|  |
| --- |
| **CE212 Lab 4: Java Server Pages (JSP)**  **Introduction**  The purpose of this lab is to gain some experience of using Java Server Pages (JSP) to generate dynamic web pages.  Before working through this lab you should have studied the JSP introductory lecture notes.  JSP: Embedding Java in HTML to produce dynamic server-side content.  We shall be not be running our programs in an IDE this week; to help understand the deployment of JSP applications we wish to see how the content is arranged on the server. You may still use your preferred IDE to compile the Java files. In lab 5 we will see how to build a JSP application using IntelliJ.  JSP is executed solely on the server.  Therefore, providing the dynamically produced content is reasonably standard, it should work in any web browser. (Compare this with some of the JavaScript examples we have studied, which would not have worked in Internet Explorer.)  **Tomcat**  For this course, we shall use Tomcat as our web application server.  When properly installed and configured, it should run smoothly.  Tomcat configuration varies between different versions of Tomcat.  **Tomcat on the lab machines (Horizon server)**  Ignore the instructions in this section if you are not using your M: drive on the Horizon server but read the lab sheet for general information about Tomcat.  The lab machines have Tomcat pre-installed in the **Program Files** folder of the **C:** drive. Since all of the application files have to be placed within the Tomcat folder you need to make a personal copy of this installation by copying the entire folder **C:Program Files\tomcat** to the root directory of your **M:** drive if you do not already have a **M:\tomcat** folder.  You should now have a folder tree like the following one:  Note the following important folders:   * **webapps**: Tomcat expects each of your web applications to have a directory within **webapps;**the existing directories within **webapps** provide some sample web applications * **bin**: this contains the startup.bat and shutdown.bat batch files for starting and stopping tomcat; you may want to put shortcuts to these on your desktop. * **common/lib**: place any third-party .jar files here.  For example, when we use applications that talk to relational databases, you'll put the database .jar file here. * **work:** this is a folder where Tomcat places the Java servlets that it generates from your JSP pages; it is interesting to observe the relationship between a JSP page and the servlet that gets generated from it.   Check the operation of the server.  Execute the startup.bat file. A new window will appear containing messages from Tomcat - you should see at the end of these a line of the form **INFO: Server startup in 6244 ms**  If you fail to get this message see the note on environment variables below the next screenshot.  Do not close the Tomcat window (but you can minimise it to an icon). [The window will close automatically when you shut down Tomcat using shutdown.bat.]  Click this link:   * [**http://localhost:8080**](http://localhost:8080/)   If all went well, you would see something like this:  If not, possible problems are to do with your environment variables.  Manually inserting the correct JDK path into the file tomcat\bin\setclasspath.bat (line 7) will probably fix it - you'll need to find the root directory of JDK and use this in a line of the form  set JAVA\_HOME=C:\Program File\java\jdk1.14.0\_1  **Tomcat on your PC or laptop**  If you are using your own machine and do not have Tomcat installed you should download version 9 from [**here**](https://tomcat.apache.org/download-90.cgi). Just download the zip file (found under Core) and unzip it to a location of your choice. You will have a Tomcat folder; no installation is necessary. Now follow the instructions at the end of the previous section (i.e. execute the startup.bat file and then click on the localhost link).  **Hello Web Application**  As a first web application, we'll use the Hello World example from the lecture notes. Note that many file and folder-names are case-significant so always type them exactly as they appear here.  Create a new folder called **hello** within the **webapps** folder in Tomcat, then within this new folder create a file called Hello.jsp.  Using a tool such as WordPad copy the following text into it:  <html>  <head> <title> Hello JSP </title> </head>  <body>  <p> Hello World: <%= new java.util.Date() %>  </p>  </body>  </html>  Now click on this URL: [**http://localhost:8080/hello/Hello.jsp**](http://localhost:8080/hello/Hello.jsp), and you should see a page like this one:  Hit refresh a few times and note how the date is updated.  That's because the code is re-executed on the server each time the page is requested.  **Helper Classes**  A helper class is a Java class used by a JSP page to assist in doing some task.  Recall from the lecture that it is generally bad style to include complex Java code within a JSP page.  It is usually much better to farm such tasks out to helper classes.  If these classes are specific to a particular application, then the class files should go in a folder called **WEB-INF\classes** within your web application (e.g. **tomcat\webapps\hello\WEB-INF\classes**).  The next example introduces a trivial helper class (**hello.Greeting**); the point of this is to check that we can call on user-defined classes when required.  The JSP file is shown below. Copy this text into a new file called Hello2.jsp in your **hello** folder.  <%@ page import="java.util.Date" %>  <%@ page import="helo.Greeting" %>  <html>  <head><title>Greeting using Helper Class</title></head>  <body>  <p>Date is <%= new Date() %> </p>  <h2>Greeting <%= Greeting.hello() %> </h2>  </body>  </html>  Note how external java classes are imported using page import.  Edit the URL in the browser window so that it tries to access Hello2.jsp instead of Hello.jsp. You should see an error report with HTTP status 500 since the class **hello.Greeting** has not yet been provided.  The Java source code for the missing class is provided below (and can also be found in the file [**Greeting.java**](https://moodle.essex.ac.uk/pluginfile.php/870050/mod_resource/content/4/lab4/Greeting.java)).  package helo;  import java.util.Date;  public class Greeting {  public static String hello() {  return "Hello at : " + new Date();  }  }  Copy this code into a file called Greeting.java in an appropriate package and compile it. Find the .class file.  To work correctly, it is important that Tomcat can find the **.class** files that you use from within your JSP code, so you must name folders precisely as described.  As mentioned earlier the root directory for these is **WEB-INF\classes** within your web application directory. Create a **WEB-INF** folder inside the **hello** folder and a **classes** folder inside this folder. Since our class is called **helo.Greeting** we need to place it within a folder called **helo** inside the **classes** folder, so create this folder and copy your Greeting.class file into it.  Refresh the browser window. You'll probably find that you still get the status 500 error report since we need to force Tomcat to recompile the servlet. Make a minor change to the JSP page (e.g. add a space) and save it. This will force the regeneration of the servlet.  By default, Tomcat will not reload your Java helper classes when you recompile them.  For development, it's more convenient (though slightly less efficient) to have them reloaded after each change.  To make this happen you need a file called context.xml in a new folder called **META-INF** within the **webapps\hello** folder with the following contents.  <?xml version="1.0" encoding="UTF-8"?>  <Context reloadable="true" path="/">  </Context>  You can obtain such a file by right-clicking on this link: [**context.xml**](https://moodle.essex.ac.uk/pluginfile.php/870050/mod_resource/content/4/lab4/context.xml). (If you follow the link do ***not*** attempt to copy the contents from the browser window - the displayed content of an XML file in a browser is not identical to the actual contents of the file itself; you can however simply create a file called context.xml and paste the text shown above into it.)  **More Examples**  At this point, we have several different file types to keep track of. The file [**webby zip**](https://moodle.essex.ac.uk/pluginfile.php/870050/mod_resource/content/4/lab4/webby.zip) provides a complete working sample containing the hello world example and the hotel booking example (beans version) from the lecture notes. Extract the contents of the above zip file to a folder called **webby** within **tomcat\webapps**.  Then you should be able to run the hotel beans booking example by clicking on [**http://localhost:8080/webby/HotelForm.jsp**](http://localhost:8080/webby/HotelForm.jsp).  You may find it interesting to look at the servlet files generated by Tomcat from the JSP pages. As mentioned earlier these are in the **work** folder - you will need to look in the folders **hello** and/or **webby** inside **M:\tomcat\work\Catalina** and navigate down through several folders until you finally reach the **jsp** folder containing the sources and compiled versions of the servlets. |