Aplicații Integrate pentru Întreprinderi

Laborator 9

29.11.2011

Realizarea de aplicații web folosind Java Servlets

Scopul laboratorului îl reprezintă folosirea mecanismelor oferite de tehnologia Java Servlets pentru dezvoltarea 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 un Java Servlet ?

Tehnologia Java Servlets este un mecanism prin care pot fi dezvoltate aplicații web folosind limbajul de programare Java, reprezentând o alternativă pentru CGI[1] (Common Gateway Interface).

Avantajele utilizării Java Servlets includ [1]: eficiența (inițializarea se face doar la încărcarea unui servlet, cererile fiind tratate prin apelarea metodei service), persistența (după ce este încărcat, obiectele unui servlet – ce pot conține informații din baza de date – sunt disponibile cât timp acesta este în execuție, ceea ce asigură o performanță ridicată comparativ cu încărcarea din baza de date a informațiilor fiecare dată), portabilitate (asigurată de limbajul de programare Java, astfel încât platforma pe care rulează aplicația poate fi schimbată fără modificarea codului sursă), robustețe (acces la toate mecanismele oferite de limbajul de programare Java – cu o ierarhie de excepții pentru tratarea erorilor şi colectarea memoriei disponibile), extensibilitate (fiind dezvoltat într-un limbaj orientat obiect, un servlet poate fi extins potrivit cerințelor aplicației) şi securitate (conform modelului de securitate specific Java).

Un servlet reprezintă deci 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.

[1] Ca tehnologie, CGI era caracterizată prin dependența de platformă şi lipsă de scalabilitate, eliminate prin Java Servlets.

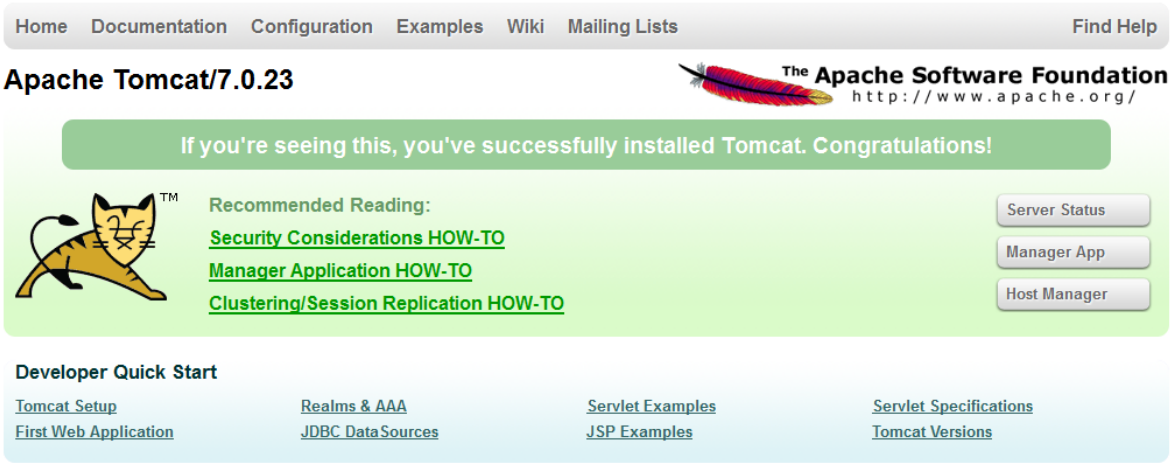
2. Integrarea Java Servlets cu Apache Tomcat

Pentru a dezvolta aplicații web folosind Java Servlets 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 Java, generând etichete HTML care să fie transmise mai departe spre client.

În cadrul laboratorului se va utiliza serverul 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 7.0.23 care suportă specificația 3.0 pentru tehnologia Java Servlet poate fi descărcată de la următoarea adresă de Internet: http://tomcat.apache.org/download-70.cgi. Versiunea pentru maşina virtuală Java trebuie să fie minim 1.6.

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

```
set JAVA_HOME= C:\Program Files\Java\jdk1.7.0\
set JRE_HOME=C:\Program Files\Java\jdk1.7.0\jre\
```

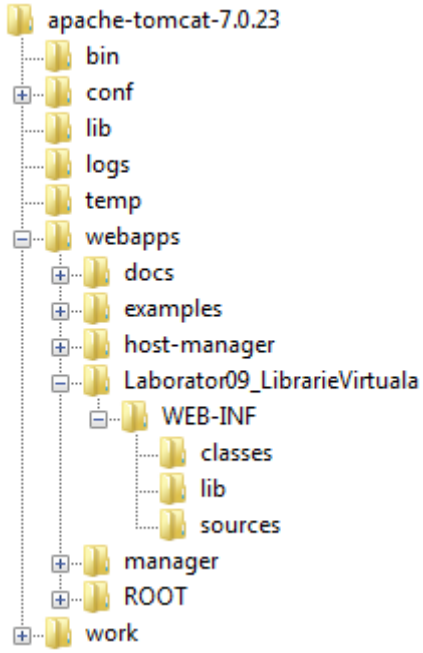
în timp ce pe Linux trebuie specificate comenzile:

```
export JAVA_HOME=/home/user/jdk1.7.0/
export JRE_HOME=/home/user/jdk1.7.0/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.

```
Nov 28, 2011 9:00:00 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory Laborator09_LibrarieVirtuala
```



Structura de directoare în serverul Apache Tomcat

Orice aplicație web dezvoltată folosind serverul Apache Tomcat trebuie plasată în directorul `webapps`.

În cazul unei aplicații web dezvoltate cu **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`)[5]. O astfel de clasă trebuie plasată obligatoriu în directorul **`classes`**[6]. În cazul când aplicația web foloseşte biblioteci apelate în cadrul lansării în execuție (cum e cazul la interacțiunea cu o bază de date prin intermediul unui „driver”), atunci acestea trebuie plasate în directorul **`lib`**[7].

Aplicația va fi disponibilă la adresa `http://localhost:8080/<locatie>/ClasaServlet/`.

```
<web-app>
 <servlet>
  <servlet-name>NumeServlet</servlet-name>
  <servlet-class>ClasaServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>NumeServlet</servlet-name>
 <url-pattern>/ClasaServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Structura fişierului `web.xml`

Pot fi specificate mai multe clase Java Servlet corespunzătoare paginilor care sunt proiectate.

---

[5] Opțional, poate fi specificat un parametru `load-on-startup` care, atunci când are valoare pozitivă indică ordinea în care sunt încărcate clasele respectiv, lasă la latitudinea container-ului care conține servlet-ul să încarce clasele atunci când este necesar, dacă este dată o valoare negativă.

[6] 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.

[7] Totuşi, în această situație, încărcarea driver-ului trebuie să se facă explicit (prin apelarea metodei `Class.forName("...")`), întrucât serverul Apache Tomcat nu realizează acestă operație atunci când aplicația web este configurată.

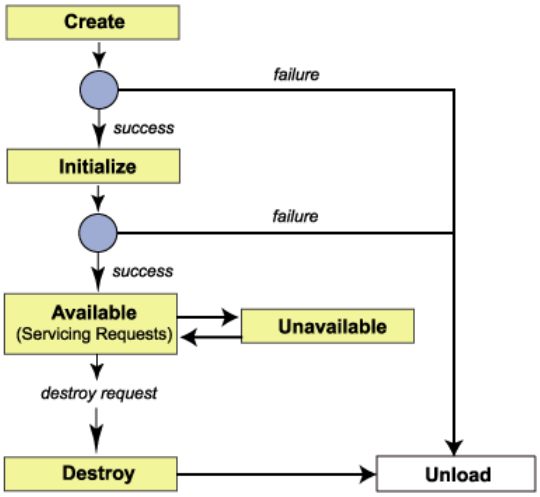
3. Care este ciclul de viață al unui Java Servlet ?

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 [2]:

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.



Ciclul de viață pentru un Java Servlet [3]

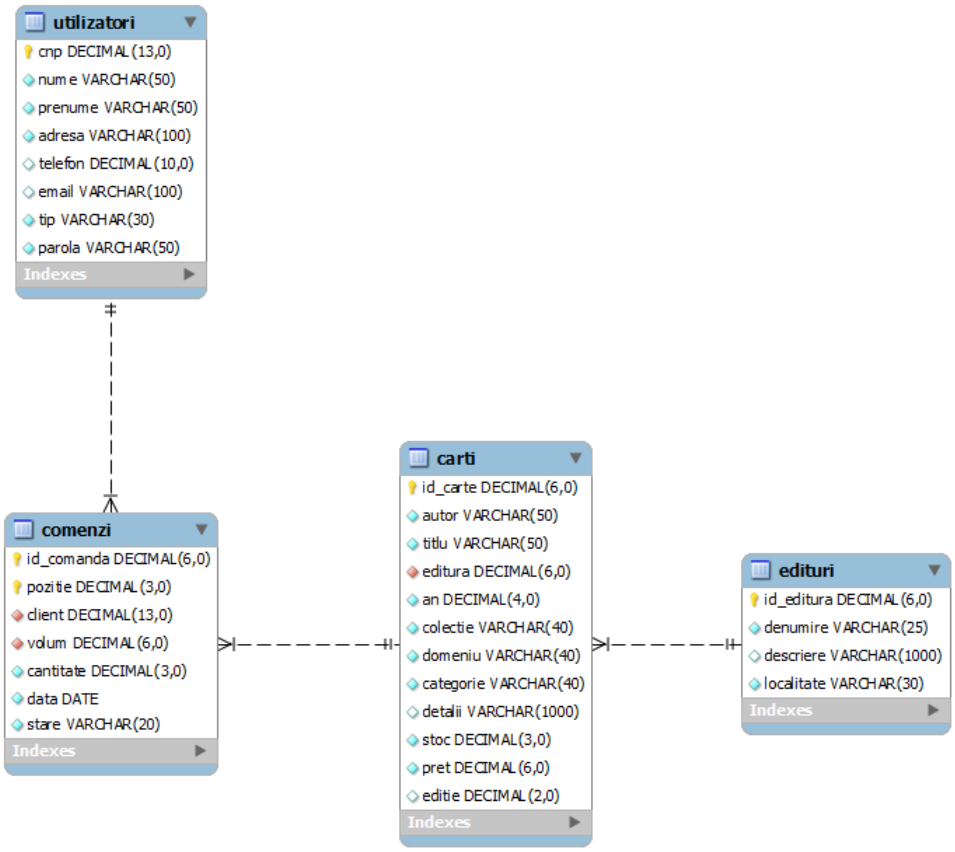
Se pot defini obiecte care reacționează la evenimentele din ciclul de viață al unui Java Servlet (folosind adnotarea `@WebListener`):

Obiect	Eveniment	Interfață (ce trebuie) Implementată	Obiect Eveniment
aplicație web	creare, distrugere	`javax.servlet.ServletContextListener`	`ServletContextEvent`
	operații asupra atributelor	`javax.servlet.ServletContextAttributeListener`	`ServletContextAttributeEvent`
sesiune	creare, distrugere, activare, dezactivare	`javax.servlet.http.HttpSessionListener` `javax.servlet.http.HttpSessionActivationListener`	`HttpSessionEvent`
	operații asupra atributelor	`javax.servlet.http.HttpSessionAttributeListener`	`HttpSessionBindingEvent`
cerere	procesarea cererii	`javax.servlet.ServletRequestListener`	`ServletRequestEvent`
	operații asupra atributelor	`javax.servlet.ServletRequestAtrributeListener`	`ServletRequestAttributEvent`

Exemplu

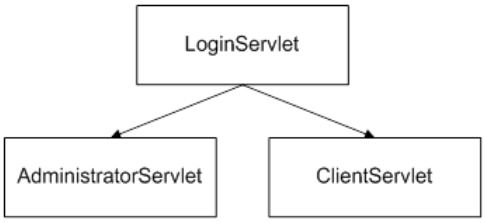
Se doreşte dezvoltarea unei aplicații care să implementeze funcționalitatea pentru o librărie virtuală.

Aplicația *Librărie Virtuală* va fi accesată de un utilizator tip administrator (care va gestiona[8] – prin intermediul aplicației – informațiile din baza de date) şi de un utilizator tip client, care va plasa o comandă[9] după ce a specificat un anumit coş de cumpărături după consultarea catalogului de produse.



Structura conceptuală a bazei de date

Au fost definite clase Java Servlet corespunzătoare paginilor din aplicație: pagina de autentificare (`LoginServlet`) de unde, pe baza tipului de utilizator identificat pot fi accesate pagina de administrare (`AdministratorServlet`) respectiv pagina client (`ClientServlet`).



[8] Operațiile realizate un utilizator de tip administrator sunt adăugare, editare şi ştergere informații pentru oricare dintre tabelele din baza de date.

[9] O comandă este definită prin combinația (`id_comanda`, `pozitie`), unde `id_comanda` reprezintă identificatorul comenzii (un număr de ordine) în timp ce `pozitie` specifică un produs din cadrul comenzii respective (comanda este formată din unul sau mai multe produse).

4. Care este structura unui Java Servlet ?

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

public class CatalogServlet extends HttpServlet
{
    final public static long    serialVersionUID = 10001000L;

    ...

    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 parameters = request.getParameterNames();

        while(parameters.hasMoreElements())
        {
            String parameter = (String)parameters.nextElement();
            if (parameter.contains("..."))
                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
    {
        ...
    }

    public void destroy()
    {
        ...
    }
}
```

Întrucât un servlet se găseşte pe un server care poate fi accesat de către mai mulți clienți simultan, trebuie asigurată sincronizarea resurselor partajate (variabile ale claselor, fişiere, conexiuni la baza de date sau la rețea).

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

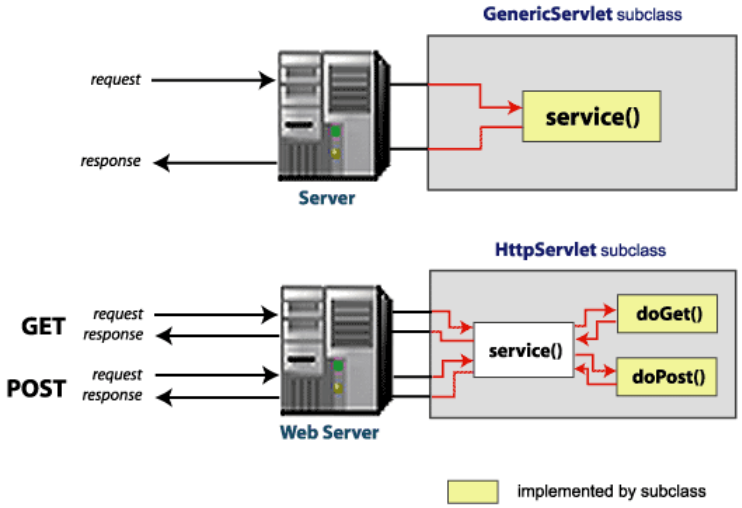
```
public abstract class HttpServlet
extends GenericServlet implements java.io.Serializable
```

Orice clasă derivată din `HttpServlet` trebuie să implementeze ***cel puțin*** una dintre metodele:

- `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`[10], pentru a gestiona resursele alocate în timpul în care servlet-ul este în execuție;
- `getServletInfo`, folosit de servlet pentru a oferi informații despre el;

Nu este necesară implementarea metodei `service` (apelată atunci când există o instanță a clasei Servlet), deoarece este definită în clasa abstractă `HttpServlet`, implementarea apelând mai departe metoda responsabilă de tipul de cerere HTTP[11].

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).



Tratarea cererilor de către un servlet generic / servlet HTTP [3]

Câteva dintre metodele mai folosite ale clasei `HttpServletRequest` sunt: `getContentType()`[12], `getCookies()`, `getHeaderNames()`, `getHeaders()`, `getSession()` `getInputStream()`[12], `getMethod()`, `getParameter()`[12], `getParameterNames()`[12].

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()`.

[10] În general, în metodele `init` şi `destroy` sunt alocate şi dezalocate resurse partajate cum ar fi conexiunea la baza de date sau accesul la fişiere.
[11] De asemenea, nu este necesară implementarea metodelor `doOptions` şi `doTrace`.
[12] Aceste metode sunt moştenite din clasa `ServletRequest`.

Pentru a se defini un servlet în cadrul unei aplicații web poate fi folosită adnotarea `@WebServlet`[13] ce poate conține metadate referitoare la locația de unde este disponibilă componenta respectivă (se folosesc atributele `urlPatterns`/`value`[14] toate celelalte atribute fiind opționale). O clasă servlet trebuie să extindă clasa `javax.servlet.http.HttpServlet`.

Metoda `init()` este apelată (de către container-ul care conține servlet-ul) după încărcarea şi crearea unei instanțe a clasei Servlet şi înainte de a procesa cereri şi răspunsuri prin care este realizată comunicația cu un client. Aici, serverul va realiza operații care trebuie executate doar o dată (configurări, inițializări ale unor resurse[15]). În cazul în care procesul de inițializare nu poate fi completat, se va arunca o excepție de tipul `UnavailableException`.

Funcționalitatea pusă la dispoziție de către un servlet este implementată în metoda `service`[16] a clasei `GenericServlet`, prin apelarea metodelor corespunzătoare `doMethod()` (unde `Method` poate avea – aşa cum am văzut mai sus – valorile `Get`, `Post`, `Put`, `Delete`, `Options` sau `Trace`) ale unui obiect `HttpServlet` sau prin metode specifice protocolului definite de o clasă care implementează interfața `Servlet`. O metodă serviciu va primi informațiile din cerere (obiect tip `HttpServletRequest`) apelând `getParameterNames()`/`getParameter()` şi va transmite informațiile în răspuns (obiect tip `HttpServletResponse`) – folosind un obiect `PrinterWriter` asociat acestuia (obținut prin metoda `getWriter()`).

O cerere conține datele transmise de la client către servlet şi trebuie să implementeze interfața `ServletRequest`, unde sunt definite metode pentru accesarea de: parametri (pentru comunicarea de informații între clienți / servleți) atribute – instanțe ale unor obiecte (folosite la comunicarea dintre un container al unui servlet şi servlet sau pentru comunicarea între mai mulți servleți) şi informații despre protocolul prin care este transmisă cererea sau despre locație[17]. Obiectul `HttpServletRequest` transmis prin cerere conține URL-ul cererii[18], antetele HTTP şi interogarea formată din perechi de parametri şi valori.

Un răspuns conține datele transmise de la servlet către client şi trebuie să implementeze interfața `ServletResponse`, unde sunt definite metode pentru obținerea unui flux prin care se poate realiza comunicarea cu clientul[19], indicându-se tipul de conținut (prin metoda `setContentType`, al cărei parametru este `text/html`), dacă se aloca o zonă de memorie (metoda `setBufferSize()` oferă un timp înainte de stabilirea unor coduri de stare) şi informații despre locație.

---

[13] Această adnotare este definită în pachetul `javax.servlet.annotation.*`.

[14] Atributul `value` este folosit în cazul în care nu mai sunt definite şi alte atribute ale adnotării, iar în caz contrar se foloseşte atributul `urlPatterns`.

[15] Se poate folosi atributul `initParams` al adnotării `@WebServlet` ce conține o adnotare `@WebInitParam`.

[16] Denumirea de metodă serviciu este folosită pentru orice metodă dintr-o clasă servlet care oferă o funcționalitate pentru un client.

[17] Există posibilitatea de a analiza datele folosind un obiect `BufferedReader` creat dintr-un obiect `ServletInputStream` întors de metoda `getInputStream`.

[18] URL-ul cererii are forma [http://[adresa]:[port]/[cale]?[interogare]] şi conține: calea contextuală (contexul aplicației web în care rulează servlet-ul), calea către servlet (care indică componenta care procesează cererea) precum şi alte informații. Acestea pot fi obținute apelând metodele: `getContextPath`, `getServletPath` şi `getPathInfo`.

[19] Pentru transmiterea de date de tip caracter se foloseşte un obiect `PrinterWriter` întors de metoda `getWriter`, în timp ce pentru transmiterea de date binare se foloseşte un obiect `ServletOutputStream` întors de metoda `getOutputStream`.

Obiectul `HttpServletResponse` are câmpuri care reprezintă antete HTTP ce conțin coduri de stare, pentru a indica motivele pentru care o cerere nu poate fi satisfăcută sau faptul că s-a realizat redirecționarea către altă resursă ca şi pentru transmiterea unor obiecte ce vor reține informații specifice aplicației (eventual, legate de sesiune).

Un container ce conține un servlet poate determina distrugerea acestuia (în cazul în care se realizează colectarea memoriei disponibile sau atunci când acesta se închide), apelându-se metoda `destroy` din interfața `Servlet`. Aici toate resursele utilizate de către servlet trebuie eliberate, asigurându-se şi persistența prin reținerea informațiilor necesare în baza de date. Toate metodele serviciu corespunzătoare unui servlet trebuie să fie terminate atunci când container-ul urmează să îl distrugă. Astfel, metoda `destroy` este apelată doar după ce toate metodele serviciu s-au terminat sau după expirarea unei anumite perioade stabilită de server. Trebuie ca toate firele de execuție pe care sunt realizate operații de către server să se termine atunci când este apelată metoda `destroy`[20].

5. Cum se asigură interacțiunea cu baza de date într-un Java Servlet ?

Ca în orice aplicație Java, accesul la baza de date se face folosind metodele puse la dispoziție prin API-ul JDBC (pachetul `java.sql.*`), biblioteca ce conține „driver”-ul de conectare fiind plasată în directorul `lib` din structura aplicației.

```
import java.sql.*;
import java.util.*;
import java.io.*;

public class DataBaseConnection
{
    private String DataBase =
                            "jdbc:mysql://localhost/librarie_virtuala?...";
    private Connection connection;

    public void openConnection()
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(DataBase);
        }
        catch (Exception e) {    e.printStackTrace();    }
    }

    public void closeConnection()
    {
        try
        {
            connection.close();
        }
        catch (Exception e) {    e.printStackTrace();    }
    }
```

[20] O soluție ar fi contorizarea numărului de fire de execuție care rulează şi invocarea `Thread.sleep()` cât timp acestea nu s-au terminat.

```
    public String select(String table, String where, String order)
    {
        String result = "<table border=\"1\">";

        ArrayList<String> columns = new ArrayList<String>();
        StringTokenizer values =
                                new StringTokenizer (getColumns(table),",");

        while (values.hasMoreTokens())
            columns.add(values.nextToken());

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

        result += "<tr>";

        for (String column: columns)
            result += "<td>"+column+"</td>";

        result += "</tr>";

        try
        {
            Statement statement =
                      connection.createStatement
                      (ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
            ResultSet source = statement.executeQuery(query);

            while (source.next())
            {
                result += "<tr>";
                for (String column: columns)
                    result += "<td>"+source.getString(column)+"</td>";
                result += "</tr>";
            }
        }
        catch (Exception e)    {    e.printStackTrace();    }

        result += "</table>";

        return result;
    }
    ...
}
```

6. Cum se realizează controlul sesiunilor ?

Protocolul HTTP[21] 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 accesări (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.

[21] 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).

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

Rescrierea URL-urilor presupune adăugarea unei (aceleiaşi) informații la 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
```

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

Pentru un obiect de tip `Cookie` se poate stabili numele şi valoarea (metodele `set/get Name` şi `Value`), timpul de expirare, domeniul sau calea, acesta putând fi inclus într-un obiect de tip `HttpServletResponse`:

```
response.addCookie(cookie);
```

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

```
Cookie[] cookies = request.getCookies();
```

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);
```

Metoda `getSession()`[22] întoarce sesiunea asociată unei cereri, sau dacă nu există o sesiune asociată cererii, aceasta este creată, dacă se specifică astfel.

Obținerea atributelor este realizată prin metodele `getAttributeNames()`, respectiv `getAttribute(String)`, iar stabilirea lor prin metoda `setAttribute(String,Object)`. Totodată, un atribut poate fi eliminat prin metoda `removeAttribute(String)`.

Întrucât un client HTTP nu poate specifica momentul în care sesiunea nu mai este necesară, este asociat un timp de expirare (prin metodele `set/get MaxInactiveInterval`), astfel încât resursele utilizate de sesiune să poată fi refolosite. O sesiune nu va expira în cazul în care este accesată periodic (frecvența fiind mai mare decât timpul de expirare) – prin metodele serviciu care inițializează (din nou) perioada în care sesiunea poate fi utilizată. Totodată, atunci când interacțiunea cu un client HTTP este încheiată, se poate folosi metoda `invalidate` pentru a distruge sesiunea eliberând resursele folosite.

[22] Metoda `getSession` poate fi apelată şi fără parametri: `getSession() = getSession(true)`.

Activitate de Laborator

[0p] 1. Să se instaleze baza de date prin rularea script-ului (specific) `Laborator09_LibrarieVirtuala.sql`.
[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 copieze aplicația `Laborator09_LibrarieVirtuala` în directorul `webapps` al serverului Apache Tomcat.
[0p] 4. Să se completeze în clasa `DataBaseConnection` utilizatorul şi parola pentru accesarea bazei de date.
[0p] 5. Să se dezvolte aplicația prin rularea script-ului `deploy[.bat|.sh]` din directorul `scripts`.
[0p] 6. Să se testeze aplicația prin accesarea adresei http://localhost:8080/Laborator09_LibrarieVirtuala/LoginServlet.

În script-ul Laborator09_LibrarieVirtuala.sql există exemple de utilizatori care pot fi folosite pentru accesarea paginilor de administrator, respectiv de client.

[4p] 7. În clasa `ClientServlet`, metoda `doPost`, 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 clasa `ClientServlet`, metoda `displayForm`, să se afişeze conținutul coşului de cumpărături.
[1p] 9. Să se adauge un buton prin care se poate specifica o comandă (o comandă conține produsele selectate în coşul de cumpărături).
[4p] 10. În clasa `ClientServlet`, metoda `doPost`, să se implementeze operația pentru transmiterea unei comenzi:

- se înregistrează comanda în baza de date;
- stocurile sunt actualizate corespunzător;
- coşul de cumpărături este golit.

Bibliografie

[1] James Goodwill, *Developing Java Servlets – Web Applications with servlets and JSP*, 2nd Edition, SAMS, 2001
[2] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, Chinmayee Srivathsa, *The Java EE 6 Tutorial, Basic Concepts*, 4th Edition, Addison Wesley, 2011
[3] *Java Servlets* – School of Electronic Engineering, Dublin City University, http://wiki.eeng.dcu.ie/ee448/g2/850-EE.html