

Aplicații Integrate pentru Întreprinderi

Laborator 10

06.12.2011

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 aceeași funcționalitate de care ar beneficia prin folosirea unor aplicații care să solicite resurse (programe instalate) pe mașina unde sunt rulate. Aceste cerințe se vor transfera mașinii pe care este găzduită aplicația, funcționalitatea fiind accesibilă printr-un client universal – browser-ul.

1. Ce este tehnologia Java Server Pages ?
2. Integrarea Java Server Pages cu Apache Tomcat
3. Care este ciclul de viață al unei pagini JSP ?
4. Care este structura unei pagini JSP ?

1. Ce este tehnologia Java Server Pages ?

JSP este o tehnologie pentru realizarea de pagini web generate dinamic și 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 și 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 și elemente de adnotare (*eng.* markup) și dinamice, care pot fi directive JSP și *scripteți*, adică blocuri de cod sursă Java folosite pentru implementarea unor funcționalități complexe, cum ar fi, de exemplu, comunicația cu o bază de date.

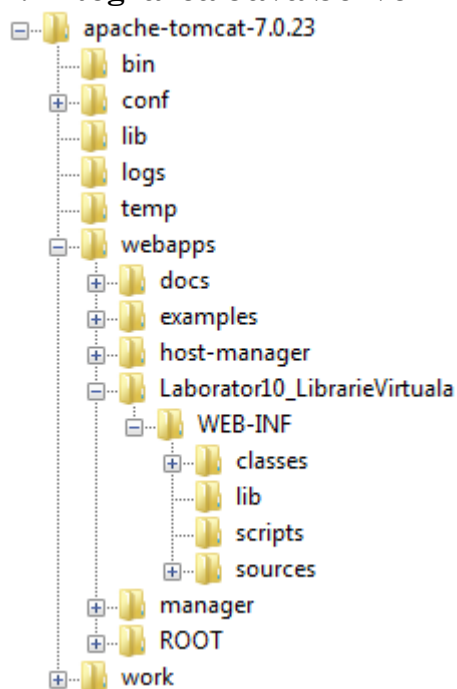
Avantajele utilizării Java Server Pages includ: separarea logicii aplicației de prezentare¹ (întreținerea este mai ușoară atât din partea programatorilor, cât și a dezvoltatorilor web), portabilitate (asigurată de limbajul de programare Java – aplicația web poate fi utilizată pe mai multe platforme cu arhitectură diferită fără a fi necesară modificarea lor), conținutul dinamic care poate oricând fi generat precum și faptul că se bazează pe tehnologia Java Servlets², astfel încât preia și avantajele pe care le oferă aceasta.

Deși din punctul de vedere al programatorului un Java Servlet reprezintă un program 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 și 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 și procesul de compilare corespunzător sunt reluate atunci când se face o cerere pentru aceasta.

¹ Logica aplicației este folosită pentru a genera conținutul dinamic și 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 **JSP**, în rădăcina directorului unde este aplicația web trebuie să se găsească pagina denumită `index.jsp` care va fi afișată automat atunci când în browser este cerută adresa:

<http://localhost:8080/<locatie>/>. Tot aici vor fi incluse și celelalte pagini JSP către care există legături. Nu mai este necesară specificarea fișierului `web.xml`, deoarece pentru fiecare pagină JSP este generat un Java Servlet, proprietățile sale fiind generate în mod automat.

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 ușurință. Dacă se folosesc biblioteci speciale³ (în arhive `.jar`), trebuie incluse în `WEB-INF/lib`.

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

```
Dec 04, 2011 8:00:00 AM org.apache.catalina.startup.HostConfig deployDirectory  
INFO: Deploying web application directory  
ApacheTomcat\apache-tomcat-7.0.23\webapps\Laborator10_LibrarieVirtuala
```

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 și clasele obținute prin compilarea acestor Java Servlets. Atât operația de transformare din pagină JSP în Java Servlet cât și 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 și compilarea acestuia (în cazul în care acest proces nu a fost realizat sau pagina JSP a fost modificată de la ultima accesare), vizualizarea sa în browser se va face după o perioadă mai mare decât în situația în care aceste prelucrări nu mai trebuie realizate (un Java Servlet corespunzător celei mai recente versiuni a paginii JSP a fost generat).

³ Astfel de biblioteci speciale pot fi „driver”-ul de conectare la baza de date.

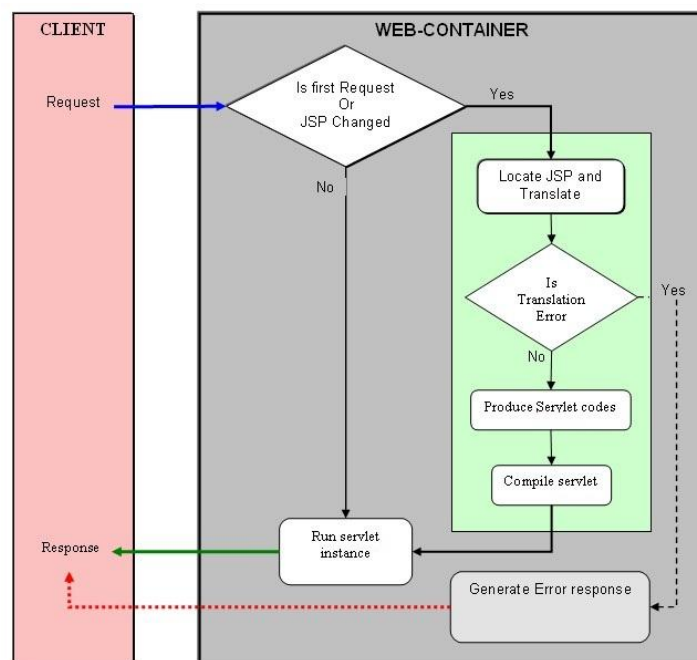
3. Care este ciclul de viață al unei pagini JSP ?

O pagină JSP tratează cererile asemenea unui servlet. Acesta este motivul pentru care ciclul de viață (ca de altfel și 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 tratează 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 compilează⁴.

Odată ce pagina JSP a fost transformată și 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 și procesări prin metoda `service()`, precum și apelarea metodei `destroy()` dacă servlet-ul nu mai este necesar). Metodele (generate) se vor numi, în acest caz, `jspInit`, `_jspService`, `jspDestroy`⁵.

```
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

⁴ Un avantaj pe care îl au paginile JSP față de Java Servlets este că procesul de compilare este realizat automat.

⁵ Metodele `jspInit` și `jspDestroy` (apelate o singură dată în ciclul de viață al unei pagini JSP) pot fi suprascrise ca declarații JSP (cuprinse între `<%! și %>`).

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 transformare și cât și 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 și un mesaj care include numele servlet-ului asociat paginii JSP și 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"%>
```

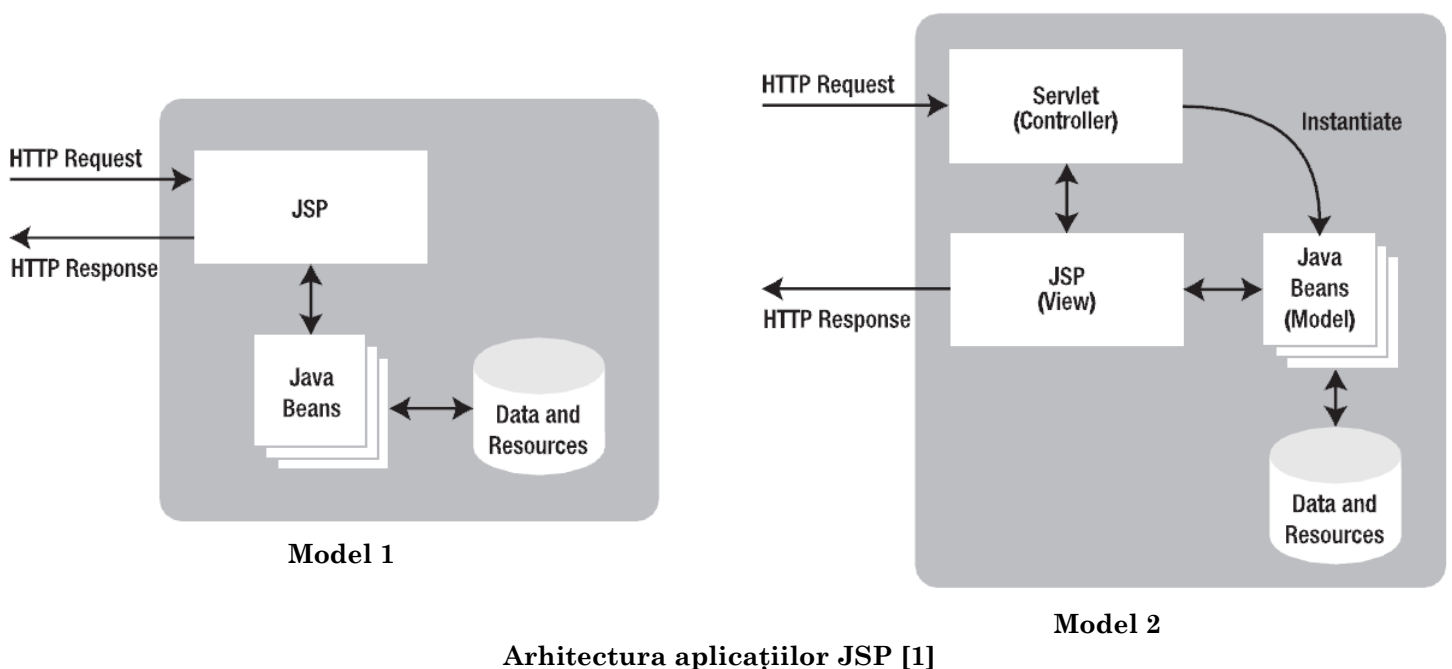
Pentru ca excepția (într-un obiect de tip `javax.servlet.jsp.JspException`) să fie disponibilă în cadrul paginii de eroare pentru a o interpreta, trebuie specificată, la începutul paginii de eroare, directiva:

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

4. Care este 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 și 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.



În modelul 1, logica aplicației este implementată în clase Java (componente Java Beans) și 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 numește client-server, are avantajul simplității și dezavantajul lipsei de scalabilitate.

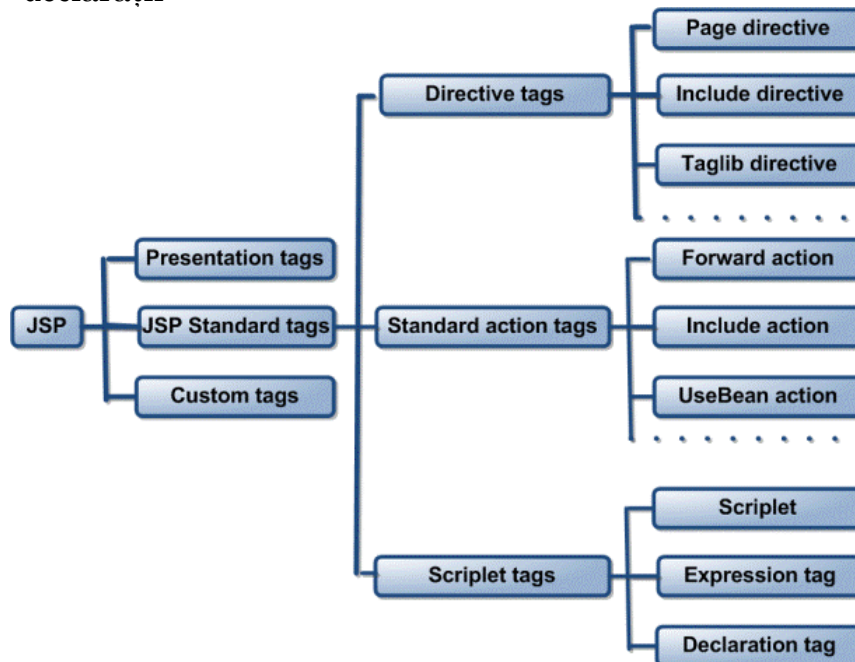
În modelul 2 (denumit și abordare pe N-niveluri datorită separării arhitecturii serverului pe mai multe niveluri), se respectă șablonul de proiectare Model View Controller unde servlet-ul se ocupă de cererea de la client, implementează logica aplicației, realizând totodată și instanțierea componentelor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

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

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

JSP folosește ca etichete standard:

- directive
- acțiuni
- scripțeti
 - expresii
 - declarații



Structura unei pagini JSP

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

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

O *directivă*⁶ 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.*, common.*" %>
```

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

Directiva `include` este folosită pentru a include resursa specificată⁷ în fișierul curent, în timp ce directivea `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
<code><jsp:useBean></code>	instanțierea unei componente
<code><jsp:setProperty></code>	stabilirea valorii proprietății unei componente
<code><jsp:getProperty></code>	obținerea valorii proprietății unei componente
<code><jsp:param></code>	obținerea valorii unui parametru transmis prin obiectul implicit <code>request</code>
<code><jsp:include></code>	includerea resursei specificate
<code><jsp:forward></code>	cererea este transmisă către o altă pagină JSP
<code><jsp:plugin></code>	generarea de marcaje HTML care comandă încărcarea unor aplicații specifice

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

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

utilizarea unei astfel de componente fiind ilustrată în exemplul anterior.

Un *scriptlet* 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.

⁶ Mai multe informații cu privire la utilizarea directivelor pot fi obținute de la adresa: <http://www.jsptube.com/jsp-tutorials/jsp-page-directive.html>.

⁷ În general, această metodă trebuie evitată preferându-se în schimb definirea de etichete.

O *expresie* este un scriplet care întoarce un rezultat⁸, fiind evaluat când este executat servlet-ul, rezultatul fiind convertit la tipul `String` și afișat.

```
<%
    ArrayList<Common.Record> adaugare = new ArrayList<Common.Record>();

    Enumeration e = request.getParameterNames();

    while(e.hasMoreElements())
    {
        String parameter = (String)e.nextElement();

        if (parameter.contains("adaugare_"))
            adaugare.add(
                new Common.Record(
                    parameter.substring(parameter.indexOf("_")+1),
                    request.getParameter(parameter)
                )
            );
    }

    bazadedate.add(TABLE,
        getAttributes(adaugare),
        getValues(adaugare),
        new PrintWriter (response.getWriter())
    );
%>
```

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

    public String getValues (ArrayList<Common.Record> records)
    {
        String result = "";
        for (Common.Record record: records)
        {
            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());
    }
%>
```

⁸ Spre diferență de scriplet care este cuprins între etichetele `<%` și `%>`, o expresie este cuprinsă între etichetele `<%=` și `%>`.

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

Obiect Implicit	Tip	Descriere
<code>request</code>	subclasă a <code>javax.servlet.HttpServletRequest</code>	cererea care a invocat pagina JSP
<code>response</code>	subclasă a <code>javax.servlet.HttpServletResponse</code>	răspunsul
<code>pageContext</code>	<code>javax.servlet.jsp.PageContext</code>	contextul paginii în funcție de serverul web
<code>session</code>	<code>javax.servlet.http.HttpSession</code>	obiect sesiune
<code>config</code>	<code>javax.servlet.ServletConfig</code>	
<code>application</code>	<code>javax.servlet.ServletContext</code>	context pagină JSP
<code>page</code>		
<code>out</code>	<code>javax.servlet.jsp.JspWriter</code>	obiect care scrie în fluxul de ieșire



Activitate de Laborator

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

`Laborator10_LibrarieVirtuala.sql`⁹.

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

[0p] 3. Să se copieze aplicația `Laborator10_LibrarieVirtuala` în directorul `webapps` al serverului Apache Tomcat.

[0p] 4. Să se completeze în clasa `DataBaseConnection` utilizatorul și parola pentru accesarea bazei de date.

[0p] 5. Să se compileze clasele ce vor fi folosite de aplicația web prin rularea script-ului `compile[.bat|.sh]` din directorul `scripts`.

[0p] 6. Să se testeze aplicația prin accesarea adresei http://localhost:8080/Laborator10_LibrarieVirtuala/

În script-ul `Laborator10_LibrarieVirtuala.sql` există exemple de utilizatori care pot fi folosite pentru accesarea paginilor de administrator, respectiv de client.

[3p] 7. În pagina `client.jsp`, să se determine conținutul coșului de cumpărături, ținând cont de situația în care pentru un produs care există deja în coșul de cumpărături se poate actualiza cantitatea.

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

[1p] 8. În pagina `client.jsp` să se afișeze conținutul coșului 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 coșul de cumpărături).

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

- se înregistrează comanda în baza de date (tabela `comenzi`);
- stocurile sunt actualizate corespunzător (tabela `carti`);
- coșul de cumpărături este golit.

⁹ În cazul în care operația nu a fost realizată în cadrul laboratorului precedent.

[1p] 11. În paginile `administrator.jsp` și `client.jsp` să se implementeze funcționalitatea prin care un utilizator se poate deconecta, fiind afișată 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, București, 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>