Chapter 3. Build a JSF Web Site for the GUI Chat Client

The source code of this chapter appears in *package jwg.ch3.chatGUIClient* (the applet) and in the project file *ch3_nodb.zip* (the JSF web site).

To test run:

1. Unzip *ch3_nodb.zip* and build a Java project for it with project name "*ch3_nodb*".
2. Deploy the project to a web server (refer to section 1 for setting up a web server).
3. Copy the *jwg.ch3.chatGUIClient* directory from the main *jwg_code* project to the deployment location "*apache-tomcat/webapps/ch3_nodb*". That is, after the copy you will see a directory *apache-tomcat/webapps/ch3_nodb/jwg.*
4. Start the *jwg.ch2.chatGUI.ServerChatGUI* server.
5. Open a web browser and access the URL *http://localhost/ch3_nodb.*
6. Log in using an arbitrary username, and click "Play". You will be able to access the GUI chat client from the web browser.

Figure 1 illustrates different stages of a client web browser.

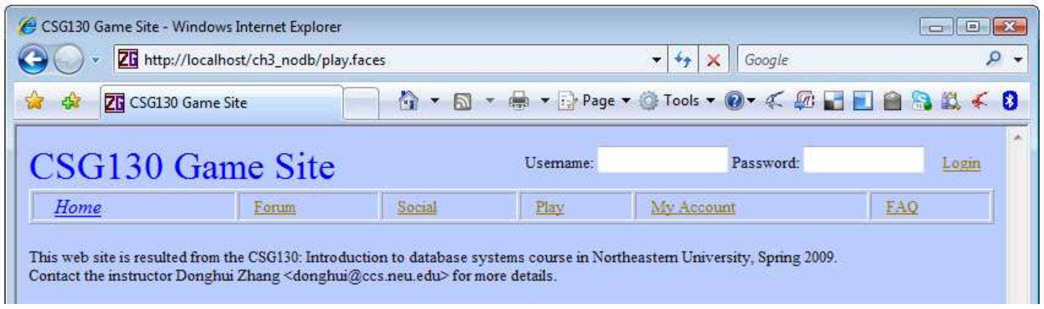
This chapter provides a step-by-step guide to creating this web site from scratch. The main focus is how to work with JSF in MyEclipse. Section 1 illustrates how to set up the Tomcat Apache web server and how to deploy the GUI chat client to the web server. Section 2 shows the basics of constructing a JSF website. Section 3 expands and enriches the JSF project with various techniques such as CSS.

1 Bring the GUI Chat Client to Web

1.1 Set up the Tomcat Apache Web Server

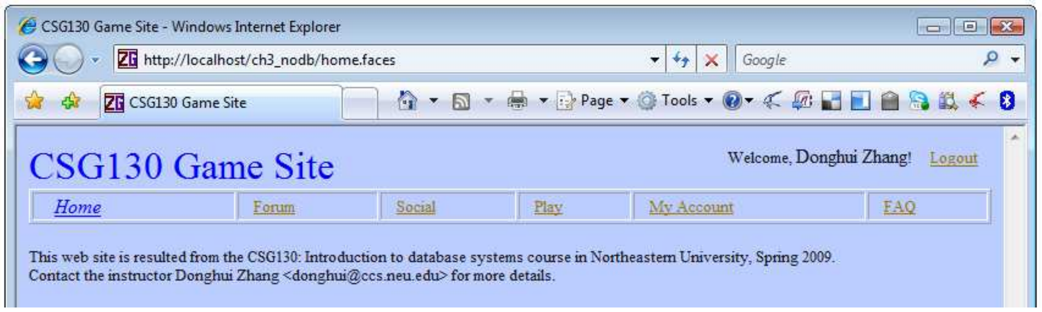
MyEclipse 7.0 ships with Tomcat 6 web server. However you may want to install a separate web server so as to have more control. This section describes how to set up your own Tomcat Apache web server. You should study the MyEclipse Tutorial: "Working with Application Server Connectors", then follow the steps below:

1. Visit *http://tomcat.apache.org.*
2. In the "Download" section, choose the latest version of "Tomcat".
3. Download the zip file under "Binary distributions|Core".
4. Unzip to some place you choose. Let's denote the root directory as "*apache-tomcat*".



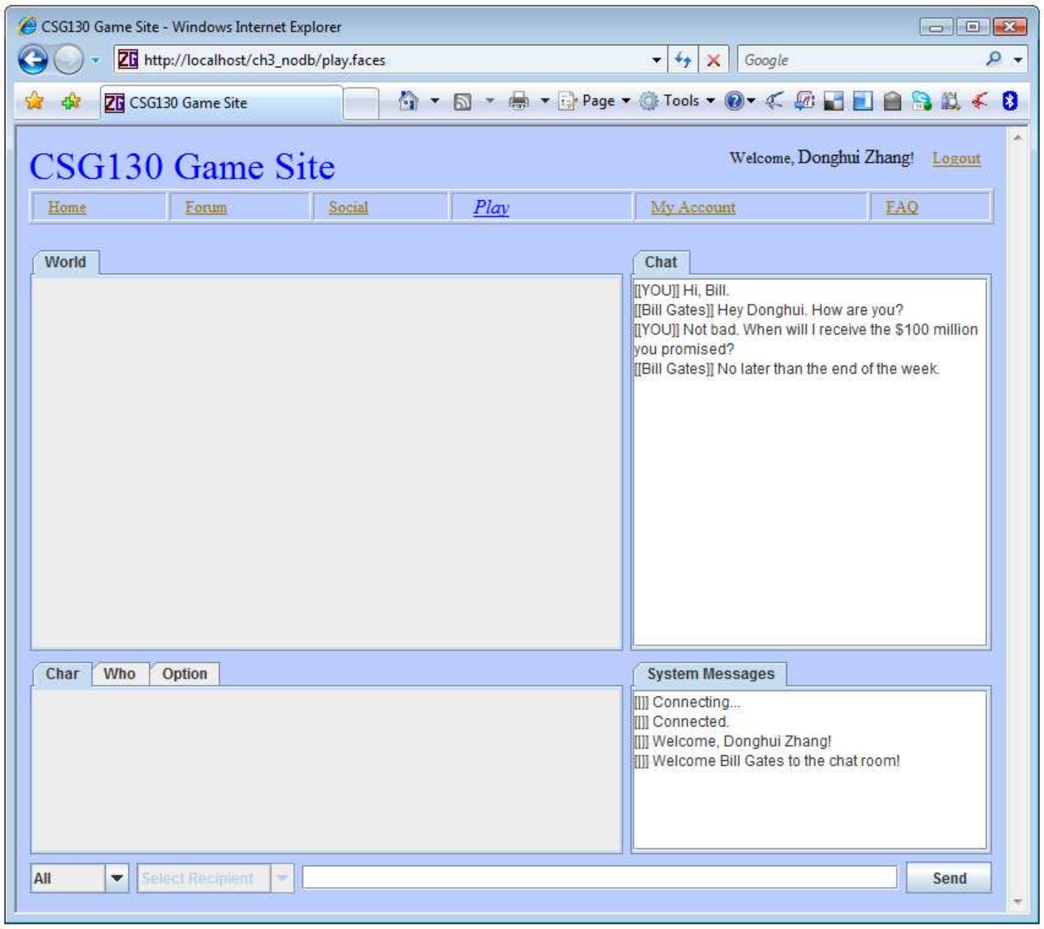
This web site is resulted from the CSG130: Introduction to database systems course in Northeastern University, Spring 2009.
Contact the instructor Donghui Zhang <donghui@ccs.neu.edu> for more details.

(a) the initial page



This web site is resulted from the CSG130: Introduction to database systems course in Northeastern University, Spring 2009.
Contact the instructor Donghui Zhang <donghui@ccs.neu.edu> for more details.

(b) after logging in



[[YOU]] Hi, Bill.
[[Bill Gates]] Hey Donghui. How are you?
[[YOU]] Not bad. When will I receive the $100 million you promised?
[[Bill Gates]] No later than the end of the week.

[[]] Connecting...
[[]] Connected.
[[]] Welcome, Donghui Zhang!
[[]] Welcome Bill Gates to the chat room!

(c) the game page

Figure 1: The GUI client integrated with the JSF web site.

5. Modify the file "*apache-tomcat/conf/server.xml*" by replacing all occurrences of "8080" with "80" (the default HTTP port), so that one does not need to provide the port number in the URL to access the web server.

6. Start the web server: on Windows run *apache-tomcat/bin/startup.bat*; on Unix run *apache-tomcat/bin/startup.sh*.

7. Verify that the server is up, by opening a web browser and visiting the URL "*http://localhost*".

8. To shut down the web server, on Windows run *apache-tomcat/bin/shutdown.bat*; on Unix run *apache-tomcat/bin/shutdown.sh*.

9. Configure MyEclipse to use the newly installed web server as follows: In MyEclipse choose "*Window > Preferences*". Then expand "*MyEclipse Enterprice Workbench > Servers > Tomcat*". Click "*Tomcat 6.x*" (or the latest version if any). And finally provide the root directory of your installed Tomcat web server.

FAQ:

- **Q:** *I'm running Windows. I double clicked "startup.bat" to start the web server. A black window appeared and then disappeared. Yet I do not seem to have access to* http://localhost.

  A: If a black window appeared and then disappeared, the server startup has failed. Try running *startup.bat* from a command prompt to see what the problem is. A common error is that your JAVA_HOME environment variable is not set (to the root directory of your JDK installation). You can set an environment variable by opening the "System Properties" dialog box from the "Control Panel", click the "Advanced" tab, and click "Environment Variables".

1.2 Run the GUI Chat Client in a Web Browser

Please go through the MyEclipse tutorial "Create a J2EE Web Application" to understand all about J2EE web application. Below are some steps to create a JSP page that contains an applet: the GUI chat client.

1. Create a web project.

2. In the "New Web Project" dialog box, type "*ch3_nodb*" as project name, and select "Java EE 5.0". Click "Finish".

3. If you see a dialog box asking you whether you want to use a custom setting for the project, answer "No".

4. In the package explorer window, expand "*ch3_nodb > WebRoot*".

5. Double click "*index.jsp*". This will open the file in Design mode. You will see two windows showing the content of the file: a visual designer on the top, and a source window at the bottom.

6. In the JSP file body (in the source window), after "This is my JSP page.<br>", add:
   ```
   <APPLET codebase="." CODE="jwg.ch2.chatGUI.ClientChatGUI.class"
   WIDTH=785 HEIGHT=527/>
   ```

7. Deploy the project using "Exploded Archive (development mode)". This will create the following directory: "*apache-tomcat/webapps/ch3_nodb*". Note: if for some reason you want to delete the project from MyEclipse, you should first undeploy the project from the web server.

8. Copy the "*jwg*" directory (in the MyEclipse workspace) into the "*apache-tomcat/webapps/ch3_nodb*" directory. This will add the applet for GUI chat client (developed in Chapter 2) into the deployed web project.

9. Start the GUI chat server as done in Chapter 2.

10. In a web browser, visit the URL "*http://localhost/ch3_nodb*". You should be able to see the GUI chat client running inside the web browser.

2 A Simple JSF Site

Please go through the MyEclipse tutorial "Create a JSF Web Application" and "Develop using the Visual JSF Page Designer". Then follow the steps below to enrich your "ch3_nodb" web project with JSF components. Also, you are recommended to study the JSF tutorial slides from

http://www.coreservlets.com/JSF-Tutorial/

2.1 Add JSF Capabilities

1. In the package explorer of MyEclipse, right click the project "ch3_nodb", then select "MyEclipse > Add JSF Capabilities".

2. Study the "Add JSF Capabilities" dialog box for a minute. Note that the URL pattern is "**.faces*". This means if you create a JSF file named "*login.jsp*", to access the file the URL should say "*login.faces*".

3. In the dialog box, keep all the default choices, and click "Finish".

The directory structure for the project is:

- *ch3_nodb/src:* the page beans, Java source code, and resource bundle file should be put here.
- *ch3_nodb/WebRoot:* the JSF files and style sheet should be put here.
- *ch3_nodb/WebRoot/WEB_INF/faces-config.xml:* you modify this file to create a managed beans or to create navigation rules among different JSF pages. After the file is opened, you can change the display mode to one of "Flow", "Design", and "Source". You typically do not need to edit the file in "Source" mode.

2.2 Add a Backing Bean

1. Open *faces-config.xml* in Design mode.

2. Click "Managed Beans" in the left panel.

3. Click "Add" in the right panel.

4. In the "New Managed Bean" dialog box, change Scope to "session", input "UserBean" to both the "Class" field and the "Name" field, and click "Finish". Note that a backing bean file "*UserBean.java*" has been created for you in the "src" directory of the project. A managed bean in JSF can have one of four scopes:

- None: The bean is not stored.
- Request: The bean new bean will be created upon every HTTP request.
- Session: The bean is durable across multiple requests but will expire when the session times out.
- Application: The bean is durable during the entire lifetime of the container.

5. Still in *faces-config.xml*, click "Add" to add a new property into the backing bean.

6. In the "Add Property" dialog box, input "*username*" to the "Property-Name" field, input "*java.lang.String*" to the "Property-Class" field, and click "Finish". You may also provide a default value for this property, in the "Value" field.

7. To add other properties, click "UserBean" in the "faces-config" panel, and repeat the previous two steps. Add another three fields:

   - *java.lang.String password*;
   - *boolean loggedIn* with default value = *false*;
   - *boolean notLoggedIn* with default value = *true*.

8. Open "UserBean.java", and you will see the four fields plus their getters and setters. You do not see the default values of the two boolean variables, but don't worry. Warning: to add new attributes in the backing bean file, you should always do so through *faces-config.xml* (design mode). Attributes added this way can be seen in both *faces-config.xml* and the Java bean file; but attributes added directly into the bean file is not visible in *faces-config.xml*.

9. Add two member functions to "UserBean.java":

```
public String login() {
   loggedIn = true;
   notLoggedIn = false;
   return "toHome";
}

public String logout() {
   loggedIn = false;
   notLoggedIn = true;
   return "toHome";
}
```

As a precursor to the next few steps, we will create a JSF file called "home.jsp". The JSF file will contain text fields allowing the user to input a username and password. These fields will be linked with the corresponding properties in the backing bean. The JSF file will also contain a welcome message for users already logged in. Obviously, at any time, either the username/password or the welcome message is displayed, but not both. To control this, we will link the "rendered" property of the username/password group with the "notLoggedIn" property, and link the "rendered" property of the welcome message with the "loggedIn" property. The return values of these function will be used to navigate among multiple JSF files.

2.3 Add a JSF File and Test-Run the Project

1. Open *faces-config.xml* in Flow mode.
2. In the vertical palette, click the last icon whose tooltip text is “JSP - Add JSP page”.
3. Click anywhere in the gridded canvas.
4. In the “Create a new JSP page” dialog box, modify the File Name field from “MyJsp.jsp” to “*home.jsp*”, make sure the “Template to use” field is “Default JSF template” (note: not “Default JSP template”), and click “Finish”.
5. Note that a new file called “home.jsp” has been created in the “WebRoot” directory of the project. Open this file.
6. Replace the line

   <title>My JSP ‘home.jsp’ starting page</title>

   with

   <title>CSG130 Game Site</title>

7. Delete the line “This is my JSF JSP page. <br>”.
8. In the palette, expand the “JSF HTML” tab, click “Form”, then click the space in the JSF file window where the deleted line (refer to the previous step) used to be. This will add, between `<f:view>` and `</f:view>` in the source window, the following code:

   ```
   <h:form></h:form>
   ```

   In case you do not see a palette that includes a “JSF HTML” tab, it is hidden. Click the small triangle located at the upper-right corner of the visual designer window, and the palette should be displayed.

9. Add an “Output Text” element into the form.
10. In the “Properties” window, change the value of the “value” field from “outputText” to “CSG130 Game Site”. Note that you need to press the ENTER key in order for the change to take effect.
11. In the palette, click “Panel Group”, then click the space in the JSP file window after the output text that we just inserted.
12. In the “Properties” window, click “Attributes”, and then set the “rendered” property of the panel group to “*#{UserBean.notLoggedIn}*” (without the double quotation marks).
13. Add another “Panel Group”, after the first one, whose “rendered” property is “*#{UserBean.loggedIn}*”.
14. In the first “Panel Group”, add:
    - an “Output Text” (value=“*Username:* ”),
    - a “Text Input” (value=“*#{UserBean.username}*”),
    - another “Output Text” (value=“ *Password:* ”),

- a “Secret Input” (value=“*#{UserBean.password}*”), and
- a “Command Link” (value=“*Login*”, action=“*#{UserBean.login}*”).

By default a “Command Link” element contains an “Output Text” element. You should remove the contained “Output Text”. In case you try to set the “action” property of the command link but do not see such a field in the “Quick Edit” panel of the “Properties” window, you are probably setting the properties of the enclosed output text rather than the command link itself.
At run time, if the user clicks a command link, the corresponding function (*login* in this case) in the backing bean will be executed. Later we will add navigation rules to direct the user to a certain JSF page, depending on the return value of the function.

15. In the second “Panel Group”, add three “Output Text” elements (values being “*Welcome,* ”, “*#{UserBean.username}*”, and “*!*”, respectively), and a “Command Link” (value=“*Logout*”, action=“*#{UserBean.logout}*”).

Now let’s deploy the project, and view it using a web browser by visiting:

http://localhost/ch3_nodb/home.faces

Once again, the JSF file is *home.jsp*, but to access it the URL should use “faces” as suffix. Next section will illustrate many JSF techniques through enriching the JSF project, starting with improving the look-and-feel.

3 Enrich the JSF Project

3.1 Use CSS to Improve the Look-and-Feel

Each JSF component has a “styleClass” property. This section describes how to create a CSS file, link the CSS file in a JSF file, and set the “styleClass” property of a JSF component to equal to a class defined in the CSS file.
To more completely learn CSS, go through a tutorial, e.g.

http://www.w3schools.com/css/

To create a CSS file:

- Right-click the WebRoot directory of the project, and select “New > other”.
- In the “New” dialog box, expand “MyEclipse > Web”, then select “CSS” and click “Next”.
- Provide “styles.css” as file name and then click “Finish”.
- Open the “styles.css” file and add the following definitions to it:

```
body {
   background-color: #bbccff }
.title {
   font-size: x-large;
   color: blue }
.input {
   font-size: x-small;
   color: black }
.normal {
   font-size: x-small;
   color: #111111 }
.name {
   font-size: small;
   color: #111111 }
.command {
   font-size: x-small;
   color: #997700;
   margin-left: 0.8em;
   margin-right: 0.4em }
.floatright {
   position: absolute;
   right: 1em;
   text-align: right }
.error {
   font-size: x-small;
   color: red }
.selectedcommand{
   font-size: small;
   color: blue;
   margin-left: 1em;
   margin-right: 1em;
   font-style: italic }
```

The style file sets the background color of a web page that refers to it, and defines a set of style classes.

To refer to the CSS file in a JSF file:

- Uncomment the following line in "home.jsp":

  <link rel="stylesheet" type="text/css" href="styles.css">

To use the style classes defined in the CSS file:

- In "home.jsp", click the "Output Text" with value = "CSG130 Game Site".
- In the "Properties" window, click "Attributes", then enter "*title*" to the "styleClass" field.
- Similarly,
  - set styleClass="*input*" for the two input fields;
  - set styleClass="*name*" for the "Output Text" which appears between "Welcome" and "!";
  - set styleClass="*command*" for the "Command Links";

  - set styleClass="*floatright*" for the two "Panel Groups"; and
  - set styleClass="*normal*" for the remaining "Output Text" elements.

Now re-deploy and test-run the project. You will see the look-and-feel has been significantly improved.

3.2 Use Message Bundle

A message bundle is a file storing pairs of (name, value). The JSF pages can refer to a *name* to actually display the associated *value*. This allows easy modification of common string values that appear in one or multiple JSF pages, and easy internationalization.

- Right-click on the project's "src" folder and select "*New > File*".
- Create a file named "*MessageBundle.properties*".
- Add an entry with name="*title*" and value="*CSG130 Game Site*".
- In the file "home.jsp", after "<f:view>", load the message bundle by adding:

  ```
  <f:loadBundle basename="MessageBundle" var="bundle"/>
  ```

- Replace the hard-coded title

  ```
  <h:outputText value="CSG130 Game Site" styleClass="title"></h:outputText>
  ```

  with

  ```
  <h:outputText value="#{bundle.title}" styleClass="title"></h:outputText>
  ```

The project still hard-encode many strings like "Username". To internationalize the project, you may choose to put those strings (or their corresponding strings in another language) in the message bundle has well.

3.3 Form Validation

There are multiple ways of validating a form in JSF. This section describes server-side validation using backing beans.

- Give the form in *home.jsp* an ID="*loginForm*".
- Add a "Messages" component immediately before

  ```
  </h:form>
  ```

  i.e. the end of the form. Let the styleClass of the new component be "*error*".

- Add the following code to the beginning of the *login()* function in the backing bean:

```
username = username.trim();
if (username.length()==0) {
   FacesContext facesContext = FacesContext.getCurrentInstance();
   FacesMessage facesMessage = new FacesMessage(
      "You have entered an invalid user name and/or password");
   facesContext.addMessage("loginForm", facesMessage);
   return "toHome";
}
```

Import the necessary packages to get rid of compilation errors. The addon code trims the leading and ending spaces of *username* and checks whether the username becomes empty. If it does, an error message will be added to the form with ID=*loginForm*. The message will be displayed at the place where the "Messages" component is located at.

Redeploy the project and try to login with an empty username, and you will see the error message.

3.4 Use JSP Forwarding

The default file, in a directory on a web server, that is accessed if the URL does not provide a file name, is "*index.jsp*". To forward an access of "*index.jsp*" to "*home.faces*", in *index.jsp*, just before

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

add:

<jsp:forward page="home.faces" />

And then replace the body of the file with

forwarding...

Now to test-run the project you can provide the following shorter URL:

http://localhost/ch3_nodb

3.5 Add More JSF Files

First, modify "home.jsp" to include a bar of command links.

- In "home.jsp", right before the "Messages" component, add a "Panel Grid".
- Change the "columns" property of the panel grid to 6, and change the "style" property of the panel grid to "*width=100%*".
- Remove the four "Output Text" elements from the panel grid.
- Add six "Command Links" to the panel grid, all having styleClass="*command*":
  - value="Home", action="#{UserBean.home}";
  - value="Forum", action="#{UserBean.forum}";
  - value="Social", action="#{UserBean.social}";
  - value="Play", action="#{UserBean.play}";
  - value="My Account", action="#{UserBean.account}";
  - value="FAQ", action="#{UserBean.faq}";

Next, modify the backing bean to handle these new actions.

- In the design mode of "faces-config.xml", add a property in the managed UserBean: Property-Name="*which*"; Property-Class="*int*"; and Value="*0*". This variable stores which page should be displayed.
- Add the following functions to "UserBean.java":

```
public String redirect() {
   switch (which) {
      case 0:   return "toHome";
      case 1:   return "toForum";
      case 2:   return "toSocial";
      case 3:   return "toPlay";
      case 4:   return "toAccount";
      case 5:   return "toFAQ";
   }
   return "toHome";
}

public String home() {
   which = 0;
   return redirect();
}

public String forum() {
   which = 1;
   return redirect();
}

public String social() {
   which = 2;
   return redirect();
}

public String play() {
   which = 3;
   return redirect();
}

public String account() {
   which = 4;
   return redirect();
}

public String faq() {
   which = 5;
   return redirect();
}
```

- Replace the three return statements, which are located in the functions *login()* and *logout()*, with "*return redirect()*".

Third, create the other JSF files and navigation rules among the JSF files.

- Open "faces-config.xml" in Flow Mode and then add five more JSF files: "forum.jsp", "social.jsp", "play.jsp", "account.jsp", and "faq.jsp".
- Use the content of "home.jsp" to replace those of the five new JSF files.
- In each of the six JSF files, replace the styleClass of one particular "Command Link" from "command" to "*selectedcommand*". For instance, in "home.jsp", the "Command Link" to be replaced has value="Home".
- (Optional:) Add some descriptive text in each file.
- Open *faces-config.xml* in design mode.
- Click "Navigation Rules", then click "Add".
- In the "Add Rule" dialog box, leave both fields empty and click "Finish". You will see that a new navigation rule identified by "[any]" has been generated.
- Add six navigation cases under the "[any]" navigation rule:
  - From-Outcome="*toHome*", To-view-ID="*home.jsp*";
  - From-Outcome="*toForum*", To-view-ID="*forum.jsp*";
  - From-Outcome="*toSocial*", To-view-ID="*social.jsp*";
  - From-Outcome="*toPlay*", To-view-ID="*play.jsp*";
  - From-Outcome="*toAccount*", To-view-ID="*account.jsp*";
  - From-Outcome="*toFAQ*", To-view-ID="*faq.jsp*".

Redeploy and test-run the project. Notice that clicking the login/logout links do not change the currently displayed JSF file. Under the hood what happened when the login link is clicked are:

1. The *login()* function in the backing bean is called.
2. The function calls *redirect()*.
3. The function returns a string depending on the value of *which*.
4. The string is used to help choose a navigation case under the "[any]" navigation rule.
5. The corresponding JSF file is executed.

3.6 Prevent Direct Access to Individual JSF Files

If the user enters, in a new web browser, the URL *http://localhost/ch3_nodb/play.faces*, the play page will be displayed. In most applications, such behavior of directly accessing individual JSF files should be prevented. This section describes how.

Add a *PhaseListener* Java file which says a null session (resulted from accessing individual JSF files directly) should be handled by a navigation case with From-Outcome = "*sessionExpired*":

- Open *faces-config.xml* in design mode.
- Click "faces-config.xml" on the left side of the window.

- In the "Lifecycle" panel, click "Add".
- In the "Add Phase Listener" dialog box, fill "Phase-Listener" with "*MyPhaseListener*", and click "Finish".
- From the package explorer, right click "src" and add a Java file named *MyPhaseListener.java* whose content is as follows:

```
import javax.faces.context.FacesContext;
import javax.faces.event.*;
import javax.servlet.http.HttpSession;

@SuppressWarnings("serial")
public class MyPhaseListener implements PhaseListener {
   public PhaseId getPhaseId() {
      return PhaseId.ANY_PHASE;
   }

   public void beforePhase(PhaseEvent e) {
      if(e.getPhaseId()== PhaseId.RENDER_RESPONSE) {
         FacesContext facesContext = e.getFacesContext();
         HttpSession sessionx = (HttpSession)facesContext.getExternalContext().
            getSession(false);
         if (sessionx == null) {
            facesContext.getApplication().getNavigationHandler().handleNavigation(
                  facesContext, "", "sessionExpired" );
         }
      }
   }

   public void afterPhase(PhaseEvent e) {
   }
}
```

Add a navigation rule saying to redirect any "*sessionExpired*" access to "*home.jsp*":

- Open *faces-config.xml* in design mode.
- Under the "[any]" navigation rule, add a new navigation case with From-Outcome="*sessionExpired*" and To-view-ID="*home.jsp*".

Redeploy the project and check. Open a new web browser and try to access the URL *http://localhost/ch3_nodb/play.faces*. You should see the content of "home.jsp" displayed instead.

3.7 Integrate the GUI Chat Client with the JSF Web Site

Modify *play.jsp* to refer to the GUI client, with parameter:

- Add the following code to *play.jsp*, immediately before </f:view>. The code contains two panel groups, one showing if the user has logged in, and the other showing otherwise. The first panel group contains an applet – this is how to integrate the chat GUI client with the JSF site. The applet has a parameter "username" which is linked with the corresponding property in the backing back. Note that in the *param* tag, we should use "$" rather than "#" to refer to property in the backing bean. The second panel group simply shows an error message if the user has not logged in.

```
<h:panelGroup rendered="#{UserBean.loggedIn}">
   <center>
   <APPLET codebase="." WIDTH=785 HEIGHT=527
      CODE="jwg.ch3.chatGUIClient.ClientChatGUI.class">
      <param name="username" value="${UserBean.username}" />
      Somehow your browser does not support Java Applet. If you have not
      installed Java plugin, install from
      <a href="http://www.java.com">java.com</a>. If you have, enable
      Applet in your browser. You need to restart your web browser before
      coming back to the game.
   </APPLET>
   </center>
</h:panelGroup>

<h:panelGroup rendered="#{UserBean.notLoggedIn}">
   <center>
   <h:outputText value="Please log in before playing." styleClass="normal">
   </h:outputText>
    </center>
</h:panelGroup>
```

Create a slightly modified version of the GUI client:

- In the *jwg_code* project, create a new package called *jwg.ch3.chatGUIClient*.
- Copy the three client files (*ClientChatGUI.java*, *ClientSideListenerChatGUI.java*, and *ClientChat-GUI.form*) from *jwg.ch2.chatGUI* to the new package.
- Modify *ClientChatGUI.java* to read a username from applet parameter as

  ```
  private String username;
  username = getParameter("username");
  ```

  and write this username to the server, instead of reading a nickname from keyboard and write to server. Also, change the background color of the applet to be the same as the background of the JSF site as specified in the CSS file.
- Deploy the project.
- Copy the "*jwg*" directory (from the MyEclipse workspace) into the "*apache-tomcat/webapps/ch3_nodb*" directory.
- Start the GUI chat server as done in Chapter 2.
- In a web browser, visit the URL "*http://localhost/ch3_nodb*". You should be able to see the GUI chat client running together with our developed JSF part.