4th assignment user interface using java servlets

Davide Lissoni Mat.179878
3/12/2016

1.Introduction:

This assignment is about java servlets, JSP and Filters used in a java dynamic web project. Its goal was to create a web project (web site) that works as user interface used to access a DB through an Enterprise JavaBean developed for the fourth assignment.
The web site must contains the same functionalities used in the two clients developed for the 4th assignment.
Basically the requirements was to replace the desktop interface implemented with a web site using the technologies mentioned above.

The main difference between the old clients and the web site is that for the 4th assignment project, the client has been divided in two different programs, one used for manage user operation and the other used to administrator operation assuming that, the person who will use the administrator client actually is the administrator.
For the web site instead has been chosen, for obvious reasons, to merge the two clients in a single website, distinguishing normal users from administrator users using the log in. Since the database was not prepared to handle this, for simplicity the user-name of an administrator user must be “admin”.

2.Implementation:

For the implementations of this project has been used four different technologies:

- **Bean interfaces**: used to call the bean methods implemented in the server site;
- **Java Servlets**: used as core of the program;
- **JSP** : used to implement the dynamic web pages that will be displayed. Dynamic attributes and parameters are passed from the servlets to the jsp;
- **Filters**: used to prevent unauthorized access to specific jsps or servlets.

2.1 Servlets:

All the servlets of this project basically follow the same structure. For the explanation I decided to show as example the servlet RegisterOperation which is the one used to register a new user on the database. The flow of the servlets can be summarized with this steps:

1. Getting session attributes and parameters wether there are some (figure 1);

```
HttpSession session = request.getSession();
inputuser = request.getParameter("Inputname");
inputpswd = request.getParameter("InputPassword");
```

Figure 1

2. Looking up the ejb used in the servlet. This operation has been carrie out in the same way

and using the same settings of the fourth assignment (figure 2);

```
Properties jndiProperties = new Properties();
jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY, "org.jboss.naming.remote.client.InitialContextFactory");
jndiProperties.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
jndiProperties.put("jboss.naming.client.ejb.context", true);
final Context ctx = new InitialContext(jndiProperties);

AccountManagerRemote am = (AccountManagerRemote) ctx.lookup(appName + "/" + moduleName + "/" + accountName + "!" + accountView);
```

Figure 2

3. Call the bean methods (figure 3);

```
am.registerAccount(inputuser, inputpswd);
```

Figure 3

4. Set the changed session attribute;
5. Redirect to the jsp page (figure 4).

```
request.setAttribute("Registration", true);
javax.servlet.RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/Login.jsp");
 dispatcher.forward(request, response);
```

Figure 4

The only servlets that contains remarkable differences from the other servlets in this project its LoginOperation. The servlet, after has retrieved the account manager bean follow these steps:

1. Takes username and password passed as parameter from the Login.jsp ;
2. Set the session attribute “iduser” as the return state of AccountManager.login(username,password) which will be the user id if the method succeed, 0 otherwise (figur 5);

```
int iduser=am.login(inputuser, inputpswd);
session.setAttribute("iduser", iduser);
```

Figure 5

3. Check the session attribute just setted, if 0 return to the index page, otherwise check if in the current session has been already setted the attribute CartManager, and, if so, retrieve the current CartManager bean from the session, else initialize a new Cart (figure 6).

```
if (iduser!=0){
    String username= am.getAccountInfo(iduser).getUsername();
    session.setAttribute("username", username);

    if(session.getAttribute("CartManager")!=null){
     cm= (CartRemote)session.getAttribute("CartManager");
     cm.setId(iduser);
        }else{

            cm= (CartRemote) ctx.lookup(appName + "/" +moduleName + "/" + cartName + "!" + cartView);
            cm.initialize(iduser);
            session.setAttribute("CartManager", cm);

        }

    javax.servlet.RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/ListBook.jsp");
    dispatcher.forward(request, response);

}else{javax.servlet.RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/Login.jsp");
dispatcher.forward(request, response);}
```

Figure 6

Comments: With the initialization of a new cart, the servlets will save in a session attribute the CartManager bean, in order to use the same bean in all the servlets (this since jsp don't let to pass object as request parameters). This will allows the user to retrieve the cart when logs in.

2.2 JSP:

The dynamic web pages used as browser user interface has been implemented through jsp technology in order to follow as much as possible the architectural pattern MVC. For the same motivation in order to do some java operation without include java code (i.e if, cycle statement, substring function, set and get parameters and fields) has been used JSTL, an JSP extension, and, in particular the tag library core and fn (figure 7).

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
```

Figure 7

The web-site design is minimal and managed using html and css languages.

2.3 Filters:

The filter has been implemented in order to filter request and response from particular servlets. For this project has been used two different filters:

FilterLogin: this filter verify if the user which send a request or get a response is logged in the system by checking the session attribute iduser. If the user is logged the filter do nothing otherwise it calls a redirect to the Login.jsp page (figure 8).

```
        if (session == null) {
            RequestDispatcher rd = request.getRequestDispatcher("/Login.jsp");
            rd.forward(request, response);
        } else {

            if ("0".equals(session.getAttribute("iduser")) || session.getAttribute("iduser") == null) {

                this.context.log("Unauthorized access request");
                RequestDispatcher rd = request.getRequestDispatcher("/Login.jsp");
                rd.forward(request, response);

            } else {

                chain.doFilter(request, response);
            }
        }
    }
```

Figure 8

The filter is used only in the servlets where the log in is required as showed in figure 9:

```
<filter-mapping>
     <filter-name>FilterLogin</filter-name>
     <servlet-name>BuyCartOperation</servlet-name>
     <servlet-name>AdminAddBook</servlet-name>
      <servlet-name>AdminViewOperations</servlet-name>
      <url-pattern>/AdminViewHistory.jsp</url-pattern>
</filter-mapping>
```

Figure 9

FilterAdmin: this filter is used to verify if the user is logged as administrator. The verification has been carried out in the same way of the FilterLogin.
The filter is used only in the servlets reachable only for the admin user (figure 10):

```
<filter-mapping>
     <filter-name>FilterAdmin</filter-name>
     <servlet-name>AdminAddBook</servlet-name>
     <servlet-name>AdminViewOperations</servlet-name>
      <url-pattern>/AdminViewHistory.jsp</url-pattern>
</filter-mapping>
```

Figure 10

3.Deployment:

In order to deploy the server site, the first step is to configure the JDBC Datasource in the Wildfly console. For this step follow the instructions explained in the 4th assignment report.

Then go in the Derby folder and run Derby using the command :

java -jar lib/derbyrun.jar server start &

The last operation in order to deploy the server is to deploy on Wildfly the Enterprise Application LibraryServerSession.
In order to do that, copy LibraryServerSession.ear in $JBOSS_HOME/standalone/deployments folder and then restart Wildfly. Your output should look like (figure 11):

```
17:07:55,021 INFO  [org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeplo
ymentUnitProcessor] (MSC service thread 1-7) JNDI bindings for session bean name
d BookManager in deployment unit subdeployment "LibraryServerSession-ejb.jar" of
 deployment "LibraryServerSession.ear" are as follows:

        java:global/LibraryServerSession/LibraryServerSession-ejb/BookManager!be
ans.BookManagerRemote
        java:app/LibraryServerSession-ejb/BookManager!beans.BookManagerRemote
        java:module/BookManager!beans.BookManagerRemote
        java:jboss/exported/LibraryServerSession/LibraryServerSession-ejb/BookMa
nager!beans.BookManagerRemote
        java:global/LibraryServerSession/LibraryServerSession-ejb/BookManager
        java:app/LibraryServerSession-ejb/BookManager
        java:module/BookManager
```

Figure 11

In order to run the web site there is the need to deploy the ClientLibrary.war on a application server. For testing the application has been used apache tomcat 8.5 and I am going to explain how to deploy the application on it.

If you are deploying both client and server on the same computer make sure that Wildfly and Apache will use different http port number.
Then copy the war file in $CATALINA_HOME/webapps and start apache typing in the command line:

$CATALINA_HOME/bin/startup.sh

Then open a browser and go to the address:

localhost:<http-apache-port-number>/ClientLibrary/