CO3409 Distributed Enterprise Systems: Basics of Java Server Pages (JSPs)

JSPs allow you to embed **Java** (not JavaScript) code inside HTML documents. This code is executed on the **server** before the page is delivered to the client. Consequently, the client only sees the HTML that was originally in the document along with any HTML that may be generated by the Java code. Of course the document may also contain **JavaScript** code and embedded **Java Applets** that will be executed on the **client**.

JSP is the same type of technology as Microsoft's ASP (particularly the version of ASP that was used before ASP.Net) and PHP. In this exercise, you will develop a series of pages to demonstrate a variety of JSP features.

Purpose

On completion of this **you should be able to**:

1. Use NetBeans to create a basic JSP-based application
2. Use Java Scriptlets in JSP pages
3. Mix HTML and scriptlets
4. Use built-in variables, built-in tags and directives

Activities

1. Preparation

Create a new NetBeans Project from the Java Web category: a Web Application called jspbasic.

Note: If you are using a network drive, be aware that Glassfish doesn't like UNC paths. When you create a project on the n: drive, replace the UNC name by `n:\` (e.g. to give `n:\My Documents\NetBeans Projects\aproject`). If you forget, close the project and reopen it, replacing the UNC path by `n:\`.

Select an appropriate server (Tomcat, a basic Web Server and Java Servlet container appears to be faster, but we will use **Payara** or if not available, **Glassfish**, which provide more sophisticated facilities but have more demanding processor and memory requirements.). Note the context path (`/jspbasic`). click **Finish** – do not select a framework.

2. Java Expressions

Create a new JSP page (right-click on the Web Pages folder, **New**, **Other**, **Web** category, JSP file type) called `index.jsp`. Delete the default `index.html` from the same folder. Modify the body of generated `index.jsp` file:

```
<body>
<p>The time is <em><%= new java.util.Date() %></em></p>
</body>
```

Run the page by right-clicking on the project (jspbasic) in the Projects pane and choosing “Run”. It should automatically load the browser (you may have to wait). While waiting, you can entertain yourself by watching the Server Log – the Server Tab in the Output Pane. You don't have to read or understand it, just marvel at everything that is going on. This tab can be useful for debugging Server problems, but there is a lot of information to hunt through.

Note: A Possible problem: if the application does not run with an error message that `… context.xml" could not be found, try closing NetBeans and reopening it`.

Troubleshooting: don't forget to change the UNC name for the path to the project folder to N:\ because of problems with Glassfish. If you forget, close the project, and reopen it, replacing the UNC path by N:.

If the page doesn’t load and you are on the main University network, try modifying the Payara/Glassfish Proxy settings: **Tools**, **Options**, **General**. Then set the **Manual Proxy Settings HTTP Proxy** to `proxy.uclan.ac.uk` and the port to `3128`. Click on the **More** button and set No Proxy Hosts to `*.uclan.ac.uk, <local>`, `localhost, 127.0.0.1`, `GCXXXXXX`. GCXXXXXX should be the name of your computer, which you can find by **Start**, **Computer**. Look at the bottom left of the Windows Explorer window. Re-run the application.

After it has loaded, refresh the page by clicking refresh in the browser. Notice that each time you reload the page in the browser, the current time is displayed. The character sequences `<%=` and `%>` enclose Java expressions, which are evaluated at run time, when the page is delivered by the server. The server ignores anything not enclosed in `<%=` and `%>` brackets.

This is what makes it possible to use JSP to generate dynamic HTML pages that can vary depending on who is requesting the page or in response to user actions.

3. Java Scriptlets

JSP also allows you to execute blocks of Java code when the server generates the page. The Java code to be executed is put between `<%` and `%>` delimiters (just like expressions, but without the `=` sign at the start of the code.)

This block of code is known as a ***scriptlet***. An expression (enclosed by `<%=` and `%>`) is evaluated and the result is included in the generated HTML. A scriptlet (in `<%` and `%>`) is executed, ***but doesn’t produce a result***. However, it can generate information to be included in the HTML through explicit output statements.

Modify `index.jsp` to the following:

```
<body>
<p>The time is <em><%= new java.util.Date() %></em></p>
<%
    // This scriptlet declares and initialises a variable
    // used in later JSP code.
    System.out.println("Evaluating date now");
    java.util.Date date = new java.util.Date();
%>
<p>The time when the scriptlet ran was <%= date %></p>
</body>
```

Note: that Netbeans will reformat the source if you right-click and select ***Format***.

Run the example as before. Notice the output from the `System.out.println` on the `server log` (Server Tab) in the output page. You could use such output statements for debugging, but NetBeans's debugger works with JSP and allows you to single-step, include breakpoints and watch variable values, so is a more sophisticated way to work out what's going on.

A scriptlet can output HTML directly to the page rather than to the log by using the built-in "out" variable, which is of type `javax.servlet.jsp.JspWriter`:

```
<body>
<p>The time is <em><%= new java.util.Date() %></em></p>
<%
    // This scriptlet declares and initialises a variable
    // used in later JSP code.
    System.out.println("Evaluating date now");
    java.util.Date date = new java.util.Date();
%>
<p>The time when the scriptlet ran was <%= date %></p>
<p>The time output directly by a scriptlet is
<%
    out.println( String.valueOf( date ));
%></p>
</body>
```

Questions

1. What are the differences between JavaScript and Java Scriptets?
2. Why does the code put `java.util` before the `Date` type?
3. What is the difference between `out` and `System.out`?

Test the debugger on this by inserting a breakpoint (Click on the grey left margin – the breakpoint will be marked by a red bar and a coloured block in the margin.). Run the application by clicking on the **Debug Main Project** Button (or use the Run menu to select **Debug**, or right-click and select **Debug**.).

Note: in NetBeans, breakpoints won't work if you simply **Run** the application. This may be quite slow to start because of the different applications involved and the need to restart the server in debug mode. Use the Debug menu to work out how to step and run. Click on the date variable where it is declared. Use the Debug menu to **watch** it. It will be displayed in the **Variables** pane, at the bottom of NetBeans.

4. Reading the request parameters

When an HTML page is requested, parameters can be provided by the browser. These parameters include the name of the requested file, cookies and any information provided by the user, e.g. through a form. The built-in variable called **request** holds the HTTP parameters provided by the client. It is of type `javax.servlet.http.HttpServletRequest`.

Create a new page by right-clicking on the project in the **Projects** tab and selecting **New, Other**. The **JSP** pages are in the **Web** category. Name the file, requests.jsp (you don't need to provide the extension – you can see what the final filename and location will be on the form.), and make sure the option is **JSP file**.

Add the following to the body:

```
<%
    out.println( "<br>The client machine's address is " +
request.getRemoteHost());
%>
```

By using the NetBeans autocomplete facility, add some additional statements to display some other parameters available from the **request** object as well.

Run the file by right-clicking and selecting **Run File**.

5. The response Variable

The built-in `response` variable affects the response being sent back to the browser. For example, response.sendRedirect("www.uclan.ac.uk"); asks the browser to send a request back to the browser asking it to load the page "www.uclan.ac.uk". Test this.

Fix the problem (Look at the address in the browser address bar. Think relative addressing.)

6. Processing of JSP

Each JSP page is converted to a servlet. After you have run the page, right-click on it in the project pane and select "**View Servlet**". Read through the servlet and explain how it works. You won't understand all the details, but you should be able to explain how a JSP page is translated into a Servlet.

7. Mixing Scriplets and HTML

You can use the `out` variable to generate HTML output from within a scriptlet, but this gets messy for more complicated HTML. It loses some of the advantages of JSP programming making it more like using a Servlet. However, scriptlets can be inter-mixed with HTML.

Look at the following example that generates a table containing the squares of numbers from 1 to 4.

Create a page called `table.jsp` containing just the following JSP fragment:

```
<table border=2>
<%     for ( int i = 1; i <= 4; i++ ) {%>
         <tr>
         <td><%= i %></td>
         <td><%= i * i %></td>
         </tr>

<%     }%>
</table>
```

Notice how the HTML and JSP code are mixed. In particular, the body of the for-loop contains both code and HTML. The HTML is output automatically each time round the loop

Question

1. What would the output from the above code be?

Modify the above code to allow the client to supply a parameter to define the range of numbers to be squared.

```
int lim = Integer.parseInt(request.getParameter("limit"));
```

Run this page by right-clicking on it in NetBeans. Why is an error thrown? What could you do to improve the behaviour (**hint: exceptions**)? Access the page from a browser by entering the following address `http://localhost:8080/jspbasic/table.jsp?limit=2`

Note: The page should be automatically made available to the server when you save it. Without an IDE, you would have to upload changed pages to the server.

The important thing to notice about the code is how the `%>` and `<%` characters appear in the middle of the "for" loop, to switch back into HTML and then back to Java.

8. JSP Directives

JSP directives can be used to provide instructions to the run-time system. Add the following line to the top of `index.jsp` before `<html>` tag:

```
<%@ page import="java.util.*" %>
```

You do can now use class from the java.util package without specifying the full path. For example:

```
Date date = new Date();
```

To import more than one item, separate the package names by commas, e.g.

```
<%@ page import=" java.text.*,java.util.* " %>
```

9. JSP Tags

A JSP tag is like an XML tag, with a **start tag**, a element body and an **end tag**. The start and end tag have the element name, enclosed in `<` and `>` characters. The end tag name is preceded by a `/` character after the `<` character. The element names have an embedded colon character, `:`, in them, the part of the name before the colon describes the namespace of the tag. For instance:

```
<jsp:tag>
body
</jsp:tag>
```

Again since this is XML, if the element does not require a body, the start and end tags can be conveniently merged together, as `<jsp:tag/>`

Elements can have attributes, name/value pairs specified in the start tag. Some tags are built-in, others are loaded from an external tag library. Built-in or pre-defined tags start with `jsp:`. For instance, `jsp:include` is a predefined tag that is used to include another page, calling the included file as if it had been called by the browser.

Modify `index.jsp` to contain:

```
<jsp:include page="table.jsp"/>
```

Try it.

Question

1. Identify any attributes in the above element?
2. How is the behaviour different if you use `jsp:forward` rather than `jsp:include`?
3. What is a namespace?

10. JSP Sessions

On a typical web site, a visitor might visit several pages and it can be useful to associate some data with the visitor that will persist from one page to the next. Sessions permit this.

A session is an object associated with a visitor that can be used to hold data. If it is not used, it will be destroyed after a configuration-specified time.

What do the following pages do? (**Hint: start with `getit.htm`**)

getit.htm

```
<html>
<body>
<form method=post action="saveit.jsp">
Item to buy: <input type=text name=item size=30>
<p><input type=submit>
</form>
</body>
</html>
```

When the form is submitted, the request will be passed to `saveit.jsp`. The variable `session` is another built-in variable.

saveit.jsp

```
<%
   String name = request.getParameter( "item" );
   session.setAttribute( "theItem", name );
%>
<html>
<body>
<a href="final.jsp">continue</a>
</body>
</html>
```

final.jsp

```
<html>
<body>
The item is <%= session.getAttribute( "theItem" ) %>
</body>
</html>
```

The session is kept around until no requests are received before a timeout elapses. The server assumes the user has left the site, and deletes the session.

11. Answer the following questions

1. What character sequences is used to delimit the JSP scriptlet?
2. What character sequence introduces a JSP expression?
3. What is the difference between a JSP expression and a JSP scriptlet?
4. Is it possible to include comments in JSP Java?
5. What is the purpose of the out variable?
6. What is the purpose of the request variable?
7. What does request.getParameter(...) do?
8. What is meant by “intermixing HTML and scriptlets”?
9. What is a JSP tag?
10. How similar is the JSP tag syntax and XML?
11. Give an example of how you could use a session object in an e-commerce site?
12. What are the differences between a JSP page and a servlet?
13. When should you use a JSP page and when should you use a servlet? (See http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html)