PA165 - Lab session - Web Presentation Layer

Author: Martin Kuba <makub@ics.muni.cz>

Goal

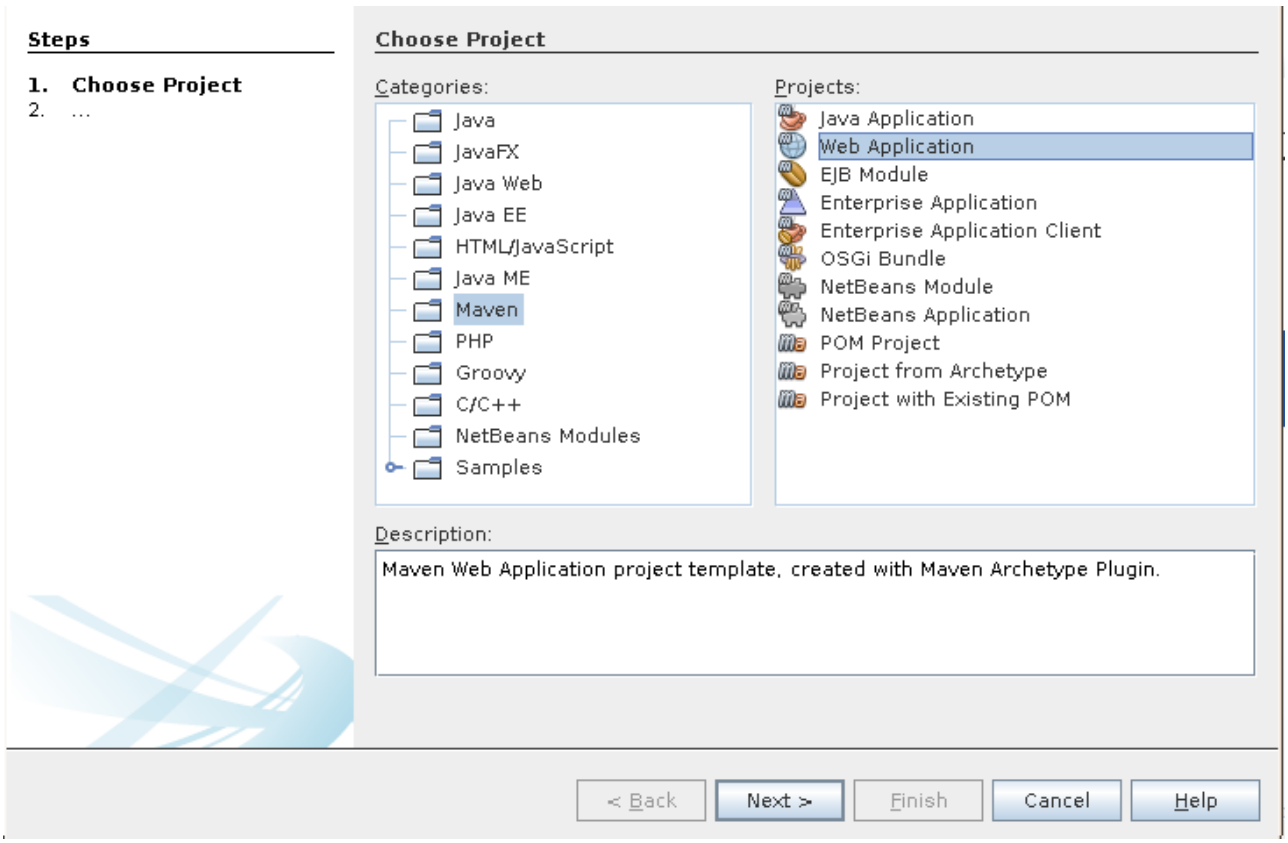
Experiment with basic building blocks of Java server-side web technology - servlets, filters, context listeners, Java Server Pages, JSP EL

Prerequisites

Netbeans 7.3.x, Tomcat 7, Java 7, Maven 3

Maven web application

Create a Maven web application. In NetBeans, choose **File - New Project**, a proct creation wizard open. In its first step, select Categories: **Maven**, Projects: **Web application** :



In its seconds step, type into the Package text field:: **cz.muni.fi.pa165.web1**
In its third step, select **Tomcat** and **JavaEE 6 web**. Click Finish. A new project is created.

In **Tools - Options - General - Web browser** choose you favorite browser.

Press **F6** or select **Run - Run project** in menu. The project should compile, build, start Tomcat, deploy and open browser showing a page with "Hello World !". Scream if something goes wrong.

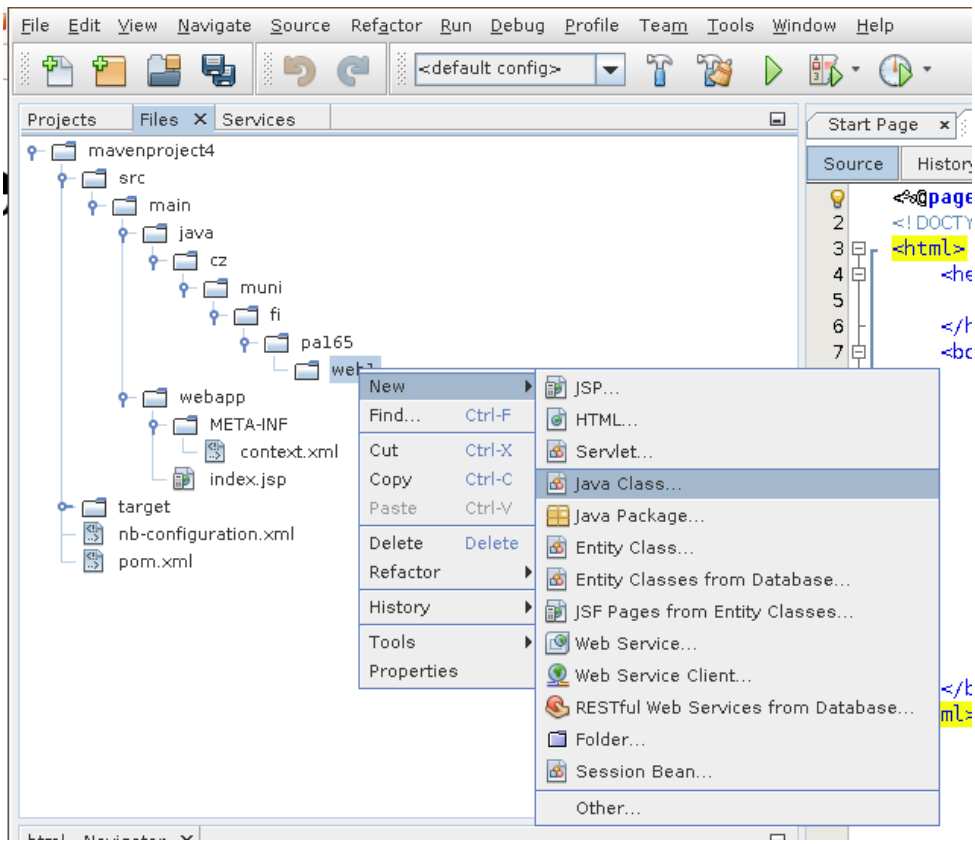
Servlet games

Find the source file **src/main/webapp/index.jsp**, delete its content and write the following:

```
<%@page contentType="text/html; charset=utf-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <title>Home</title>
  </head>
  <body>
    <h1>Hooray, it works !</h1>
    <ul>
      <li><a href="${pageContext.request.contextPath}/text/direct?a=1&b=ccc">text directly from servlet</a></li>
      <li><a href="${pageContext.request.contextPath}/file">file</a></li>
      <li><a href="${pageContext.request.contextPath}/redirect">redirect</a></li>
      <li><a href="${pageContext.request.contextPath}/text/bla">intentional error</a></li>
    </ul>
    <h2>will work later</h2>
    <ul>
      <li><a href="${pageContext.request.contextPath}/text/fromjsp">text from JSP</a></li>
      <li><a href="${pageContext.request.contextPath}/protected/protectedfile.txt">protected by password</a></li>
    </ul>
  </body>
</html>
```

The expression **${pageContext.request.contextPath}** includes URL prefix of the web application.

In the cz.muni.fi.pa165.web1 package in the src/main/java directory create a new Java class (do not select New - Servlet) named **MyDemoServlet.java** :



Put there the following code:

```
package cz.muni.fi.pa165.web1;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import javax.xml.bind.DatatypeConverter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URLEncoder;
import java.util.*;
import java.util.zip.*;

@WebServlet(urlPatterns = {"/text/*", "/file/*", "/redirect/*"})
public class MyDemoServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        if ("/text".equals(request.getServletPath()) && "/direct".equals(request.getPathInfo())) {
            response.setContentType("text/html;charset=utf-8");
            PrintWriter out = response.getWriter();
            out.println("<h1>Text</h1><pre>generated directly from servlet code");
            out.println("serverInfo=" + getServletContext().getServerInfo());
            out.println("parameters:<i>");
            for (String p : Collections.list(request.getParameterNames())) {
                out.println(p + "=" + request.getParameter(p));
            }
            return;
        } else if ("/text".equals(request.getServletPath()) && "/fromjsp".equals(request.getPathInfo())) {
            request.setAttribute("mylist", Arrays.asList("milk", "bread", "pizza"));
            request.getRequestDispatcher("/mypage.jsp").forward(request, response);
            return;
        } else if ("/file".equals(request.getServletPath())) {
            response.setContentType("application/zip");
            response.setHeader("Content-disposition", "attachment; filename=\"myfile.zip\"");
            ServletOutputStream sos = response.getOutputStream();
            ZipOutputStream zos = new ZipOutputStream(sos);
            zos.putNextEntry(new ZipEntry("something.txt"));
            zos.write("some text".getBytes());
            zos.putNextEntry(new ZipEntry("differentfile.txt"));
            zos.write("other text".getBytes());
            zos.close();
            return;
        } else if ("/redirect".equals(request.getServletPath())) {
            String url = request.getContextPath() + "/text/direct?z=" + URLEncoder.encode("?/ =", "utf-8");
            response.sendRedirect(response.encodeRedirectURL(url));
        } else {
            response.sendError(HttpServletResponse.SC_NOT_FOUND, "Nothing here, you are lost");
            return;
        }
    }
}
```

Run the project again (**F6** or **Run - Run project**)

Try to click the links on the displayed page, and note the following:

- the servlet class is mapped to URLs using the **@WebServlet** annotation
- the mapped URL contains 3 parts - **contextPath**, **servletPath** and **pathInfo**
- the method **request.getParameterNames()** returns **java.util.Enumeration** for historical reasons, it can be converted to **Iterable** using **Collections.list()**

- all request parameters are of type **String**
- when forwarding to another servlet or JSP, data are passed using the **setAttribute()** method
- when sending binary content, like the ZIP file, the header **Content-disposition** selects what the browser should do with the file- **inline** means to display, **attachment** means to save to a file
- when redirecting, the application-relative URL must be prepared:
    - prepended with contextPath to make it absolute
    - encoded using encodeRedirectURL() to keep session
    - parameter values must be URL-encoded
- instead of content or a redirect, we can send an error message using **sendError()**

JSP - Java Server Page

In the previous example, the link **/text/fromjsp** did not work, because the servlet forwarded to a non-existing JSP page. Now is the time to create it. Create the file **src/main/webapp/mypage.jsp.** (Do not write the .jsp extension in the file name, NetBeans adds it even when it is already present, naming the file mypage.jsp.jsp !) Put there the following code, which is a mix of 4 languages: HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets), JavaScript and Java::

```
<%@page import="java.util.List"%>
<%@page contentType="text/html; charset=utf-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
 <title>My JSP page</title>
 <style>
  #myheading { color: red; }
  table.mytable { border-collapse: collapse; margin: 10px; }
  table.mytable th, table.mytable td {  border: solid 1px black; padding: 4px; }
 </style>
 <script>
    console.log("Hello JavaScript console !");
 </script>
</head>
<body>
 <h1 id="myheading">My JSP page</h1>
 <p>list</p>
 <table class="mytable">
    <% List<String> list = (List<String>) pageContext.findAttribute("mylist");
    for(int i=0,n=list.size();i<n;i++) { %>
    <tr>
       <td><%=i%></td>
       <td><%=list.get(i)%></td>
    </tr>
    <% } %>
 </table>
</body>
</html>
```

Run the project again (F6).

In Firefox press CTRL+Shift+K, in Chrome press F12, a javascript console will open showing the log message.

Please note the following:

- at the first lines of a JSP, the @page directive should define

    - the output encoding in contentType, otherwise the default iso-8859-1 will be used
    - the source file encoding in the pageEncoding attribute, otherwise OS default will be used
- the CSS selector **#myheading** select the one element with attribute **id="myheading"**
- the CSS selector **.mytable** selects all elements with attribute **class="mytable"**
- in JavaScript, we can write tracing messages using **console.log()**
- Java code in JSP is written between <% %>
- there is always implicit variable **pageContext** in a JSP of type PageContext
- there is always implicit variable **out** in a JSP of type JspWriter
- instead of **<%out.print(i);%>** it is possible to write **<%=i%>**
- direct output of arbitrary texts is dangerous, as they may contain special HTML characters <>&, but there is no standard class for HTML-encoding using entities < > &

JSTL - JSP Standard Tag Library

Instead of Java code, we can use JSTL library, which, albeit being standard, is not a standard part of a web container. We have to add a dependency to the **pom.xml** Maven file:

```
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>javax.servlet.jsp.jstl</artifactId>
    <version>1.2.2</version>
  </dependency>
```

Edit the mypage.jsp and add the line
`<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`

and replace the loop with:

```
<c:forEach items="${mylist}" var="s" varStatus="i">
    <tr>
        <td>${i.count}</td>
        <td><c:out value="${s}"/></td>
    </tr>
    </c:forEach>
```

Filter

Create a new class in src/main/java/cz/muni/fi/pa165/web1 named **ProtectFilter.java**:

```
package cz.muni.fi.pa165.web1;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.bind.DatatypeConverter;
import javax.servlet.annotation.WebFilter;

@WebFilter("/protected/*")
public class ProtectFilter implements Filter {

    private String username = "pepa";
    private String password = "heslo";

    @Override
    public void init(FilterConfig fc) throws ServletException {
        String u = fc.getInitParameter("username");
        if(u!=null) username = u;
        String p = fc.getInitParameter("password");
        if(p!=null) password = p;
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) res;
        String auth = request.getHeader("Authorization");
            if (auth == null) {
                response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                response.setHeader("WWW-Authenticate", "Basic realm=\"type password\"");
                return;
            }
            String[] creds = new String(DatatypeConverter.parseBase64Binary(auth.split(" ")[1])).split(":", 2);
            if(!creds[0].equals(username)||!creds[1].equals(password)) {
                response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                response.setHeader("WWW-Authenticate", "Basic realm=\"type password\"");
                return;
            }
            chain.doFilter(req, res);
    }

    @Override
    public void destroy() {
    }
}
```

Create a folder src/main/webapp/protected and inside it create file protectedfile.txt with content “top secret”.

Run the project (F6) and click the protected link. Log in using the username and password.

Context listener

Create a new class in src/main/java/cz/muni/fi/pa165/web1 named **StartListener.java**:

```
package cz.muni.fi.pa165.web1;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class StartListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent ev) {
        ServletContext servletContext = ev.getServletContext();
        servletContext.log("starting application ");
        //initialize your app here ...
        servletContext.setAttribute("mydb", "value");
    }

    @Override
    public void contextDestroyed(ServletContextEvent ev) {
    }
}
```

Run the project again (F6), you will in the Apache log the log message.

Context listener is the right place for one-time initialization at application start.

That’s all folks ...