Proiectare Web – Lucrarea de laborator Nr. 5

PROGRAMARE LA SERVER WEB FOLOSIND SERVLETI SI PAGINI JSP

Un document HTML continut într-un fisier este o pagina Web statica, cu continut care nu se modifica decât daca se modifica fisierul însusi. Daca se doreste ca documentul de raspuns sa contina informatii care se modifica în timp sau în functie de datele trimise de client, este necesara generarea dinamica a paginilor Web, sub forma de documente HTML al carui continut se creeaza programatic. Pentru generarea paginilor HTML dinamice, serverul Web transmite cererea clientului unui program executabil sau script, care genereaza pagina de raspuns, iar serverul o trimite apoi clientului.
Generarea dinamica a paginilor HTML se realizeaza prin diferite metode de programare la server (server-side), dintre care cele mai cunoscute sunt: interfata de poarta comuna (CGI - Common Gateway Interface), scripturi PHP (Personal Hypertext Preprocessor), componente servlet, scripturi JSP (JavaServer Pages).

Un servlet este o clasa Java care prelucreaza cererile clientilor si construieste dinamic pagina HTML de raspuns. O pagina JSP este un document text care contine cod Java; codul Java se converteste într-un servlet care construieste partea dinamica de raspuns si aceasta parte se combina cu partea statica din pagina JSP pentru a forma pagina HTML de raspuns catre client.

Paginile JSP si componentele servlet sunt functional interschimbabile, dar unele aspecte de programare se rezolva mai simplu într-una sau alta din tehnologii. Daca cererea clientului necesita includerea unei mari parti de cod HTML în pagina de raspuns, atunci paginile JSP sunt mai simplu de folosit; daca cererea clientului necesita multiple operatii de prelucrare a datelor, atunci componentele servlet sunt mai simplu de folosit.

Clasele si interfetele necesare pentru dezvoltarea componentelor servlet se gasesc în pachetele javax.servlet si javax.servlet.http. Orice servlet trebuie sa implementeze interfata Servlet fie direct, fie prin extinderea clasei HttpServlet care implementeaza aceasta interfata, daca se prelucreaza numai cereri HTTP.

Un servlet poate fi instalat (*deployed*) în orice container de servleti, cum este serverul de aplicatii J2EE sau serverul de Web Tomcat. Pentru instalare sunt prevazute anumite conventii de stocare a claselor servlet-ilor si a descriptorilor XML, care sunt respectate de toate containerele Web, dar mai exista si unele aspecte particulare fiecarui container, care trebuie sa fie citite din documentatia respectiva. În continuarea lucrarii veti studia si veti experimenta modul de creare a servletilor si a paginilor JSP folosind serverul Tomcat. Pentru detalii de programare instalati manualul *Core-Servlets-and-JSP.pdf*, care se poate descarca de la adresa http://coreservlets.com.

5.1 Studiati modul de creare si instalare a servletilor în serverul Tomcat. Pentru aceasta lansati serverul (cu comanda startup, din subdirecurul bin al instalarii) si accesati adresa http://localhost:8080 dintr-un browser. Se obtine pagina de prezentare a documentatiei Apache-Tomcat 5.0.28, din care puteti accesa în primul rând link-ul Documentation.
Important este sa întelegeti modul de organizare a datelor (fisierele) componentelor în serverul Tomcat (capitolul Deployment). Pentru fiecare aplicatie (componenta) se defineste un director de baza (*document root*) care contine:

- Fisiere HTML, JSP, imagini
- /WEB-INF/web.xml: descriptorul de deployment al aplicatiei
- /WEB-INF/classes: clasele Java ale aplicatiei
- /WEB/INF/lib – bibliotecile jar folosite de aplicatie

Deployment-ul se poate face în mai multe moduri: folosind instrumente din distributia Tomcat (de exemplu, manager-ul serverului Tomcat), comenzi ant sau direct, creind directorul corespunzator unei aplicatii si restartând serverul Tomcat.

5.2 Studiati câteva exemple de servleti prezentate în documentatia Apache-Tomcat: HelloWorld-Example, RequestHeaderExample, etc, pentru care sunt prezentate si sursele programelor. Aceste exemple sunt grupate în directorul *servlets-examples*, iar în fisierul de descriere \servlets-examples\WEB-INF\web.xml, fiecare servlet este definit astfel:

```
    <servlet>
        <servlet-name>HelloWorldExample</servlet-name>
        <servlet-class>HelloWorldExample</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldExample</servlet-name>
        <url-pattern>/servlet/HelloWorldExample</url-pattern>
    </servlet-mapping>
```

Ca urmare, adresa URL cu care se poate accesa servletul respectiv este: http://localhost:8080/ servlets-examples/servlet/HelloWorldExample, folosind calea relativa /servlet/HelloWorldExample fata de directorul de baza (document root - care este servlets-examples), definita prin marcajul <url-pattern>. Observati ca aceasta este adresa cu care este accesat servletul HelloWorldExample.

Remarcati, de asemenea greselile (neconcordantele) de prezentare din documentatia Tomcat, care v-ar putea încurca în a întelege modul corect de functionare. Pentru servlet-ul HelloWorldExample (a carui clasa se gaseste în fisierul HelloWorldExample din directorul \WEB-INF\classes), listingul prezentat defineste clasa HelloWorld, nu clasa HelloWorldExample, cum rezulta din structura directorului WEB-INF\classes si a fisierului web.xml.

5.3 Creati un nou servlet, denumit HelloServlet, care sa afiseze ora exacta. Codul clasei servlet-ului este urmatorul:

```
/*
*     HelloServlet.java
*/
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                                  HttpServletResponse response)
              throws ServletException, java.io.IOException {
    java.util.Calendar calendar =
           new java.util.GregorianCalendar();
    int ora = calendar.get(java.util.Calendar.HOUR);
    int min = calendar.get(java.util.Calendar.MINUTE);
    java.io.PrintWriter out = response.getWriter();

      out.print("<HTML><BODY>");
      out.print("Hello Servlet! Este ora "
            + ora + ":" + min + ".");
      out.print("</HTML></BODY>");
  }
}
```

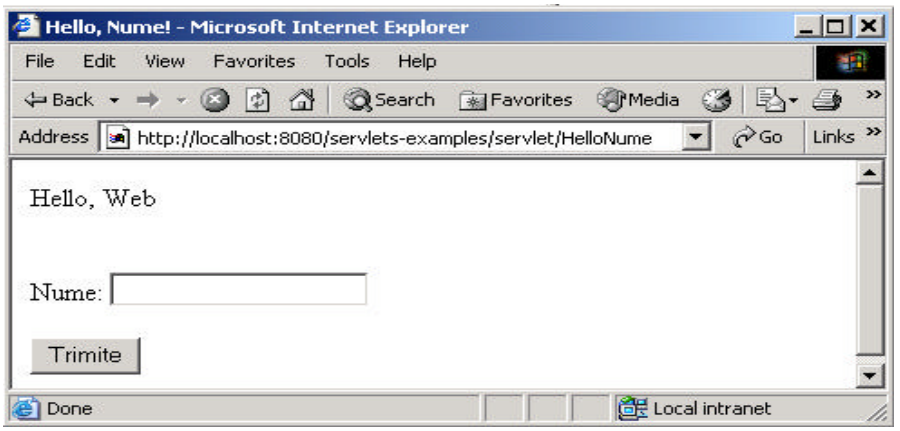
Pentru aceasta efectuati urmatoarele operatii:

1. Creati fisierul sursa cu numele HelloServlet.java, compilati-l cu compilatorul javac si copiati fisierul HelloServlet.class în directorul \webapps\servlets-examples\WEB-INF\classes.
2. Modificati fisierul web.xml din directorul \webapps\servlets-examples, adaugând liniile de script:

```
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/servlet/HelloServlet</url-pattern>
    </servlet-mapping>
```

3. Opriti si reporniti serverul Tomcat. La accesarea adresei http://localhost:8080/servlets-examples/servlet/HelloServlet dintr-un browser se va afisa pe ecran: Hello, Servlet! Este ora 4:31. (De fapt, ora exacta, din momentul executiei).

5.4 În mod asemanator, creati un servlet HelloNume, care sa genereze un document HTML care contine un formular cu o fereastra de text în care se poate introduce un nume si butonul SUBMIT, iar atributul ACTION are ca valoare adresa URL a servlet-ului HelloNume. La executia servlet-ului, acesta creeaza un document HTML, în care tipareste Hello urmat de parametrul primit, apoi genereaza formularul, asa cum se vede în figura de mai jos.



Testati diferentele de functionare între metodele GET si POST de trimitere a formularului. Detalii privind metodele claselor HTTPRequest si HTTPResponse se pot gasi în Tutorial Java 2 SDK (Standard Edition), capitolul Servlets.

5.5 Paginile JSP (*JavaServerPages*) prezinta avantaje fata de servleti deoarece permit separarea continutului static HTML al paginii de continutul generat dinamic, iar scrierea continutului HTML direct (nu prin intermediul operatiilor de print, ca în servlet) este mult mai simpla si mai intuitiva. Însa, trebuie sa se tina seama de faptul ca un document JSP este întotdeauna convertit într-un servlet, iar continutul HTML este printat în streamul de iesire al servlet-ului generat. Textul HTML (care se numeste *text template*) este trecut direct din documentul JSP în pagina returnata clientului, cu doua mici exceptii: pentru a avea secventa <% în documentul de iesire, trebuie ca textul template sa contina <\% si, daca se doreste ca un comentariu din pagina JSP sa nu apara în documentul de iesire, se foloseste marcajul: <%-- Comentariu JSP --%>. Comentariul de tip HTML <!—Comentariu HTML --> este trecut direct în documentul de iesire.

Fata de limbajul HTML normal, în paginile JSP se pot folosi trei tipuri de constructii: *scripturi* JSP (prin care se specifica parti de cod Java care vor deveni parte din servletul generat), *directive* JSP (prin care se controleaza structura de ansamblu a servlet-ului) si *actiuni* JSP (prin care se controleaza comportarea masinii JSP).

Elementele de scripting JSP sunt urmatoarele:

- Expresii: `<%= expression %>`. Expresia este evaluata si tiparita în documentul de iesire.
- Scriptleti: `<% code %>`. Codul este inserat în metoda *service* a servletului.
- Declaratii: `<%! code %>` Codul este inserat în clasa servletului, în afara oricarei metode.

Pentru elementele de scripting JSP este posibila si o alta forma sintactica, bazata pe marcaje XML:

- <jsp:expression> expresie Java </jsp:expression>
- <jsp:scriptlet> cod Java </jsp:scriptlet>
- <jsp:declaration> declaratie Java </jsp:declaration>

Expresiile si codul Java sunt, bineînteles, case-sensitive; de asemenea si marcajele XML sunt case-sensitive. Studiati detalii despre aceste elemente din Cap.10 *JSP Sripting Elements* din manualul Core-Servlets-And-JSP.

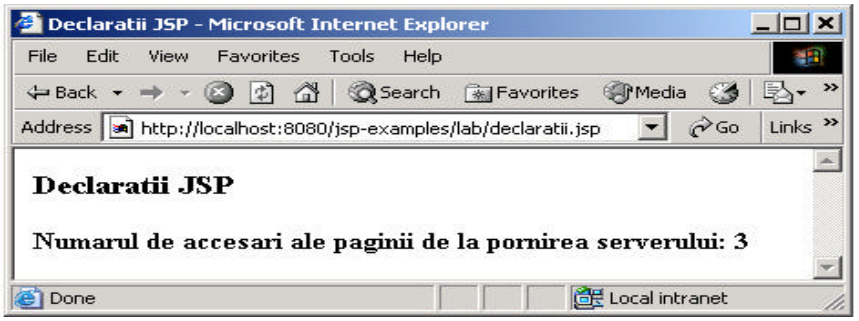
5.6 Experimentati expresiile JSP folosind urmatoarea pagina JSP (*expresii.jsp*):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Expresii JSP</TITLE>
```

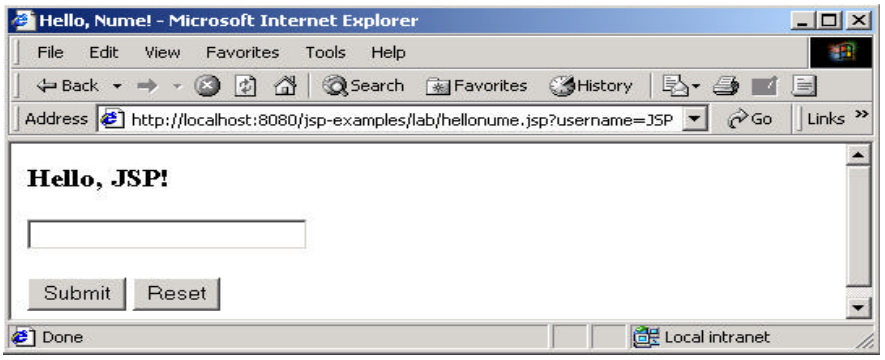
```
</HEAD>
<BODY>
<H2>Expresii JSP </H2>
<UL>
<LI>Timpul curent: <%= new java.util.Date() %>
<LI>Nume Host: <%= request.getRemoteHost() %>
<LI>ID sesiune: <%= session.getId() %>
</UL>
</BODY>
</HTML>
```

Pentru a obtine functionarea acestei pagini se memoreaza fisierul *expresii.jsp* într-un subdirector al directorului \webapps\ al instalarii serverului Tomcat (de exemplu, în directorul \webapps\jsp-examples\lab\) si se acceseaza din browser cu adresa http://localhost:8080/jsp-examples/lab/expresii.jsp. Creati un nou document JSP (cu numele *expresii_xml.jsp*) în care înlocuiti notatia traditionala JSP cu notatia XML a expresiilor si accesati noua pagina JSP.

5.7 Creati o pagina JSP care sa contorizeze numarul de operatii de acces la pagina respectiva folosind o variabila Java (`private int count=0;`) inserata în clasa servletului printr-o declaratie JSP si afisata ca expresie JSP. La accesarea paginii se va obtine o imagine ca cea de mai jos:



5.8 Reluati exemplul 5.4 de mai sus si realizati o pagina JSP (*hellonume.jsp*) care sa genereze formularul pentru introducerea textului (numele), ca în figura de mai jos.



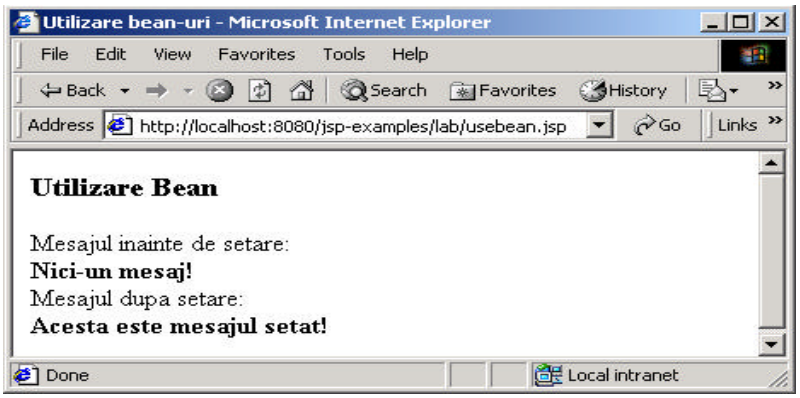
Adaugati scriptletul de mai jos care sa verifice lungimea numelui introdus si sa nu afiseze un mesaj de avertisment daca aceasta lungime este mai mica decât 2:

```
<% String nume = request.getParameter("username");
   if (nume.length() > 1) { %>
      <h3><font color="black"> Hello, ${param.username}!</font></h3>
<% } else { %>
      <h3><font color="black"> ${param.username} nu este un nume !</font></h3>
<% } %>
```

Remarcati posibilitatea utilizarii în scriptleti a unor variabile predefinite (obiecte implicite recunoscute de codul JSP): cum sunt *request* (obiect de clasa HttpServletRequest, care reprezinta cererea primita de la client), *response* (obiect de clasa *HttpServletResponse*, care reprezinta raspunsul care va fi trimis clientului) si altele. Modificati scriptletul de mai sus, astfel ca operatia de scriere a mesajului adecvat sa se efectueze în scriptlet (folosind metoda *out.println()*), nu prin intermediul codului HTML.

5.9 Studiati modul de utilizare a componentelor JavaBeans în pagini JSP din Cap. 13 *Using JavaBeans with JSP* din manualul *Core-Servlets-And-JSP*.

Creati si testati urmatorul exemplu de utilizare a bean-urilor (figura de mai jos).



În directorul \webapp\jsp-examples\WEB-INF\classes\lab definiti si compilati clasa bean-ului (StringBean.java):

```
// Fisier StringBean.java
package lab;
public class StringBean {
      private String message = "Nici-un mesaj! ";
      public String getMessage() {
            return(message);
      }
      public void setMessage(String message) {
            this.message = message;
      }
}
```

Fisierul JSP (*usebean.jsp*) se memoreaza în directorul \webapp\jsp-examples\lab:

```
<html>
<head><title>Utilizare bean-uri </title></head>
<h3> Utilizare Bean </h3>

<jsp:useBean id="stringBean" class="lab.StringBean" />

Mesajul inainte de setare:
<br><B> <jsp:getProperty name="stringBean"
     property="message" /> </br></B>
<jsp:setProperty name="stringBean"
      property="message"
      value="Acesta este mesajul setat!" />
Mesajul dupa setare:
<br><B> <%= stringBean.getMessage() %></B> </br>
</body>
</html>
```

La selectarea adresei http://localhost:8080/jsp-examples/lab/usebean.jsp din browser veti obtine fereastra afisata ca în figura de mai sus. Remarcati si studiati constructia <jsp:useBean …/> care este o *actiune JSP*, prin care se încarca clasa bean-ului si se instantiaza un obiect pentru a fi utilizat în pagina JSP.

5.10 Studiati exemplul *numguess.jsp* din distributia Tomcat. Fisierul *numguess.jsp* se afla în directorul \webapp\jsp-examples\num, iar bean-ul pe care-l foloseste (*NumberGuessBean*) se afla în directorul \webapp\jsp-examples\WEB-INF\classes\num.

Încercati sa modificati acest program. De exemplu, modificati valoarea maxima a numarului ales spre a fi ghicit, de la valoarea 100 la 10. Pentru aceasta modificati beanul NumberBuessBean, schimbând valoarea 100 în 10 în linia de program: `answer = Math.abs(new Random().nextInt() % 10) + 1;` si compilati clasa beanului. Apoi schimbati valoarea 100 cu 10 în doua locuri din fisierul numguess.jsp (liniile: *I'm thinking of a number between 1 and 100*). Veti constata ca, oricâte comenzi de *Refresh* veti da în browser si oicîte opriri si reporniri ale serverului Tomcat veti face, functionarea nu se schimba, adica tot limita 100 se foloseste si se afiseaza !

Aceasta situatie se datoreaza faptului ca în exemplul dat se foloseste clasa servletului în care a fost convertita pagina JSP, si acest lucru este prevazut în descriptorul *web.xml* din directorul root (jsp-examples). Stergeti sau comentati liniile urmatoare din fisierul web.xml:

```
<servlet>
     <servlet-name>org.apache.jsp.num.numguess_jsp</servlet-name>
     <servlet-class>org.apache.jsp.num.numguess_jsp</servlet-class>
 </servlet>
 <servlet-mapping>
     <servlet-name>org.apache.jsp.num.numguess_jsp</servlet-name>
     <url-pattern>/num/numguess.jsp</url-pattern>
 </servlet-mapping>
```

Dupa aceasta, veti constata ca modificarea efectuata în pagina JSP si în beanul folosit sunt functionale.

5.11 Completati exemplul *numguess.jsp* cu posibilitatea ca limita superioara a numarului care va fi ales sa fie setata de client printr-o caseta de text. Valoarea setata va fi folosita atât pentru initializarea beanului cât si pentru afisarea textului *I'm thinking of a number between 1 and …*