • **Required** = checked
Function name : validateUser	**Access Type** : Public **Return Type** : Array **Output** : False Argument 1 : **userFirstName** • **Type** = string • **Required** = checked Argument 2 : **userLastName** • **Type** = string • **Required** = checked Argument 3 : **userEmail** • **Type** = string • **Required** = checked Argument 4 : **userPassword** • **Type** = string • **Required** = checked Argument 5 : **userPasswordConfirm** • **Type** = numeric • **Required** = checked

function Name : updateUser	**Access Type** : Public **Return Type** : void **Output** : False Argument 1 : **userFirstName** • **Type** = string • **Required** = checked Argument 2 : **userLastName** • **Type** = string • **Required** = checked Argument 3 : **userEmail** • **Type** = string • **Required** = checked Argument 4 : **userPassword** • **Type** = string • **Required** = checked Argument 5 : **userRole** • **Type** = numeric • **Required** = checked Argument 6 : **userInstrument** • **Type** = numeric • **Required** = checked Argument 7 : **userComment** • **Type** = string • **Required** = checked Argument 8 : **userIsApproved** • **Type** = numeric • **Required** = checked Argument 9 : **userIsActive** • **Type** = numeric • **Required** = checked Argument 10 : **userID** • **Type** = numeric • **Required** = checked

When done, you should see the new '`userService.cfc`' file in the '`components`' folder and the '`userService.cfc`' component should be open in ColdFusion Builder. Take some time to review the generated code. Normally, it should not be a problem for you, even though there is much more lines of code than for the previous components.
It is a very good idea, and a best practice that I strongly encourage, to add some meaningful comments before each of the 5 methods of the component.

Define the 'validateUser()' method of the 'userService.cfc' component.

The `validateUser()` method accepts 5 arguments, one for each of the required fields on the '`frm_editUser`' form of the '`profile.cfm`' page. The method returns an array of error messages if some validation fails. If all data validates correctly, then, the returned array is empty.

1. In ColdFusion Builder, open the '`profile.cfm`' page.
2. Select the form validation script we wrote earlier. The selection begins by the `<!---Server side form validation—>` comment and ends on the line before the `<!---Continue form processing if the aErrorMessages array is empty--->` comment (about 27 lines of code).
3. Double-check your selection.
4. When you are sure that your selection is correct, cut the selected code to remove it from the page, and paste it in the body of the '`validateUser()`' method in the '`userService.cfc`' component after all the `<cfargument />` tags.

You will now adjust this code so that it works as a method of the component. Starting from top to bottom :

5. '`var`' scope the '`aErrorMeassages`' array to make it local to the function
6. Delete the `<cfset />` statement that creates the '`variables.formSubmitComplete`' variable.
7. Update all '`form.fld_userLastName`' so they all read '`arguments.userLastName`' instead.
8. Repeat the previous operation with the remaining form fields : replace all the '`form.`' by '`arguments.`'.
9. At the end of the method, use `<cfreturn />` to return the '`aErrorMeassages`' array.

The resulting code should be identical to the following:

```
<!--- validateUser() method --->
<cffunction name="validateUser" access="public" output="false"
  returntype="array">
  <cfargument name="userFirstName" type="string" required="true" >
  <cfargument name="userLastName" type="string" required="true" />
  <cfargument name="userEmail" type="string" required="true" />
  <cfargument name="userPassword" type="string" required="true" />
  <cfargument name="userPasswordConfirm" type="string" required="true" />
  <cfargument name="userInstrument" type="numeric" required="true" />
  <!--Server side form validation--->
```

```
  <cfset var aErrorMessages = arrayNew(1) />
  <!---Validate firstName--->
  <cfif arguments.userFirstName EQ ''>
      <cfset arrayAppend(aErrorMessages,'Please provide a valid first
        name') />
  </cfif>
  <!---Validate lastName--->
  <cfif arguments.userLastName EQ ''>
      <cfset arrayAppend(aErrorMessages,'Please provide a valid last
        name') />
  </cfif>
  <!---Validate Email--->
  <cfif arguments.userEmail EQ '' OR NOT isValid('email',
    arguments.userEmail) >
      <cfset arrayAppend(aErrorMessages,'Please provide a valid email
  address')/>
  </cfif>
  <!---Validate Password--->
  <cfif arguments.userPassword EQ '' >
      <cfset arrayAppend(aErrorMessages,'Please provide a password ')/>
  </cfif>
  <!---Validate Password confirmation--->
  <cfif arguments.userPasswordConfirm EQ '' >
      <cfset arrayAppend(aErrorMessages,'Please confirm your password
        ')/>
  </cfif>
  <!---validate password and password confirmation Match --->
  <cfif arguments.userPassword NEQ arguments.userPasswordConfirm >
      <cfset arrayAppend(aErrorMessages,'The password and the password
        confirmation do not match')/>
  </cfif>
  <cfreturn aErrorMessages />
</cffunction>
```

Save and run the '`userService.cfc`' component. Confirm there is no errors before moving on to the next step.

Define the 'updateUser'() method of the 'userService.cfc' component.

The '`updateUser()`' method takes 10 arguments, one for each field of the '`tbl_users`' database table. These arguments will be used to create an '`UPDATE`' query that will update an existing user.
Notice that the query changes data in the database, but does not return any recordset. Consequently, the function does not need to return data and the '`returnType`' attribute of the `<cffunction>` tag is set to '`void`'. It also means, that you won't need the `<cfreturn />` tag at the end of the function.

1. Return to '`profile.cfm`' and locate the `<!---Update the user in the database-->` comment. It should be followed by a `<cfquery>` tag block that updates a user in the database.

2. Select and cut the entire <cfquery> block. Then, return to the 'userService.cfc' component and paste the query in the body of the 'updateUser()' function, after all the <cfarguments /> .
3. Update the query so that it uses the data of the 'arguments' scope instead of the data from the 'form' scope.

Because we do not need to add the <cfreturn /> tag and you can consider this function as finished! Your code should look like the following:

```
<!--- updateUser() method --->
<cffunction name="updateUser" access="public" output="false"
  returntype="void">
    <cfargument name="userFirstName" type="string" required="true" />
    <cfargument name="userLastName" type="string" required="true" />
    <cfargument name="userEmail" type="string" required="true" />
    <cfargument name="userPassword" type="string" required="true" />
    <cfargument name="userRole" type="numeric" required="true" />
    <cfargument name="userInstrument" type="numeric" required="true" />
    <cfargument name="userComment" type="string" required="true" />
    <cfargument name="userIsApproved" type="numeric" required="true" />
    <cfargument name="userIsActive" type="numeric" required="true" />
    <cfargument name="userID" type="numeric" required="true" />
    <cfquery datasource="hdStreet">
        UPDATE TBL_USERS
        SET
        FLD_USERFIRSTNAME = '#arguments.userFirstName# ',
        FLD_USERLASTNAME = '#arguments.userLastName# ',
        FLD_USEREMAIL = '#arguments.userEmail# ',
        FLD_USERPASSWORD = '#arguments.userPassword# ',
        FLD_USERINSTRUMENT = #arguments.userInstrument# ,
        FLD_USERCOMMENT = '#arguments.userComment# '
        WHERE
        FLD_USERID = #arguments.userID#
    </cfquery>
</cffunction>
```

Save and run the 'userService.cfc' component. Confirm no error shows up before moving on to the next step.

Define the 'getUserByID()' method of the 'userService.cfc' component.

This method retrieves a single user from the database based on the 'userID' argument passed to the method.

1. Return to the 'profile.cfm' page and locate the <!---Get user to update---> comment. It should be followed by a <cfquery> tag block that retrieves a single user from the database. You will move this query to the 'userService.cfc' component.

2. Select and cut the entire <cfquery> block, then, paste it in the body of the 'getUserByID()' method, right after the <cfargument /> tag.
3. In between the <cfargument /> tag and the <cfquery> tag, use <cfset /> to create a variable local to the function named 'rsSingleUser'. Initialize it to an empty string.
4. Change the name of the query so that the it is stored in the local 'rsSingleUser' variable you just created.
5. Update the WHERE clause of the query so that it uses the data passed in the 'userID' argument, instead of the hard-coded value.
6. Finally, use <cfreturn /> to return the 'rsSingleUser' query to the method's caller.

Your final code should be similar to the following

```
<!--- getUserByID() method --->
<cffunction name="getUserByID" access="public" output="false"
  returntype="query">
    <cfargument name="userID" type="numeric" required="true" />
    <cfset var rsSingleUser = '' />
    <cfquery name="rsSingleUser" datasource="hdStreet">
        SELECT FLD_USERID, FLD_USERFIRSTNAME, FLD_USERLASTNAME,
            FLD_USERCOMMENT, FLD_USEREMAIL, FLD_USERROLE,
            FLD_USERPASSWORD, FLD_USERINSTRUMENT, FLD_USERAPPROVED,
            FLD_USERISACTIVE
        FROM TBL_USERS
        WHERE FLD_USERID = #arguments.userID#
    </cfquery>
    <cfreturn rsSingleUSer />
</cffunction>
```

Save and run the 'userService.cfc' component. Confirm there is no errors before moving on to the next step.

Define the 'getInstruments()' method of the 'userService.cfc' component.

This one is easy! It simply provides an alphabetically sorted list of all the instruments present in the database.

1. Return to the 'profile.cfm' page and locate the <!---Get instruments to feed the form's Drop-Down list---> comment. Select and cut the <cfquery> block that follows. This removes the query from the page.
2. Return to the 'userService.cfc' component.
3. Paste the query in the body of the 'getInstruments()' method.
4. Just before the <cfquery> tag, use <cfset /> to create the 'rsInstrumentsList' variable. Make it local to the function and initialize it to an empty string.

5. At the very end of the function, use `<cfreturn />` to have the method return the '`rsInstruments`' query.

Your code is the following

```
<!--- getInstruments() method --->
<cffunction name="getInstruments" access="public" output="false"
  returntype="query">
    <cfset var rsInstrumentsList = '' />
    <cfquery datasource="hdStreet" name="rsInstrumentsList">
        SELECT FLD_INSTRUMENTID, FLD_INSTRUMENTNAME
        FROM TBL_INSTRUMENTS
        ORDER BY FLD_INSTRUMENTNAME ASC
    </cfquery>
    <cfreturn rsInstrumentsList />
</cffunction>
```

Save and run the '`userService.cfc`' component. Confirm there is no error before moving on to the next step.

Define the 'addUser' method of the 'userService.cfc' component.

The last method of this long exercise takes 9 arguments : one for each of the fields of the '`tbl_users`' database table (except for the '`userID`' field that is automatically generated by the database whenever a new user record is created).
Once again, the code of that method has, for the most part, already been written earlier in this course.

1. Save and close the `profile.fm` page
2. Open the '`comePlayWithUs.cfm`' file.

On the side bar of that file (run it if you do not remember), is a form that the users can fill to mark their interest in joining the band. During the form processing script, we used a `<cfquery>` tag to write an `INSERT` query that saves the user to the database based on the information entered in the form. This is the query that you will now transfer to the component.

3. On the '`comePlayWithUs.cfm`' file, locate the `<!---Insert new user in database—>` comment. Select and cut the `<cfquery>` tag block that follows.
4. Return to the '`userService.cfc`' component and paste the query in the body of the '`addUser()`' method, right after the last `<cfargument />` tag.
5. Update the query so that it uses the data from the '`arguments`' scope and not from the '`form`' scope anymore.

If you take a look at the '`returntype`' attribute of the `<cffunction>`, you'll see that it is set to '`void`'. It means that the function does not return any data, so the `<cfreturn />` tag is not needed.

Your code should be the same as the one below.

```
<!---updateUser() method--->
<cffunction name="updateUser" access="public" output="false"
  returntype="void">
    <cfargument name="userFirstName" type="string" required="true" />
    <cfargument name="userLastName" type="string" required="true" />
    <cfargument name="userEmail" type="string" required="true" />
    <cfargument name="userPassword" type="string" required="true" />
    <cfargument name="userRole" type="numeric" required="true" />
    <cfargument name="userInstrument" type="numeric" required="true" />
    <cfargument name="userComment" type="string" required="true" />
    <cfargument name="userIsApproved" type="numeric" required="true" />
    <cfargument name="userIsActive" type="numeric" required="true" />
    <cfargument name="userID" type="numeric" required="true" />
    <cfquery datasource="hdStreet" >
      INSERT INTO TBL_USERS
      (FLD_USERFIRSTNAME, FLD_USERLASTNAME, FLD_USEREMAIL,
           FLD_USERPASSWORD, FLD_USERCOMMENT, FLD_USERAPPROVED,
           FLD_USERISACTIVE, FLD_USERROLE, FLD_USERINSTRUMENT)
      VALUES
      ('#arguments.userFirstName#', '#arguments.userLastName#',
           '#arguments.userEmail#', '#arguments.userPassword#',
           '#arguments.userComment#', #arguments.userIsApproved#,
           #arguments.userIsActive#, #arguments.userRole#,
           #arguments.userInstrument#)
    </cfquery>
</cffunction>
```

Save and run the component to make sure the component does not generate any error.

Save and close all the open files before continuing!