Aplicații Integrate pentru Întreprinderi

Laborator 9

17.12.2012 / 20.12.2012

Realizarea de aplicații web folosind Java Server Pages

Scopul laboratorului îl reprezintă folosirea mecanismelor oferite de tehnologia Java Server Pages pentru realizarea de aplicații (cu conținut dinamic) care să ofere utilizatorilor aceeaşi funcționalitate de care ar beneficia prin folosirea unor aplicații care să solicite resurse (programe instalate) pe maşina unde sunt rulate. Aceste cerințe se vor transfera maşinii pe care este găzduită aplicația, funcționalitatea fiind accesibilă printr-un client universal – browser-ul.

1. Ce este tehnologia Java Server Pages ?

JSP este o tehnologie pentru realizarea de pagini web generate dinamic şi bazate pe HTML, XML sau alte tipuri de documente. A fost lansată pe piață de compania Sun Microsystems în anul 1999 ca răspuns la tehnologiile PHP şi ASP, demonstrând faptul că Java este un limbaj de programare suficient de robust pentru a răspunde la provocările web-ului.

O pagină JSP este un document text care cuprinde două tipuri de date: statice care pot fi descrise în orice format (HTML, XML, SVG, WML) – denumite şi elemente de adnotare (*eng.* markup) şi dinamice, care pot fi directive JSP şi *scripleți*, adică blocuri de cod sursă Java folosite pentru implementarea unor funcționalități complexe, cum ar fi, de exemplu, comunicația cu o bază de date.

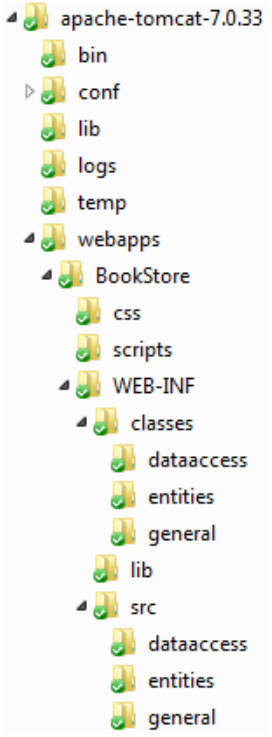
Avantajele utilizării Java Server Pages includ: separarea logicii aplicației de prezentare[1] (întreținerea este mai uşoară atât din partea programatorilor, cât şi a dezvoltatorilor web), portabilitate (asigurată de limbajul de programare Java – aplicația web poate fi utilizată pe mai multe platforme cu arhitectură diferită fără a fi necesară modificarea lor), conținutul dinamic care poate fi generat precum şi faptul că se bazează pe tehnologia Java Servlets[2], astfel încât preia şi avantajele pe care le oferă aceasta.

Deşi din punctul de vedere al programatorului un Java Servlet reprezintă o clasă Java (unde generarea elementelor de adnotare se face prin apelarea unor metode) iar o pagină JSP se aseamănă mai mult cu un document (unde conținutul static este separat de conținutul dinamic – generat de componente Java Beans sau etichete JSP), tehnologiile sunt echivalente. O pagină JSP este transformată într-un Java Servlet (care este compilat) responsabil de comunicația cu clientul şi de toate celelalte prelucrări. Modificările realizate asupra unei pagini JSP sunt detectate în mod automat, astfel încât atât procesul de transformare într-un Java Servlet cât şi procesul de compilare corespunzător sunt reluate atunci când se face o cerere pentru aceasta.

---

[1] Logica aplicației este folosită pentru a genera conținutul dinamic şi poate fi realizată prin componente Java Beans, în timp ce interfața cu utilizatorul este formată prin etichete JSP.
[2] Tehnologia Java Server Pages reprezintă o abstractizare de nivel înalt a Java Servlets. Totodată, în pagina JSP poate fi reutilizat codul care a fost dezvoltat într-un Java Servlet.

2. Integrarea Java Server Pages cu Apache Tomcat



Structura de directoare în serverul Apache Tomcat

În cazul unei aplicații web folosind tehnologia **JSP**, în rădăcina directorului unde se găseşte aplicația web trebuie să se adauge pagina denumită `index.jsp` care va fi afişată automat atunci când în browser este cerută adresa: `http://localhost:8080/<locatie>/`.

Tot aici vor fi incluse şi celelalte pagini JSP către care există legături. Nu mai este necesară specificarea fişierului `web.xml`, deoarece pentru fiecare pagină JSP va fi generat (atunci când pagina JSP va fi accesată[3]) un Java Servlet, proprietățile sale fiind generate în mod automat. Totodată, transferarea contextului (între paginile JSP) se face direct prin mecanismul de legături, fără a se mai face apel la funcționalitatea clasei `RequestDispatcher`.

De asemenea, în condițiile în care sunt folosite metode din clase Java (de regulă pentru logica aplicației – în special pentru conexiunea cu baza de date), acestea vor fi plasate într-un director `WEB-INF/classes`. Este recomandat ca acestea să fie modularizate pe pachete pentru ca referirea lor să se poată realiza cu uşurință. Dacă se folosesc biblioteci speciale[4] (în arhive `.jar`), trebuie incluse în `WEB-INF/lib`.

Încărcarea tuturor acestor clase se face atunci când este pornit serverul:

```
Dec 16, 2012 8:00:00 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory
ApacheTomcat\apache-tomcat-7.0.33\webapps\BookStore
```

Spre diferență de Java Servlets, nu mai este necesară realizarea operației de dezvoltare (*eng.* deploy) a aplicației web de fiecare dată când sunt realizate modificări asupra paginilor JSP, întrucât acestea sunt detectate automat, determinând generarea unor Java Servlets corespunzătoare. Repornirea serverului Apache Tomcat va fi necesară doar în cazul operării unor modificări pentru clasele Java ce sunt utilizate de aplicație, fiind necesară reîncărcarea lor (după ce compilarea lor este realizată de către programator) astfel încât să fie vizibile cele mai recente versiuni ale acestora.

Clasele Java Servlet generate pentru fiecare aplicație pot fi vizualizate în `%CATALINA_HOME%\work\Catalina\<adresa>\<locatie>\org\apache\jsp`. Corespondentul unei pagini `fisier.jsp` va fi un Java Servlet `fisier_jsp.java`. Tot aici vor fi plasate şi clasele obținute prin compilarea acestor Java Servlets. Atât operația de transformare din pagină JSP în Java Servlet cât şi compilarea claselor Java Servlet generate este realizată automat de serverul Apache Tomcat.

Datorită faptului că o cerere pentru o pagină JSP implică transformarea într-un Java Servlet precum şi compilarea acestuia, vizualizarea sa se va face după o perioadă mai mare decât în situația când aceste prelucrări nu mai trebuie realizate.

[3] Translatarea unei pagini JSP într-un Java Servlet se face atunci când, la accesarea din browser, clasa respectivă nu a fost generată niciodată (în directorul work al serverului Apache Tomcat) sau nu corespunde ultimei versiuni a paginii.
[4] Astfel de biblioteci speciale pot fi „driver”-ul de conectare la baza de date.

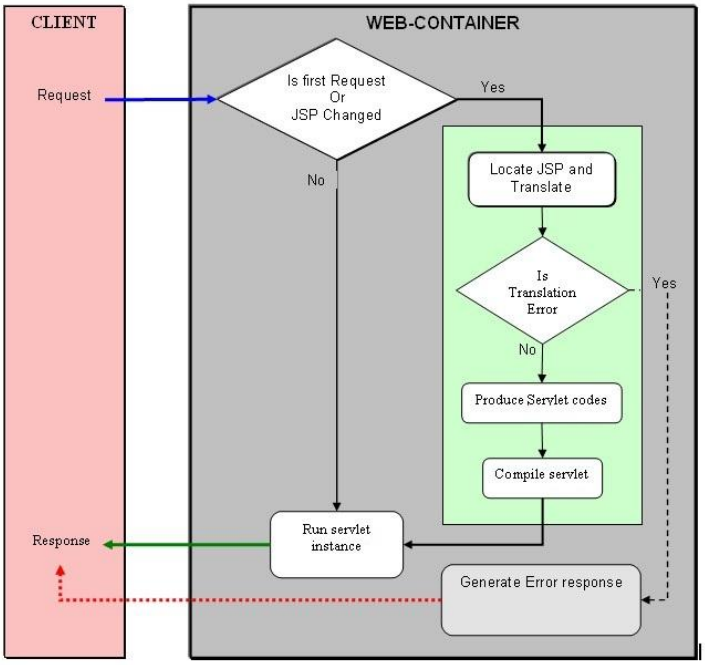
3. Ciclul de viață al unei pagini JSP

O pagină JSP tratează cererile asemenea unui servlet. Acesta este motivul pentru care ciclul de viață (ca de altfel şi multe alte capabilități) al paginilor JSP (în special aspectele dinamice) este determinat de tehnologia Java Servlet.

Atunci când o cerere este asociată unei pagini JSP, aceasta este tratată de un servlet special care verifică dacă servlet-ul asociat paginii JSP este mai vechi decât pagina în cauză, caz în care transformă pagina JSP într-un servlet pe care îl compilează[5].

Odată ce pagina JSP a fost transformată şi compilată, servlet-ul corespunzător va urma ciclul de viață corespunzător (încărcarea claselor, instanțierea lor, apelarea metodei `init()` pentru inițializarea unor resurse (partajate), comunicarea cu clientul şi procesări prin metoda `service()`, precum şi apelarea metodei `destroy()` dacă servlet-ul nu mai este necesar). Metodele (generate) se vor numi, în acest caz, `jspInit`, `_jspService`, `jspDestroy`[6].

```
public void _jspInit() {
    ...
}
public void _jspDestroy() {
    ...
}
public void _jspService
            (final javax.servlet.http.HttpServletRequest request,
             final javax.servlet.http.HttpServletResponse response)
            throws java.io.IOException, javax.servlet.ServletException
{
    ...
}
```



Ciclul de viață al unei pagini JSP

[5] Un avantaj pe care îl au paginile JSP față de Java Servlets este că procesul de compilare este realizat automat.

[6] Metodele `jspInit` şi `jspDestroy` (apelate o singură dată în ciclul de viață al unei pagini JSP) pot fi suprascrise ca declarații JSP (cuprinse între `<%!` şi `%>`).

Toate aceste metode fac parte din interfața `javax.servlet.HttpJspPage`. Metoda `_jspService()` nu trebuie suprascrisă niciodată pentru că este generată din conținutul paginii JSP.

Atât procesul de tranformare şi cât şi procesul de transformare pot genera erori care sunt scoase în evidență doar atunci când pagina este accesată pentru prima dată.

- dacă eroarea apare în timpul transformării, serverul va întoarce excepția `ParseException` iar clasa servlet va fi incompletă, astfel că ultima linie incompletă va fi un indicator către elementul JSP incorect;
- dacă eroarea se produce atunci când pagina JSP este compilată (dacă avem o eroare de sintaxă în scriptlet), serverul va întoarce excepția `JasperException` precum şi un mesaj care include numele servlet-ului asociat paginii JSP şi linia la care s-a constatat eroarea.

Atunci când se execută o pagină JSP pot apărea excepții, acestea trebuind tratate separat, în cadrul unei pagini dedicate unei astfel de acțiuni:

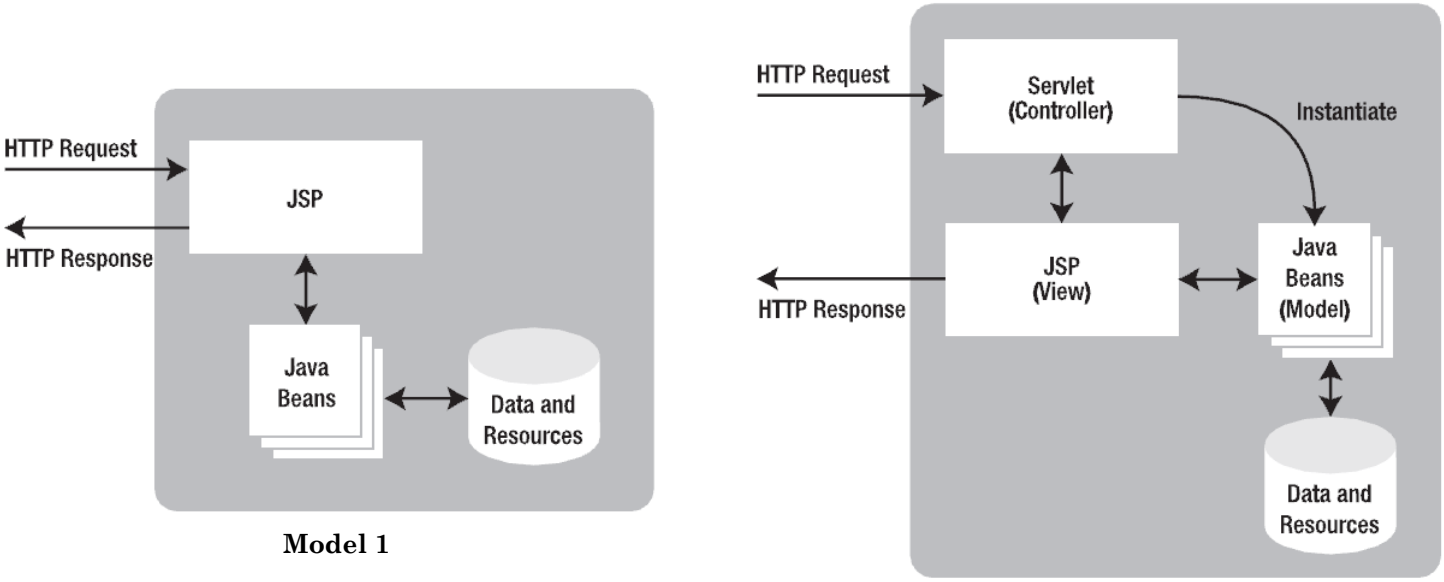
```
<@page errorPage="errorpage.jsp"%>
```

Pentru ca excepția (într-un obiect de tip `javax.servlet.jsp.JspException`) să fie disponibilă în cadrul paginii de eroare pentru a o interpreta, trebuie specificată, la începutul paginii de eroare, directiva:

```
<@page isErrorPage="true|false"%>
```

4. Structura unei pagini JSP

Combinarea de cod Java cu etichete HTML oferă posibilitatea realizării de pagini cu conținut dinamic, însă în cazul aplicațiilor cu funcționalitate complexă, procesul de întreținere poate fi dificil, problema provenind în special din faptul că partea de logică (care este dezvoltată de programatori – implicând algoritmi şi interacțiune cu baza de date) nu este separată de partea de prezentare (realizată de dezvoltatori web – concentrându-se mai ales pe elemente de grafică). Arhitectura aplicației JSP va reflecta raportul dintre aceste aspecte.



Model 1 **Model 2**

Arhitectura aplicațiilor JSP [1]

În modelul 1, logica aplicației este implementată în clase Java (componente Java Beans) şi poate fi apelată din partea de prezentare realizată prin pagini JSP. Această soluție este adecvată pentru cazul în care aplicația nu este foarte complexă, o problemă fiind constituită de faptul că toată comunicația cu clientul trebuie realizată din pagina JSP. O astfel de abordare se mai numeşte client-server, are avantajul simplității şi dezavantajul lipsei de scalabilitate.

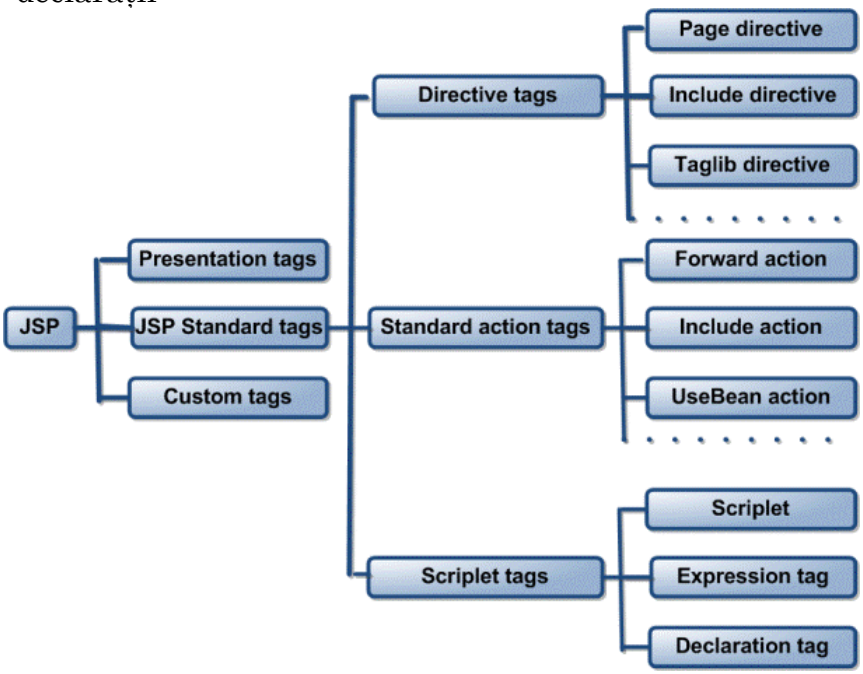
În modelul 2 (denumit şi abordare pe N-niveluri datorită separării arhitecturii serverului pe mai multe niveluri), se respectă şablonul de proiectare Model View Controller unde servlet-ul se ocupă de cererea de la client, implementează logica aplicației, realizând totodată şi instanțierea componentelor Java Beans. Pagina JSP obține date de la componentele Java Beans şi transmite un răspuns către client, prelucrările fiind realizate în mod transparent.

Aşadar, în cadrul unei pagini JSP se utilizează o combinație între etichete HTML (sau XML) şi blocuri de cod sursă Java pentru generarea de elemente dinamice. Fiecare pagină JSP este compilată într-un servlet, acesta primind cererea şi transmițând mai departe răspunsul.

O pagină JSP poate conține etichete de prezentare (în limbajul HTML), etichete JSP standard şi etichete definite de utilizator.

JSP foloseşte ca etichete standard:

- directive
- acțiuni
- scripleți
  - expresii
  - declarații



Structura unei pagini JSP

De asemenea, pot fi folosite *comentarii* care pot fi:

- comentarii JSP: acestea sunt ingorate de motorul JSP;
```
<%-- comentariu; --%>
```
- comentarii HTML: acestea sunt ignorate de browser;
```
<!-- comentariu -->
```

O *directivă*[7] este o instrucțiune care indică informații generale despre pagina respectivă. Sunt posibile următoarele directive: `page`, `include`, `taglib`.

```
<%directiva {atribut=”valoare”}%>
```

În cazul directivei `page`, este posibilă specificarea următoarelor atribute: `language`, `extends`, `import`, `session`, `buffer`, `autoFlush`, `isThreadSafe`, `info`, `isErrorPage`, `errorPage`, `contentType`, `pageEncoding`, `isELIgnored`.

```
<%@page import="java.sql.*, java.util.*, java.io.*,
dataaccess.*, entities.*, general.*"%>
```

Din exemplul de mai sus se observă faptul că pot fi incluse mai multe pachete în cadrul aceleiaşi directive, iar acestea pot fi clase Java standard sau clase definite de utilizator (şi plasate în directorul `classes`).

Directiva `include` este folosită pentru a include resursa specificată[8] în fişierul curent, în timp ce directiva `taglib` permite specificarea de etichete.

```
<%@include file="utilitare.jsp"%>
```

O *acțiune* este un marcaj care modifică modul în care se va comporta servlet-ul, în momentul compilării marcajele fiind înlocuite cu codul sursă Java corespunzător acțiunii.

Acțiune	Rezultat
`<jsp:useBean>`	instanțierea unei componente
`<jsp:setProperty>`	stabilirea valorii proprietății unei componente
`<jsp:getProperty>`	obținerea valorii proprietății unei componente
`<jsp:param>`	obținerea valorii unui parametru transmis prin obiectul implicit `request`
`<jsp:include>`	includerea resursei specificate
`<jsp:forward>`	cererea este transmisă către o altă pagină JSP
`<jsp:plugin>`	generarea de marcaje HTML care comandă încărcarea unor aplicații specifice

Spre exemplu, dacă se doreşte folosirea unor metode specifice din cadrul unei clase (incluse în directorul classes), se va folosi codul următor:

```
<jsp:useBean id="connection" scope="page"
             class="dataaccess.DataBaseConnection">
<jsp:setProperty name="connection" property="*"/>
</jsp:useBean>
```

utilizarea unei astfel de componente fiind ilustrată în continuare.

Prin specificarea atributului `property="*"`, se indică faptul că parametrii transmişi paginii JSP (ca cerere de la client) având acelaşi nume cu proprietățile clasei pentru care este creat Java Bean-ul respectiv vor fi instanțiați automat atunci când se face încărcarea documentului în browser.

Alternativ, anumite atribute ale clasei pot fi specificate (prin intermediul atributului `property`), putându-i-se asocia anumiți parametrii din cadrul cererii având denumiri diferite (prin intermediul atributului `param`), respectiv valori (prin intermediul atributului `value`, acesta putând fi specificat ca şir de caractere sau ca expresie JSP).

[7] Mai multe informații cu privire la utilizarea directivelor pot fi obținute de la adresa: http://www.jsptube.com/jsp-tutorials/jsp-page-directive.html.
[8] În general, această metodă trebuie evitată preferându-se în schimb definirea de etichete.

Acțiunea `jsp:forward` indică pagina JSP către care se transferă contextul:

```
<jsp:forward page="administrator.jsp"/>
<jsp:forward page="client.jsp"/>
```

Un *scriplet* este un cod sursă Java care va fi plasat în metoda de tip `service()` care se va genera în servlet-ul paginii JSP respective, accesând orice variabilă sau metodă declarată în contextul paginii respective.

O *expresie* este un scriplet care întoarce un rezultat[9], fiind evaluat când este executat servlet-ul, rezultatul fiind convertit la tipul `String` şi afişat.

```
<%
    ArrayList<Record> genericRecords = new ArrayList<Record>();
    Enumeration parameters = request.getParameterNames();
    int operation = Constants.OPERATION_NONE;
    String primaryKeyValue = "";
    while(parameters.hasMoreElements()) {
        String parameter = (String)parameters.nextElement();
        if (parameter.equals("adaugare")) {
            operation = Constants.OPERATION_INSERT;
        }
        else {
            genericRecords.add(
            new Record(parameter,request.getParameter(parameter)));
        }
        if (parameter.equals("selectedTable")) {
            selectedTable = request.getParameter(parameter);
            session.setAttribute(Constants.SELECTED_TABLE,selectedTable);
        }
    }
    switch (operation) {
        case Constants.OPERATION_INSERT:
            for (Record record:genericRecords) {
                String attribute = record.getAttribute();
                String value     = record.getValue();
                if (attribute.endsWith("_adaugare")) {
                    insertRecords.add(
                        new Record(
                            attribute.substring(0,attribute.indexOf("_")),
                            value)
                     );
                }
    }
    if (operation != Constants.OPERATION_UPDATE_PHASE1)
        primaryKeyValue = null;
%>
```

O *declarație* permite precizarea de variabile sau de metode, similar cu modul în care s-ar face într-o clasă, domeniul de vizibilitate fiind pe tot parcursul paginii respective.

```
<%!
    public String getAttributes(ArrayList<Record> records) {
        String result = "";
        for (Record record: records) {
            result += record.getAttribute()+",";
        }
        int position = result.lastIndexOf(",");
        return result.substring(0,position!=-1?position:result.length());
    }
```

[9] Spre diferență de scriptlet care este cuprins între etichetele `<%` şi `%>`, o expresie este cuprinsă între etichetele `<%=` şi `%>`.

```
    public String getValues(String tableName, ArrayList<Record> records) {
        String result = "";
        DataBaseConnection connection = new DataBaseConnection();
        String attributes = connection.getTableStructure(tableName);
        StringTokenizer attributeName =
                        new StringTokenizer(attributes,",");
        while (attributeName.hasMoreTokens()) {
            String attribute = attributeName.nextToken();
            for (Record record: records) {
                if (attribute.equals(record.getAttribute())) {
                    String value = record.getValue();
                    if (value==null || value.equals("")) {
                         value="invalid";
                    }
                    result += value+",";
                }
            }
        }
        int position = result.lastIndexOf(",");
        return result.substring(0,position!=-1?position:result.length());
    }
%>
```

Obiectele implicite care pot fi referite în cadrul unei pagini JSP sunt:

Obiect Implicit	Tip	Descriere
request	subclasă a `javax.servlet.ServletRequest`	cererea care a invocat pagina JSP
response	subclasă a `javax.servlet.ServletResponse`	răspunsul
pageContext	`javax.servlet.jsp.PageContext`	contextul paginii în funcție de serverul web
session	`javax.servlet.http.HttpSession`	obiect sesiune
config	`javax.servlet.ServletConfig`	
application	`javax.servlet.ServletContext`	context pagină JSP
page		
out	`javax.servlet.jsp.JspWriter`	obiect care scrie în fluxul de ieşire

Astfel de obiecte vor fi disponibile doar în cadrul metodei `_jspService`, astfel încât referirea acestora în cadrul unei declarații JSP nu are sens.

Pe obiectele `request` şi `response` vor putea fi apelate aceleaşi metode ca pentru obiecte de tipul `HttpServletRequest`, respectiv `HttpServletResponse` (`getParameterNames`, `getParameter`, respectiv `setContentType`, `getWriter`).

Obiectul `session` este creat în mod implicit (nu mai este necesară crearea dintr-un obiect de tip `request`), putându-se opera cu acesta în mod similar ca şi cu un obiect `HttpSession` (`get/setAttribute`).

Obiectul `out` este echivalentul obiectului `PrintWriter` din Java Servlets, fiind utilizat la afişarea conținutului dinamic în cadrul paginii JSP, atunci când aceasta nu se poate face prin intermediul unor etichete HTML (eventual combinate cu expresii JSP).

Activitate de Laborator

[0p] 1. Să se instaleze baza de date prin rularea script-ului (specific) `Laborator09.sql`.

Conținutul scriptului `Laborator09.sql` este identic cu cel din `Laborator04|08.sql` astfel că această etapă nu mai este necesară în cazul în care aceasta a fost deja realizată.

[0p] 2. Să se specifice variabilele de mediu `JAVA_HOME` şi `JAVA_JRE` în script-urile `setClasspath[.bat|.sh]` (din directorul `bin` al serverului Apache Tomcat).

[0p] 3. Să se completeze în clasa `Constants` utilizatorul şi parola pentru accesarea bazei de date.

[0p] 4. Să se „publice" aplicația BookStore în contextul serverului Apache Tomcat

Detalii cu privire la „publicarea" unei aplicații în contextul serverului Apache Tomcat sunt redate în Laboratorul 8 (pentru mediile de dezvoltare NetBeans şi Eclipse EE, cât şi pentru cazul în care nu se foloseşte un mediu de dezvoltare);

[0p] 5. Să se testeze aplicația prin accesarea adresei http://localhost:8080/BookStore/

În script-ul `Laborator09.sql` există exemple de utilizatori care pot fi folosite pentru accesarea paginilor de administrator, respectiv de client.

[0p] 6. Să se acceseze pagina de administrator şi să se testeze funcționalitățile implementate în cadrul acesteia (adăugare, editare, ştergere).

[4p] 7. În pagina `client.jsp`, să se determine conținutul coşului de cumpărături, ținând cont de situația în care pentru un produs care există deja în coşul de cumpărături se poate actualiza cantitatea.

În situația în care solicitarea pentru un produs depăşeşte stocul existent, aceasta nu va fi luată în considerare.

[1p] 8. În pagina `client.jsp` să se afişeze conținutul coşului de cumpărături.

[1p] 9. În pagina `client.jsp`, să se adauge un buton prin care se poate transmite o comandă (o comandă conține produsele selectate în coşul de cumpărături).

[4p] 10. În pagina `client.jsp` să se implementeze operația pentru transmiterea unei comenzi, aceasta implicând următoarele etape:

- înregistrarea comenzii în baza de date;
- actualizarea stocurilor;
- golirea coşului de cumpărături.

[1p] 11. În paginile `administrator.jsp` şi `client.jsp` să se implementeze funcționalitatea prin care un utilizator se poate deconecta, fiind afişată pagina `index.jsp`.

Bibliografie

[1] Giulio Zambon with Michael Sekler – *Beginning JSP™, JSF™ and Tomcat Web Development: From Novice to Professional*, *Apress*, 2007

[2] Irina Athanasiu – *Java ca limbaj pentru programarea distribuită*, editura Matrix Rom, Bucureşti, 2002

[3] *JSP Tutorial* - http://www.tutorialspoint.com/jsp/index.htm

[4] *JSP Tutorial* - http://www.jsptutorial.net/

[5] *Servlet World* - http://www.servletworld.com/jsp-tutorials/index.html