Aplicații integrate pentru întreprinderi

Laborator 7

21.12.2010

Realizarea de aplicații web folosind Java Servlets / JSP[1]

Scopul laboratorului îl reprezintă folosirea mecanismelor puse la dispoziție de tehnologiile Java Servlets / JSP pentru realizarea de aplicații web care să ofere utilizatorilor aceeaşi funcționalitate de care ar beneficia prin instalarea sau lansarea în execuție a unor programe care să solicite anumite resurse pe maşina unde sunt rulate.

1. Ce sunt tehnologiile Java Servlets / JSP ?
2. Ce aplicații trebuie instalate pentru a putea dezvolta aplicații web folosind Java Servlets / JSP ?
3. Care este ciclul de viață al unui Java Servlet / JSP ?
4. Care este structura unui Java Servlet ?
5. Care este structura unei pagini JSP ?
6. Cum este accesată baza de date ?
7. Cum este realizat controlul sesiunilor ?

1. Ce sunt tehnologiile Java Servlets / JSP ?

Tehnologiile Java Servlets / JSP reprezintă mecanisme diferite prin care pot fi dezvoltate aplicații web folosind limbajul de programare Java.

Un servlet reprezintă o clasă implementată în limbajul de programare Java, utilizată pentru a extinde capabilitățile unui server care găzduieşte aplicații accesate conform modelului cerere-răspuns. Cea mai frecvent întâlnită funcționalitate în legătură cu un servlet este legată de aplicațiile web, cu toate că acesta poate răspunde oricărui tip de cereri. Prin urmare tehnologia Java Servlet defineşte clase servlet adaptate protocolului HTTP.

Pachetele `javax.servlet` şi `javax.servlet.http` oferă interfețe şi clase pentru scrierea de Java Servlets. Orice servlet trebuie să implementeze interfața `Servlet` care defineşte metodele ce caracterizează ciclul de viață al unui astfel de obiect. Interfața `HttpServlet` oferă metode precum doGet şi doPost pentru a trata servicii specifice protocolului HTTP.

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.

[1] Java Server Pages

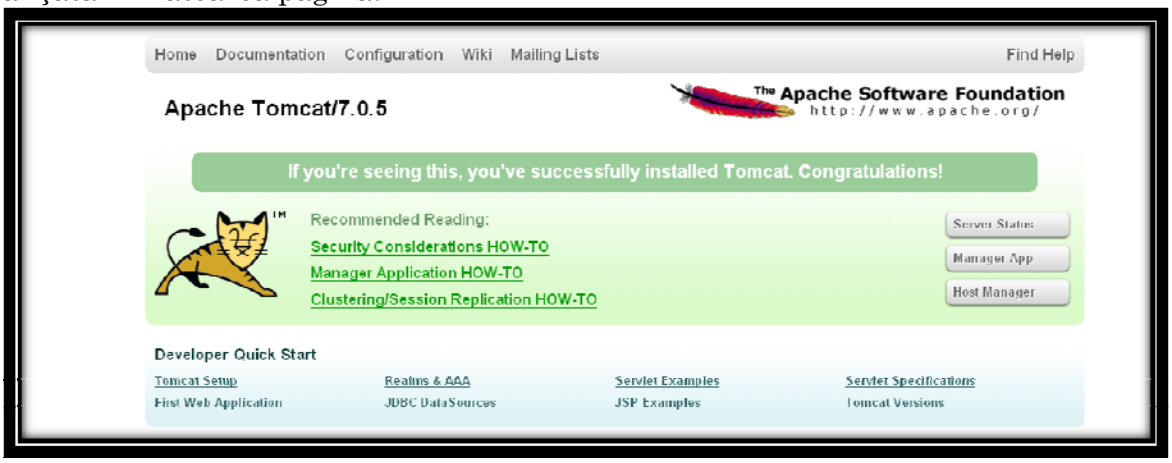
2. Ce aplicații trebuie instalate pentru a putea dezvolta aplicații web folosind Java Servlets / JSP ?

Pentru a dezvolta aplicații web folosind JSP este necesar un server web căruia să îi fie transmise cereri[2] pentru pagini de Internet ce se doresc a fi afişate Un astfel de server web trebuie să poată interpreta codul JSP şi să îl transforme în etichete HTML care să fie transmise mai departe spre client.

Este recomandată utilizarea serverului web Apache Tomcat[3].

Pentru configurare este necesară specificarea variabilelor de mediu[4] `JAVA_HOME` şi `JRE_HOME` în cazul în care ele nu sunt definite deja (în fişierul batch `setclasspath.bat` – Windows respectiv `setclasspath.sh` – Linux) după care lansarea în execuție se face prin fisierul batch `startup.bat` (Windows) respectiv `startup.sh` (Linux).

În momentul în care serverul Apache Tomcat rulează, în momentul în care adresa de internet http://localhost:8080 este specificată în browser, ar trebui afişată următoarea pagină:



Pagina web afişată atunci când rulează serverul web Apache Tomcat

Oprirea serverului web Apache Tomcat se face prin `shutdown.bat`, respectiv `./shutdown.sh`.

---

[2] O cerere este transmisă în momentul în care în browser este precizată o pagină disponibilă la adresa unde este instalat serverul web.

[3] Versiunea (BETA) 7.0.5 care suportă specificația 3.0 pentru tehnologia Java Servlet precum şi versiunea 2.2 pentru tehnologia JSP poate fi descărcată de la următoarea adresă de Internet: http://tomcat.apache.org/download-70.cgi.

[4] În Windows, specificarea variabilelor de mediu se face astfel:

```
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0_23\
set JRE_HOME=C:\Program Files\Java\jdk1.6.0_23\jre\
```

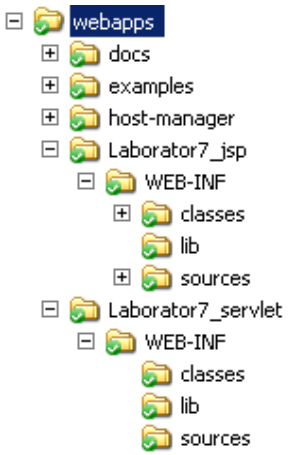
în timp ce pe Linux trebuie specificate comenzile:

```
export JAVA_HOME=/home/user/jdk1.6.0_23/
export JRE_HOME=/home/user/jdk1.6.0_23/jre/
```

Reinițializarea serverului web este necesară de fiecare dată când folosim tehnologia Java Servlets atunci când realizăm modificări la nivelul paginilor web deoarece clasele sunt „configurate” – *eng.* deployed (împreună cu dependențele) de fiecare dată când Apache Tomcat este lansat în execuție.

Sunt parcurse toate directoarele din `webapps` (unde sunt aplicațiile web) şi fiecare aplicație în parte este configurată în contextul serverului web care tocmai a fost lansat în execuție.

```
20.12.2010 17:02:41 org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory Laborator7_jsp
20.12.2010 17:02:41 org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory Laborator7_servlet
```



Structura de directoare în serverul de web Apache Tomcat

Orice aplicație web dezvoltată folosind serverul Apache Tomcat, indiferent dacă utilizăm Java Servlets sau JSP, trebuie plasată în directorul `webapps`.

În cazul unei aplicații web folosind **Java Servlets**, trebuie să existe un director **`WEB-INF`** care să conțină fişierul de configurare `web.xml`.

Acesta conține numele aplicației web (`servlet-name`) precum şi clasa care conține servlet-ul implementând practic serviciul web corespunzător (`servlet-class`).

O astfel de clasă trebuie plasată în mod obligatoriu în directorul **`classes`**[5]. Dacă aplicația web foloseşte biblioteci speciale care sunt apelate exclusiv în cadrul lansării în execuție(cum se întâmplă în cazul interacțiunii aplicației cu o bază de date prin intrmediul unui „driver”), atunci acestea trebuie plasate în directorul **`lib`**.

Aplicația web va fi disponibilă prin intermediul browser-ului la adresa `http://localhost:8080/<nume_director>/servlet-class/`.

```
<web-app>
 <servlet>
  <servlet-name>Catalog</servlet-name>
  <servlet-class>CatalogServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Catalog</servlet-name>
 <url-pattern>/CatalogServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Structura fişierului `web.xml`

În cazul unei aplicații web folosind **JSP**, în rădăcina directorului unde este aplicația web trebuie să se găsească fişierul `index.jsp`, care va fi afişat automat atunci când în browser este cerută adresa:
`http://localhost:8080/<nume_director>/`

[5] În exemplul de față există şi un director sources în care se găsesc sursele claselor şi care conține fişiere batch `deploy.bat` respectiv `deploy.sh` care compilează clasele, le mută în directorul `classes` şi reinițializează serverul web Apache Tomcat astfel încât modificările realizate la nivelul claselor să fie vizibile în cadrul aplicației.

De asemenea, în condițiile în care sunt folosite funcții din clase Java, acestea trebuie plasate într-un director `classes`, iar pentru utilizarea unor biblioteci speciale (de regulă, arhivate într-un fişier `.jar`), acestea trebuie puse într-un director `lib`.

3. Care este ciclul de viață al unui Java Servlet / JSP ?

Ciclul de viață al unui Java Servlet este controlat de mediul în care servlet-ul a fost configurat.

Atunci când o cerere este asociată unui servlet, container-ul care conține servlet-ul realizează următoarele acțiuni:

1. dacă nu există o instanță a servlet-ului
    a. încarcă clasa servlet;
    b. crează o instanță a clasei servlet;
    c. inițializează instanța clasei servlet prin apelarea metodei `init`;
2. apelează metoda `service` având ca parametri obiecte cerere şi răspuns.

În condițiile în care este necesară ştergerea servlet-ului, este apelată metoda `destroy` a acestuia.

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 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 compilează[6].

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.

Odată ce pagina JSP a fost transformată şi compilată, servlet-ul corespunzător va urma ciclul de viață corespunzător unui servlet care a fost descris anterior. Metodele se vor numi, în acest caz, `jspInit`, `_jspService`, `jspDestroy`.

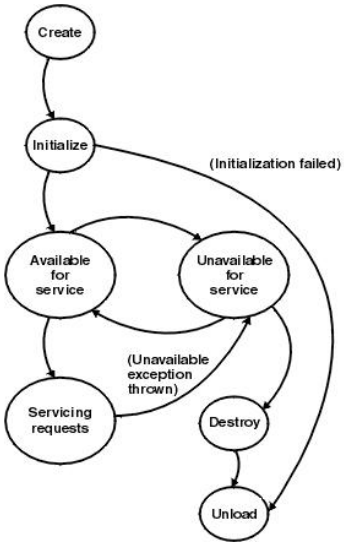
Atunci când se execută o pagină JSP pot apărea excepții, acestea trebuind tratate separat, în cadrul unei pagini dedicate unei astfel de acțiuni:

```
<@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"%>
```

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



Ciclul de viață al unei pagini JSP

În cele ce urmează, dezvoltarea unei aplicații web folosind tehnologiile Java Servlets / JSP va fi ilustrată pe cazul unei pagini care afişează situația şcolară a studenților anului IV C4.

Se va folosi baza de date situatia_scolara_4c4 care cuprinde două tabele:

- puncte_credit
- catalog

puncte_credit	
cod_disciplina	INT(4)
numar_puncte_credit	INT(1)

catalog	
numar_matricol	INT(4)
nume	VARCHAR(30)
prenume	VARCHAR(30)
grupa	VARCHAR(30)
management	INT(2)
proiect	INT(2)
pregatire_proiect_diploma	INT(2)
sisteme_de_prelucrare_grafica	INT(2)
inteligenta_artificiala	INT(2)
interactiunea_om_calculator	INT(2)
aplicatii_integrate_pentru_intreprinderi	INT(2)
disciplina1_optional	INT(2)
denumire_disciplina1_optional	VARCHAR(30)
invatare_automata	INT(2)
sisteme_cad_case	INT(2)
disciplina2_optional	INT(2)
denumire_disciplina2_optional	VARCHAR(30)

Se doreşte proiectarea unei aplicații web folosind pentru vizualizarea situației şcolare a studenților anului IV C4 permițând totodată adăugarea, editarea şi ştergerea de înregistrări.

Au fost realizate 2 aplicații, una folosind Java Servlets, alta folosind JSP pentru implementarea acestei funcționalități.

- aplicația folosind Java Servlets foloseşte clasa `CatalogServlet` derivată din `HttpServlet` care se ocupă atât cu transmiterea informațiilor din formular cât şi cu primirea (analizarea) lor;
- aplicația folosind JSP afişează informațiile în `index.jsp`, fiecare din operații fiind tratată în fişierele `adaugare.jsp`, `editare.jsp`, `stergere.jsp`;

4. Care este structura unui Java Servlet ?

```
import javax.servlet.*;
import javax.servlet.http.*;
...

public class CatalogServlet extends HttpServlet
{
    final public static long    serialVersionUID = 10241024L;

    ...

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
    }

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
           throws ServletException, IOException
    {
        ArrayList<String> values = new ArrayList<String>();

        ...

        Enumeration e = request.getParameterNames();

        while(e.hasMoreElements())
        {
            String parameter = (String)e.nextElement();
            if (parameter.contains("values"))
                values.add(request.getParameter(parameter)));
            ...
        }

        response.setContentType("text/html");

        PrintWriter pw = new PrintWriter (response.getWriter());

        displayForm(pw);

        pw.close();
    }

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
           throws ServletException, IOException
    {
        ...
    }
}
```

`HttpServlet` este o clasă abstractă prin care pot fi creați servleți HTTP care să funcționeze în contextul unei aplicații web.

```
public abstract class HttpServlet
extends GenericServlet
implements java.io.Serializable
```

Orice clasă derivată din aceasta trebuie să implementeze ***cel puțin*** una dintre metodele de mai jos:

- `doGet`, dacă servlet-ul tratează cereri `HTTP GET`;
- `doPost`, dacă servlet-ul tratează cereri `HTTP POST`;
- `doPut`, dacă servlet-ul tratează cereri `HTTP PUT`;
- `doDelete`, dacă servlet-ul tratează cereri `HTTP DELETE`;
- `init` şi `destroy`, pentru a gestiona resursele alocate în timpul în care servlet-ul este în execuție;
- `getServletInfo`, folosit de servlet pentru a oferi informații despre el;

Deşi la paragraful 3.2 am vorbit despre metoda `service` care este apelată atunci când există o instanță a clasei Servlet, nu este nevoie să fie implementată deoarece aceasta este declarată în clasa abstractă `HttpServlet`, implementarea apelând mai departe metoda responsabilă de tipul de cerere HTTP[7].

Un servlet se găseşte de regulă pe un server capabil să ruleze mai multe fire de execuție, astfel încât în condițiile unor cereri simultane, trebuie sincronizat accesul la resurse partajate, cum ar fi date din memorie (variabile tip instanță sau clase) sau obiecte externe (fişiere, conexiuni la baze de date sau conexiuni în rețea).

Câteva dintre metodele mai uzitate ale clasei `HttpServletRequest` sunt: `getContentType()`[8], `getCookies()`, `getHeaderNames()`, `getHeaders()`, `getSession()` `getInputStream()`[8], `getMethod()`, `getParameter()`[8], `getParameterNames()`[8].

Metodele cele mai folosite din clasa `HttpServletResponse` sunt moştenite din clasa `ServletResponse`: `flushBuffer()`, `get/setBufferSize()`, `get/setContentType()` `getOutputStream()`, `setCharacterEncoding()`, dar şi proprii: `addCookie()`, `get/setHeader()`, `get/setStatus()`.

5. Care este structura unei pagini JSP ?[9]

În cadrul unei pagini JSP se utilizează o combinație între linii tip 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 următoarele tipuri de elemente:

- directive;
- declarații;
- scripleți / expresii;
- acțiuni;
- etichete (tag-uri).

O *directivă*[10] este o instrucțiune care indică informații generale despre pagina respectivă. Sunt posibile următoarele directive: `page`, `include`, `taglib`.

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

[7] De asemenea, nu are rost să implementăm metodele `doOptions` şi `doTrace`.
[8] Aceste metode sunt moştenite din clasa `ServletRequest`.
[9] *cf.* Irina ATHANASIU – *Java, ca limbaj pentru programarea distribuită*, Editura MatrixRom, Bucureşti, 2002, pp. 80-85.
[10] Mai multe informații cu privire la utilizarea directivelor pot fi obținute de la adresa: http://www.jsptube.com/jsp-tutorials/jsp-page-directive.html.

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 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[11] în fişierul curent, în timp ce directiva `taglib` permite specificarea de etichete.

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

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());
    }
%>
```

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[12], fiind evaluat când este executat servlet-ul, rezultatul fiind convertit la tipul `String` şi afişat.

---

[11] În general, această metodă trebuie evitată preferându-se în schib definirea de etichete.
[12] Spre diferență de scriptlet care este cuprins între <% şi %>, o expresie este cuprinsă

```
<%
    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 *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="bazadedate" scope="page"
                                        class="Common.DataBaseConnection">
<jsp:setProperty name="bazadedate" property="*" />
</jsp:useBean>
```

utilizarea unei astfel de componente fiind ilustrată în exemplul anterior.

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

6. Cum este accesată baza de date ?

Ca în orice aplicație Java, accesul la baza de date se face folosind metodele din pachetul `java.sql.*`, biblioteca conținând „driver”-ul de conectare fiind plasată în directorul `lib` din structura de fişiere a aplicației web.

```
public class DataBaseConnection
{
    private String DataBase = "jdbc:mysql://localhost/...";

    private Connection connection;

    public void openConnection ()
    {
        try
        {
            connection = DriverManager.getConnection(DataBase);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public void closeConnection ()
    {
        ...
    }

    public String select
              (String table, String attributes, String where, String order)
    {
        openConnection();

        String result = "";

        ...
```

```
        String query = "SELECT "+attributes+
                       " FROM "+table+
              ((where!=null && !where.equals(""))?(" WHERE "+where):"")+
              ((order!=null && !order.equals(""))?(" ORDER BY "+order):"");

        try
        {
            Statement statement = connection.createStatement
            (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
            ResultSet source = statement.executeQuery (query);
            while (source.next())
            {
                for (String column: columns)
                    result += source.getString(column);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        closeConnection();

        return result;
    }
```

7. Cum este realizat controlul sesiunilor ?

Protocolul HTTP[13] este un protocol fără stări fiind caracterizat prin cereri şi răspunsuri ca tranzacții izolate.

Problema apare în momentul când trebuie să se coreleze mai multe cereri (care provin de la acelaşi utilizator).

Soluțiile presupun utilizarea de câmpuri ascunse, rescrierea URL-urilor pentru a include parametrii suplimentari, utilizarea de cookie-uri sau folosirea unor instrumente de „urmărire" a sesiunii.

Câmpurile ascunse pot fi conținute în formularele din paginile HTML (elemente de tip `<INPUT type="hidden" ...>`), dar au dezavantajul că pot fi identificate în textul HTML.

Rescrierea URL-urilor presupune adăugarea unei (aceleiaşi) informații la toate URL-urile paginilor care sunt transmise utilizatorului, informația fiind primită automat de server pentru cererile din pagina respectivă.

Ambele soluții presupun generarea dinamică a paginilor ca şi conținerea unui formular.

Cookie-urile sunt fişiere care conțin perechi de tipul `<cheie, valoare>`, fiind create de server şi transmise ca instrucțiuni în antetul mesajului HTTP transmis ca răspuns.

În pachetul `javax.servlet.http` este definită clasa `Cookie`:

```
public class Cookie
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable
```

[13] Spre diferență de protocolul FTP care este un protocol cu stări (clientul se conectează la server, realizează operațiile pe baza conexiunii după care se realizează deconectarea).

având un constructor care primeşte două şiruri de caractere reprezentând cheia, respectiv valoarea.

Pentru un obiect de tip `Cookie` se poate stabili timpul de expirare, domeniul, calea, el putând fi inclus într-un obiect de tip `HttpServletResponse`:

```
response.add(cookie);
```

Cookie-urile pot fi obținute prin metoda `getCookies()` implementată în clasa `HttpServletRequest`.

În pachetul `javax.servlet.http` este definită şi interfața `HttpSession`, care crează un singur obiect pentru o sesiune, putând stabili anumite valori pentru identificarea conexiunii dintre client şi server, legătura fiind realizată prin cookie (dacă sunt acceptate de client) sau rescrierea URL-urilor.

```
HttpSession session = request.getSession(true); // if not exists, create
Integer value = new Integer (count++);
session.setAttribute(„visitor”, value);
```

obținerea valorii realizându-se prin metoda `getAttribute(String attribute)`.

Activitate de laborator[14]

Se va instala baza de date `situatia_scolara_4c4` folosind scriptul `instalare_baza_de_date.sql`.

```
mysql> source <cale_absoluta>\instalare_baza_de_date.sql;
```

Se va descărca de pe Internet serverul de web Apache Tomcat.

Se vor completa în fişierul `setClasspath.bat | setClasspath.sh` căile către variabilele de mediu `JAVA_HOME` şi `JRE_HOME`.

Se vor copia directoarele `Laborator7_jsp` şi `Laborator7_servlet` în directorul webapps corespunzător serverului web.

Se vor compila clasele din directorul `Laborator7_servlet` manual sau folosind scriptul `deploy.bat | deploy.sh`.

Se va porni serverul de web Apache Tomcat prin intermediul comenzii `startup.bat | startup.sh`.

Se vor testa adresele:
http://localhost:8080/Laborator7_servlet/CatalogServlet/
http://localhost:8080/Laborator7_jsp/

[14] Aplicațiile pot fi realizate fie folosind Java Servlets fie folosind JSP.

(2p) 1. Să se afişeze pentru fiecare student în parte media aritmetică a notelor obținute la examene[15].
(3p) 2. Să se afişeze pentru fiecare student în parte media ponderată a notelor obținute la examene (prin realizarea unei operații de tip `join` cu tabela `puncte_credit`).
(2p) 3. Să se sorteze după diverse criterii (nume, prenume, medie) informațiile din tabelul conținând situația şcolară a studenților.
(3p) 4. În condițiile în care o operație de adăugare / editare / ştergere nu reuşeşte, se va afişa un mesaj de eroare, iar câmpurile de intrare de tip text din formular vor avea valorile introduse anterior.

Se va opri serverul de web Apache Tomcat prin intermediul comenzii `shutdown.bat | shutdown.sh`.

Se va dezinstala baza de date `situatia_scolara_4c4` folosind scriptul `dezinstalare_baza_de_date.sql`.

[15] Media se va calcula numai în condițiile în care studentul a promovat toate disciplinele din programa de învățământ, adică dacă este integralist. Pentru studenții cu restanțe, va fi afişat şirul de caractere `N-x`, unde `x` reprezintă numărul de examene nepromovate.