Introducere in programarea web in Java

In acest material voi prezenta FOARTE pe scurt niste pasi necesari pentru a incepe sa programam aplicatii web in Java.
O aplicatie web este, in principiu, un set de pagini web afisate utilizatorului si un set de servicii accesibile prin accesarea a mai multe URLuri. Practic, interfata grafica realizata de exemplu cu Swing este inlocuita de interfata realizata in HTML/JS/CSS, iar comunicarea pe retea prin Socketuri este inlocuita de comunicarea browser-server prin protocolul HTTP.

Dupa cum stim, daca vrem sa facem un server in Java, trebuie sa construim un ServerSocket si apoi sa obtinem un Socket care este o conexiune cu clientul. Apoi obtinem cele doua Stream (input/output) si incepem comunicarea. Toate aceste aspecte sunt de cele mai multe ori destul de complicate (adica necesita mult cod) si nu mereu este nevoie de comunicarea bidirectionala specifica unui Socket sau de toata flexibilitatea oferita de el. Atunci putem sa ne bazam pe un set de functii “standard”. In plus, pentru o aplicatie web partea de client este deja scrisa (browserul) deci nu trebuie sa ne batem prea mult capul cu implementarea interfetei grafice, captura de evenimente, etc.

Pentru a putea folosi toate elementele scrise deja de altii pentru a implementa o aplicatie web, trebuie sa facem insa o aplicatie care respecta numeroase reguli suplimentare fata de o aplicatie java obisnuita. Prima din aceste reguli este ca aplicatia odata construita nu este lansata cu java.exe ci este “deployed” pe o alta aplicatie (tot java) care o sa o numim “container web”. Acest “container” este responsabil cu gestionarea comunicarii pe retea, paralelism, managementul resurselor etc (adica partea cea mai generica si “multa” din aplicatie).

Containerul de web

Exista o multitudine de containere de web, unele sunt bine cunoscute altele mai putin cunoscute. Fiecare din ele ofera anumite facilitati specifice, dar si un nucleu fundamental care este comun tuturor. Noi vom folosi un container gratuit, oferit de Apache, numit Tomcat.
Puteti descarca un Tomcat de la adresa:
https://tomcat.apache.org/download-70.cgi

Voi prezenta in continuare cateva aspecte esentiale pentru configurarea initiala a unui Tomcat. In functie de ceea ce vreti sa faceti este posibil sa aveti nevoie de toate aceste configurari sau doar de o parte din ele.

- Tomcat este descarcat sub forma de arhiva si nu trebuie instalat. Dezarhivati totul in directorul unde vreti sa aveti tomcatul.
- Optional, se poate configura IPul si portul pe care vreti sa asculte serverul. Este posibil sa nu vrem sa fie vizibil din exterior serverul si atunci putem seta sa asculte doar de localhost.

- Pentru a face deploy pe tomcat este nevoie de acces pe consola de administrare. Aceasta consola este protejata de un username+parola care sunt setate in fisierul tomca-users.xml din directorul conf. Exista cateva exemplu acolo, dar ceea ce ne trebuie noua este un user care arata cam asa:

```
  <role rolename="tomcat"/>
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <role rolename="manager-jmx"/>
  <role rolename="manager-status"/>

  <user username="tomcat" password="admin" roles="tomcat,
manager-gui, manager-script, manager-jmx, manager-status"/>
```

- Acum pornim serverul, apeland startup.bat din directorul bin. Atentie, pentru oprire se recomanda folosirea lui shutdown.bat din acelasi director. Nu inchideti fereastra care apare dupa startup.
- Conectati-va la tomcat la adresa http://localhost:8080 (portul nu este 80 by default ca la orice alt server web)
- In partea dreapta avem un buton “Manager App”. Dati click pe el pentru a accesa aplicatia de deployment. Introduceti aici usernameul si parola configurate mai sus (tomcat/admin). Acum tot ce ne mai trebuie este o aplicatie web.

Prima aplicatie web

O aplicatie web este de fapt o colectie de resurse/cod/setari care sunt impachetate intr-o arhiva. Arhiva pentru aplicatiile web are extensia “war” (web archive).
Pentru a construi o aplicatie web, deschideti un Eclipse si apoi:

1. Construiti un nou proiect de tipul “Dynamic web project”. Ca nume de proiect va rog sa folositi numele vostru si grupa. De exemplu “IonGheorghe234”. Asta pentru ca veti face deploy pe acelasi server si nu vrem sa avem conflicte de nume. Analizati un pic structura care se construieste. In “Java Resources” vedem directorul src unde o sa punem codul nostru sursa. Un alt director important este WebContent care contine fisierele html/jsp. In acest director exista doua subdirectoare, META-INF si WEB-INF. O importanta aparte o are directorul WEB-INF care nu va putea fi accesat din exterior in mod direct. In acest director exista un subdirector numit lib unde sunt puse librariile externe (noi am folosit JDBC ca librarie externa, de exemplu). Tot in WEB-INF se afla niste fisiere xml cu denumiri generice (web.xml) sau specifice serverului pe care se va face deployment la aplicatie (sun-web.xml pentru Glassfish, de exemplu). Aceste fisiere XML se numesc “deployment” descriptor si practic configureaza serverul si aplicatia la deployment. Acest fisier web.xml era foarte folosit in trecut, acum insa multe configurari se pot face folosind anotari in clasele Servlet (si altele). Eclipse nici macar nu va construi acest fisier.
2. Dati click dreapta pe WebContent si adaugati un nou fisier html (“index.html”). Deschideti fisierul. Veti recunoaste structura HTML.
3. Adaugati in body textul “Iata un numar: 12345”

4. Acum haideti sa construim aplicatia si sa o punem pe server. Dati click dreapta pe proiect si alegeti “Export” apoi “WAR file”. Alegeti o destinatie pe care o doriti.
5. Mergeti acum in browser in consola Tomcat gasiti sectiunea “War to deploy” din pagina consolei. Dati click pe butonul pentru selectie fisier si alegeti fisierul War salvat mai devreme. Apasati “Deploy”.
6. Observati in lista de Applications ca a aparut aplicatia voastra. Dati click pe numele ei pentru a fi redirectati. Acum ati accesat aplicatia voastra!

Dinamizarea aplicatiei

Pana acum avem o aplicatie statica, fara nimic interesant in ea. Haideti sa ii adaugam continut dinamic.
Pentru a face serverul de web sa modifice pagina web inainte de a o trimite clientului trebuie sa ii schimbam extensia din html in jsp.
Haideti sa stergem acest fisier si sa facem unul nou (astfel IDEul va sti ca e JSP si ne va ajuta cu validarea codului). Deci faceti un nou fisier de tip JSP numit index.jsp.
In acest moment o sa putem sa adaugam marcaje JSP in fisier si in interiorul lor, cod Java.
Haideti sa facem numarul care apare acolo sa fie aleator.

1. Stergem numarul hardcodat
2. Adaugam un marcaj <% care spune serverului ca incepe cod Java
3. Adaugam cod pentru generarea unui numar aleator: `double numar = Math.random();`
4. Afisam numarul in pagina. `out.write(Double.toString(numar));` Atentie! Nu mai folosim System.out.println!
5. inchidem marcajul JSP cu %>
6. Exportam si facem deploy din nou.

Primul servlet

Atunci cand scriem accesam un URL de pe un server web scris in java va fi chemata de fiecare data o rutina numita “servlet”. Acesta este responsabil de servirea paginii. Chiar daca voi nu vedeti, atunci cand un JSP este accesat, un servlet (care nu e scris de voi) este cel care raspunde si produce textul vizibil de client.
Haideti sa facem un Servlet.

1. Expandati JavaResources si folderul src.
2. Faceti un nou pachet (numit cum vreti voi)
3. In pachet faceti un nou Servlet. Vedeti ca IDEul ne ajuta din nou. Haideti sa ii spunem “NumarAleator”. Dati click pe next.
4. Acum putem adauga parametrii de initializare ai servletului si maparea. Maparea este calea de care raspunde servletul relativ la URLul aplicatiei. Lasati totul asa cum este in aceasta pagina. Dati next.
5. Vedem acum ce metode vrem sa fie generate automat. Dupa cum vedeti, sunt destul de multe. Ar trebui sa stiti despre ele de la curs, dar ce ne intereseaza pe noi acum sunt

metodele doGet si doPost care raspund atunci cand se fac cererile respective de catre browser. Lasati totul asa cum este in acea fereastra. Dati Finish.

6. Vedem acum o multime de erori. Modivul este ca nu avem in classpath libraria care contine practic toata partea de programare web in java, si pachetul javax.
7. Dati click dreapta pe proiect, build path, configure build path, Libraries, Add External Jar si alegeti fisierul servlet-api.jar din directorul lib din directorul unde ati pus Tomcatul.
8. Analizati un pic codul generat.
9. Haideti sa modificam astfel incat sa afisam lata un numar: 0.221345 (numar aleator)
10. In metoda doGet generati un numar aleator ca in JSP
11. in loc de ceea ce se afiseaza acum, adaugati "`response.getWriter().append("Iata un numar: ").append(Double.toString(numar));`"
12. Faceti export si redeploy.
13. Accesati servletul folosind hocalhost://numeAplicatie/NumarAleator.
14. Observati ca outputul nu mai este HTML, este exact ceea ce ii dam noi sa afiseze in servlet.

Accesarea sesiunii

In momentul in care accesati o pagina web, ceea ce faceti voi acolo este urmarit de catre server pentru a putea sa va serveasca. De exemplu, dupa ce v-ati logat, ati vrea ca serverul sa stie ca sunteti logati si sa nu va ceara parola de fiecare data. Toate informatiile de acest gen sunt stocate in "sesiune". Pentru a vedea cum folosim aceasta sesiune, haideti sa facem ca servletul sa salveze numarul generat aleator intr-un camp din sesiune si JSPul in alt camp, si ambele sa afiseze cele doua valori.

1. Modificati servletul astfel incat numarul sa fie salvat in variabila "servletNum" din sesiune: `request.getSession().setAttribute("servletNum", numar);`
2. Afisati apoi variabilele de sesiune "servletNum" si "jspNum" folosind getAttribute din sesiune. Aveti grija la null!
3. Modificati JSPul astfel incat numarul generat sa fie salvat in variabila "jspNum" si apoi afisati valorile celor doua atribute.
4. Faceti export si redeploy.
5. Accesati pe rand JSPul si Servletul

Pasarea unui parametru din servlet in JSP

Putem sa folosim un Servlet pentru a face "munca grea" cum ar fi accesarea bazei de date si procesarea informatiilor de acolo si un JSP pentru a afisa aceste informatii.
Pentru a face asta, putem sa setam anumite valori in request in servlet si apoi sa "forward" requestul catre un JSP. Valorile se vor regasi in request. Haideti sa facem ca servletul sa genereze numarul aleator si apoi JSPul doar sa il afiseze.

1. Modificati servletul astfel incat doar sa fie generat numarul aleator.
2. Setati numarul in request : `request.setAttribute("number", numar);`

3. Trimiteti apoi requestul spre procesare in JSP:
   `getServletContext().getRequestDispatcher("/index.jsp").forward(request, response);` Forward inseamna ca requestul continua cu procesarea de catre servlet.
4. Modificati acum JSPul astfel incat sa nu mai genereze numarul ci sa il citeasca :
   `double numar = (Double)request.getAttribute("number");`
5. Afisati numarul
6. Faceti redeploy si accesati servletul.

Este important sa intelegeti diferenta intre forward si redirect. Forward inseamna ca cererea este pasata de catre server unui alt servlet, in timp ce redirect spune clientului sa acceseze un alt url.

Folosirea bazei de date

Pentru a ne conecta la baza de date in servlet nu vom folosi aceeasi metoda ca la programarea desktop. Vom configura serverul Tomcat astfel incat sa ne ofere un connection pool si noi vom solicita de acolo conexiuni pe care le vom folosi.
Configurarea se face tot din fisierele specifice aplicatiei. Deschideti directorul META-INF. In el, creati un nou fisier numit context.xml. In acest fisier vom defini resursa reprezentand conection poolul:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/myApp">
     <Resource name="jdbc/myDb" auth="Container"
type="javax.sql.DataSource"
          username="fooUser" password="fooPassword"
driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://82.76.115.105:3306/pao" maxActive="15"
maxIdle="3" />
</Context>
```

Observati cat de similar este cu elementele pe care le-am invatat data trecuta.

Dupa ce am configurat resursa, a venit momentul sa o folosim.

In servlet trebuie sa adaugam o variabila membru ne statica:

```
@Resource(name = "jdbc/myDb")
private DataSource dbRes;
```

Evident, numele variabilei este ales de voi si poate fi orice. Acum putem sa facem in cod:

```
Connection con = dbRes.getConnection();
```

Astfel am obtinut un Connection si de aici stim cum sa folosim baza de date. Nu uitati sa inchideti resursele!

Pentru a folosi baza de date avem nevoie de driver. Putem adauga driverul in directorul lib din tomcat si astfel va fi vizibil in toate aplicatiile sau vom putea sa il adaugam in directorul lib din WEB-INF pentru a fi accesibil doar din aplicatia noastra.
https://drive.google.com/open?id=0B5ar2tHw-X9vUDNrc05mR1AycWs

Exercitiu 1:

Afisati lista studentilor intr-un tabel cu coloanele “nume student” si “prezente”. Puteti sa amestecati cod Java cu cod HTML in felul urmator:

```
<ul>
<%
for(int i=0;i<100;i++)
{
   %><li><% out.write(i); %></li><%
}
%>
</ul>
```

Exercitiu 2:

Modificati lista studentilor astfel incat numele studentilor sa fie un link catre o pagina in care sa putem vedea cele doua campuri (prezente si nume) intr-un formular.

Exercitiu 3:

Implementati un serviciu prin care sa puteti modifica studentii (update).

Notificare importanta:

Serverul de Tomcat la care v-am dat acces ruleaza pe o masina virtuala care va fi reinitializata cand consider eu. NU STOCATI DATE PE EL! NU FACETI DEPLOY LA PROIECTE PE EL! Cel mai probabil le veti pierde.

http://82.76.115.105:8080/