Universitatea POLITEHNICA din Bucureşti
Facultatea de electronică, Telecomunicații și Tehnologia Informației

PROIECT

PROGRAMAREA INTERFEȚELOR PENTRU BAZE DE DATE

Profesor coordonator: Ș.l. Dr. Ing. Pupezescu Valentin

Student: Mușuroi Niky-Sebastian

Grupa: 433E

CUPRINS

1. Tema proiectului

Tema proiectului se bazează pe dezvoltarea unei aplicații ce conține o bază de date, creată în sistemul de gestionare a bazelor de date MySQL. Se pot utiliza diferite tehnologii: JSP, Hibernate, JPA, .NET, Python+Django, Python+Flask, etc.

Interfețele vor trebui să permită utilizatorului să execute următoarele operații pe toate tabelele: vizualizare, adăugare, modificare și ștergere date. Vizualizarea tabelelor de legătură va presupune vizualizarea datelor referite din celelalte tabele.

Pentru tema individuală primită, am ales tehnologia JSP, iar asocierea pentru tabelele din baza de date este de M:N.

2. Descrierea sistemului de gestiune a bazelor de date MySQL.

2.1. Ce este MySQL?

MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Este cel mai popular SGBD open-source la ora actuală[5], fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).[1]

2.2. Cum administrează MySQL baza de date?

Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă sau, prin descărcare de pe internet, o interfață grafică: MySQL Administrator și MySQL Query Browser. Un alt instrument de management al acestor baze de date este aplicația gratuită, scrisă în PHP, phpMyAdmin.[1]

2.3. Unde este utilizat MySQL?

Baza de date MySQL este folosită în principal ca mijloc de a stoca date pentru aplicații mari, bazate pe web. Site-uri precum WordPress, iStock, GitHub, Facebook, NASA, Marina SUA, Tesla, Scholastic, Spotify, YouTube, Netflix, Glasses Direct, Symantec (și multe altele) folosesc baza de date MySQL ca mijloc de stocare a datelor pe din interiorul sau exteriorul site-urilor web și serviciilor interne. Ce înseamnă asta pentru un utilizator mediu? Înseamnă că utilizați indirect MySQL în fiecare zi.[2]

3. Tehnologia JSP

3.1. Ce este JSP?

Este un acronim pentru paginile serverului Java. Este o tehnologie din partea serverului care ajută dezvoltatorii sau utilizatorii să genereze pagini web. Este o caracteristică de îmbunătățire a Servlets, dezvoltată de Sun Microsystems. Aceasta a fost dezvoltată pentru a acoperi toate defectele Servlets. Servletele utilizate pentru a conține o logică combinată a întreprinderilor și a interfeței cu utilizatorul. În JSP, logica de prezentare și logica de afaceri sunt separate. Așa cum am scris sau folosit anterior, etichetele sunt utilizate pentru a defini o anumită acțiune. În JSP, definim eticheta specială drept „”. Când folosim JSP, putem implementa, în mod implicit, API-ul Java în programarea web. Nu numai API-ul Java, ci JSP funcționează cu etichete HTML și XML, ceea ce anterior nu a fost posibil.[3]

3.2. Ce este un SERVLET? / Apache Tomcat

Un servlet este o clasă scrisă în limbajul Java al cărei scop este generarea dinamică de date într-un server HTTP. O astfel de clasă poate crea

atât conținut HTML, cât și documente XML, imagini, alte fișiere etc. În cea mai mare parte servleții sunt folosiți împreună cu protocolul HTTP, deși există posibilitatea de a utiliza servleți generici care nu sunt legați de un anumit protocol. Așadar prin „servlet” se înțelege de obicei „servlet HTTP”.[4]

Apache Tomcat (sau simplu Tomcat, în trecut denumit Jakarta Tomcat) este un web server open source și container servlet dezvoltat de Apache Software Foundation (ASF).[6]

4. Limbajul HTML și framework-ul Bootstrap

4.1. Ce este HTML?

HyperText Markup Language (HTML) este un limbaj de marcare utilizat pentru crearea paginilor web ce pot fi afișate într-un browser (sau navigator). Scopul HTML este mai degrabă prezentarea informațiilor – paragrafe, fonturi, tabele ș.a.m.d. – decât descrierea semanticii documentului. În cadrul dezvoltării web de tip front-end, HTML este utilizat împreună cu CSS și JavaScript. [7]

4.2. Ce este Bootstrap?

Bootstrap este o colecție de instrumente utile pentru crearea designului (grafică, animație și interacțiuni cu pagina din browser etc.) a site-urilor web și a aplicațiilor web . Este un pachet care conține coduri HTML și CSS, formulare, butoane, instrumente de navigare și alte elemente interactive, precum și extensii JavaScript opționale. Este unul dintre cele mai populare proiecte de pe platforma de management al dezvoltării GitHub .[8]

5. Descrierea aplicației

5.1. Baza de date

Tema se bazează pe ccrearea unei baze de date ce are 2 tabele în asociere M:N. Tabelele sunt: **cities** și **actors**, pentru aceste 2 tabele avem a treia tabelă **theaters** ce face legătura între tabelele de mai sus.

Ce este asocierea M:N?

Asocierea M:N (mai-mulți-la-mai-mulți) are ca si caracteristica faptul ca fiecărui element înregistrat într-o tabela îi pot fi asociate mai multe elemente din cealaltă tabela si invers. [9]

De exemplu, în cazul nostru, un actor poate fi asociat mai multor orașe, așa cum si unui oraș ii pot fi asociați mai mulți actori.

Tabela **cities**: atributul *idcity* este cheia primară(PK), urmat de celelalte atribute: *cityname*, *country*, *region*.

Tabela **actors**: atributul *idactor* este cheia primară(PK), urmat de celelalte atribute: *name*, *surname*, *age, gender*.

Tabela **theaters**: atributul *idtheater* este cheia primară(PK), urmat de celelalte atribute: *idcity, idactor, theatername*.

Ultima tabelă este cea de legătură, aceasta este necesară pentru a impărți o relație mai-mulți-la-mai-mulți în două relații unu-la-mai-mulți. În această tabelă, atributele ce au fost selectate ca și chei primare pentru tabelele anterioare vor deveni chei străine(FK).

Tabela **users**: atributul *id* este cheia primară(PK), urmat de celelalte atribute: *name, email, password, securityQuestion, answer*. Necesară pentru sistemul de autentificare al aplicației.

5.2. Diagrama logică a bazei de date (Diagrama ERD)

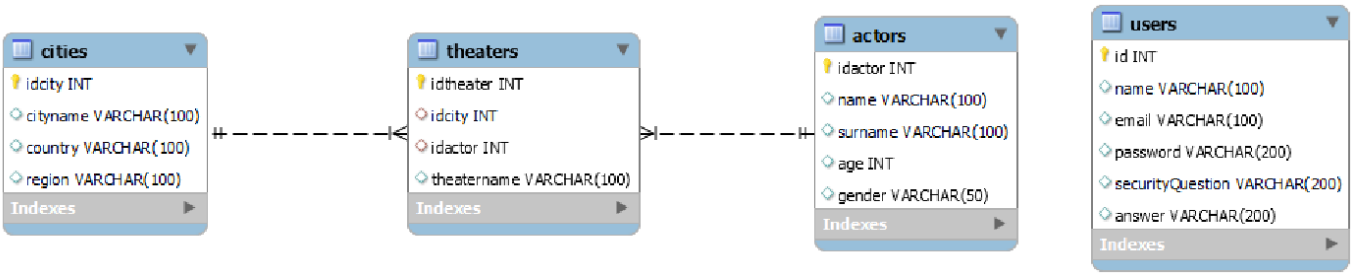


Diagrama oferă o reprezentare logică, detaliată a celor 4 tabele și relațiile dintre ele.

Relațiile dintre tabele:

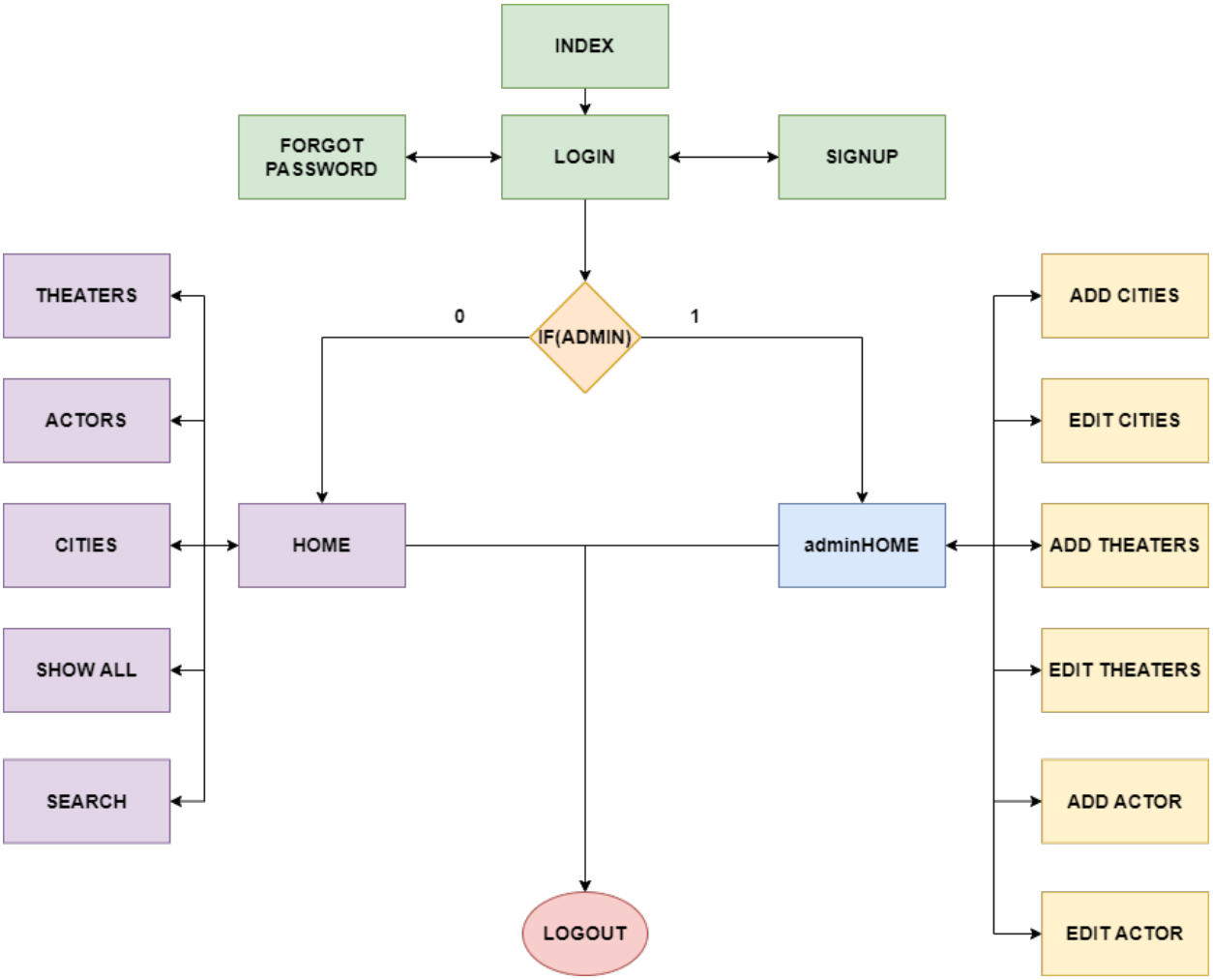
users este o tabelă **independentă**.

Între **cities** și **theaters** este o asociere **1:N**.

Între **cities** și **actors** este o asociere **M:N**.

Între **actors** și **theaters** este o asociere **1:N**.

5.3. Diagrama fișierelor ce alcătuiesc website-ul



5.4. Funcționalitatea aplicației

5.4.1. Arhitectura proiectului

Folder-ul **main/webapp**: conține **fișierele .jsp** și celelalte foldere pentru o strucutră curată și mai ușor de înțeles.

Pentru partea administrativă avem următoarele sub-foldere: **admin** și **table**, care la rândul lor conțin **fișiere .jsp**.

Folder-ul **main/java/project**: conține fișierul **ConnectionProvider.java** ce realizează conexiunea proiectului la **baza de date MySQL**.

Folder-ul **main/table**: conține fișierul **create_tables.jsp**, care creează *tabelele necesare* proiectului în *baza de date MySQL*.

5.4.2. Partea de BACKEND a proiectului

Conectarea la baza de date se face prin fișierul ConnectionProvider.java cu ajutorul JDBC(Java Database Connectivity).

JDBC este Java API-ul care gestionează conectarea la baza de date MySQL, emiterea de interogări și comenzi și manipularea rezultatelor obținute de la baza de date.

```
1  package project;
2  import java.sql.*;
3
4  public class ConnectionProvider{
5      public static Connection getCon(){
6          try {
7              Class.forName("com.mysql.cj.jdbc.Driver");
8              Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/theaterp","root","[redacted]");
9              return con;
10         }
11         catch(Exception e){
12             System.out.print(e);
13             return null;
14         }
15     }
16 }
```

Se declară clasa ConnectionProvider, ce conține funcția getCon(), care returnează obiectul con, obiect ce conține conectarea la baza de date MySQL.

După realizarea conexiunii la baza de date, se creează tabelele necesare proiectului prin intermediul fișierului create_tables.jsp

Se importă clasa ConnectionProvider și toate funcțiile din clasa java.sql.

```
<%@page import="project.ConnectionProvider" %>
<%@page import="java.sql.*" %>
```

**Aceste linii de cod sunt necesare tuturor fișierelor .jsp care necesită informații din baza de date.*

Se face conexiunea la baza de date prin variabila con de tip Connection.

```
Connection con = ConnectionProvider.getCon();
```

Se realizează interfața pentru a trimite interogări bazei de date.

```
Statement st= con.createStatement();
```

Se declară interogarea pentru crearea unui tabel și se execută.

```
String query1 = "CREATE TABLE users(id int AUTO_INCREMENT PRIMARY KEY, name varchar(100),
```

```
System.out.print(query1);
st.execute(query1);
```

Se afișează mesajul pentru confirmarea creării tabelului, se închide conexiunea la baza de date.

```
System.out.print("Tables created.");
con.close();
```

În cazul în care exista erori acestea vor fi afișate în consolă.

```
catch(Exception e)
{
    System.out.print(e);
}
```

Asemenea și pentru celelalte tabele(theaters, cities, actors**) necesare bazei de date.*

Afișarea rezultatelor în paginile website-ului

După conectarea la baza de date, se execută o interogare prin intermediul variabilei rs de tip ResultSet.

```
try{
    Connection con = ConnectionProvider.getCon();
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT theaters.theatername, cities.cityname, cities.country, cities.region,
```

Parcurgem elementele din interogare până când ajungem la finalul lor.

```
while(rs.next()){
```

Interogarea se realizează asupra tabelelor theaters, cities și actors prin joncțiunea celor 3 tabele folosind **INNER JOIN ON**.

```
"SELECT theaters.theatername, cities.cityname, cities.country, cities.region, actors.name, actors.surname, actors.age, actors.gender FROM
```

```
((theaters INNER JOIN cities ON theaters.idcity = cities.idcity) INNER JOIN actors ON theaters.idactor = actors.idactor)  "
```

Pentru funcția de **„search”** a website-ului se folosește același principiu ca cel de sus, urmat de condiția WHERE, LIKE și OR.

```
WHERE c.cityname LIKE '%"+search+"%' OR c.region LIKE '%"+search+"%' OR c.country LIKE '%"+search+"%' OR
```

În plus, avem nevoie de o variabilă inițializată cu 0. `int z = 0;`

Scopul acestei variabile este de a marca dacă există elementul cautat în baza de date. În cazul în care există variabila devine 1.

```
while(rs.next()){
    z = 1;
```

Dacă nu există afișam mesajul corespunzător în pagna web.

```
<%if (z == 0){%>

<h3 class="text-center text-white">Nothing. Try to search again.</h3><br><br><br>

<%} %>
```

Se afișează elementele necesare din baza de date prin intermediul variabilei rs și a funcției .getString(index).

```
<th>THEATER NAME</th>     <td><%=rs.getString(1) %></td>
<th>CITY NAME</th>        <td><%=rs.getString(2) %></td>
<th>COUNTRY</th>          <td><%=rs.getString(3) %></td>
<th>REGION</th>           <td><%=rs.getString(4) %></td>
<th>ACTOR NAME</th>       <td><%=rs.getString(5) %></td>
<th>ACTOR SURNAME</th>    <td><%=rs.getString(6) %></td>
<th>AGE</th>              <td><%=rs.getString(7) %></td>
<th>GENDER</th>           <td><%=rs.getString(8) %></td>
```

În cazul în care apar erori aceste vor fi afișate în consolă.

```
catch(Exception e){
    System.out.println(e);
}
```

Adăugarea și actualizarea rezultatelor în baza de date

Se realizează prin metoda „post”, după apăsarea butonului de SUBMIT sau SAVE se face legătura cu fișierul(..Action) ce realizează modificări asupra bazei de date.

```
<form action="signupAction.jsp" method="post">
```

Această metodă va transmite intrările în baza de date, astfel: se declară o variabilă de tip String căreia i se atribuie prin request.getParameter(„parametrul”) parametrul de introdus sau actualizat în baza de date.

```
String name = request.getParameter("name");
```

Apoi prin conexiunea la baza de date se declară o variabilă ps de tip PreparedStatement căreia i se atribuie interogarea de INSERT sau UPDATE pentru baza de date.

```
Connection con = ConnectionProvider.getCon();
PreparedStatement ps = con.prepareStatement("INSERT INTO users VALUES(?,?,?,?,?,?)");
```

Pentru update folosim o variabilă st de tip Statement și apoi trimitem cererea către baza de date prin st.executeUpdate(„query”).

```
Statement st = con.createStatement();

st.executeUpdate("UPDATE actors SET name='"+name+"', surname='"+surname+"', age='"+age+"',
```

Dacă totul decurge bine vom transmite un mesaj pntru a fi afișat pe pagina web.

```
response.sendRedirect("editActors.jsp?msg=done");
```

În cazul unei erori:

```
catch(Exception e){
    System.out.println(e);
    response.sendRedirect("editActors.jsp?msg=wrong");
```

Pentru afișarea mesajului:

```
<%
    String msg = request.getParameter("msg");
    if("done".equals(msg)){
%>

<h1 class="text-center text-success">ACTOR added successfully to database.</h1>

<%} %>

<%if("wrong".equals(msg)){ %>
<h1 class="text-center text-warning">Something went wrong! Try again later!</h1>
<%} %>
```

Ștergerea datelor din baza de date

Se realizează prin acționarea butonuli DELETE din pagina web.

```
<td><a href="deleteActorsByID.jsp?idactor=<%=rs.getString(1) %>" class="btn btn-danger">DELETE</a></td>
```

Acesta acționează fișierul necesar pentru ștergerea din baza de date, transmițând id-ul elementului care trebuie șters.

```
String id = request.getParameter("idactor");
```

Se utilizează o variabilă st de tip Statement, apoi se execută actualizarea tabelului prin ștergerea elementului dorit.

```
Statement st = con.createStatement();

st.executeUpdate("DELETE FROM actors WHERE idactor='"+id+"'");
```

Se redirecționează către pagina inițiala și se tranmsit mesajele corespunzătoare succesului sau eșecului.

```
    response.sendRedirect("editActors.jsp?=removed");
}

catch(Exception e){
    response.sendRedirect("editActors.jsp?=wrong");
    System.out.println(e);
```

5.4.3. Partea de FRONTEND a proiectului

Această parte se realizează prin limbajul de programare **HTML** alături de framework-ul **Bootstrap**.

Datorită claselor speciale ale framework-ului **Bootstrap** pentru elementele din **HTML** s-a realizat aspectul website-ului fără limbajul de programare **CSS**, util pentru stilizare.

Imaginea de background a website-ului a fost adăugată în fișierele .jsp care necesitau această imagine, astfel:

```
<div class="row">
    <!-- Background image -->
    <div class="bg-image" style="background-image: url('https://imgtr.ee/images/2023/01/28/G5Wki.jpg'); height: 100vh; background-position: center;
        background-repeat: no-repeat;
        background-size: cover;">
    </div>
    <!-- Background image -->
</div>
```

** ** : Împarte pagina în elemente.

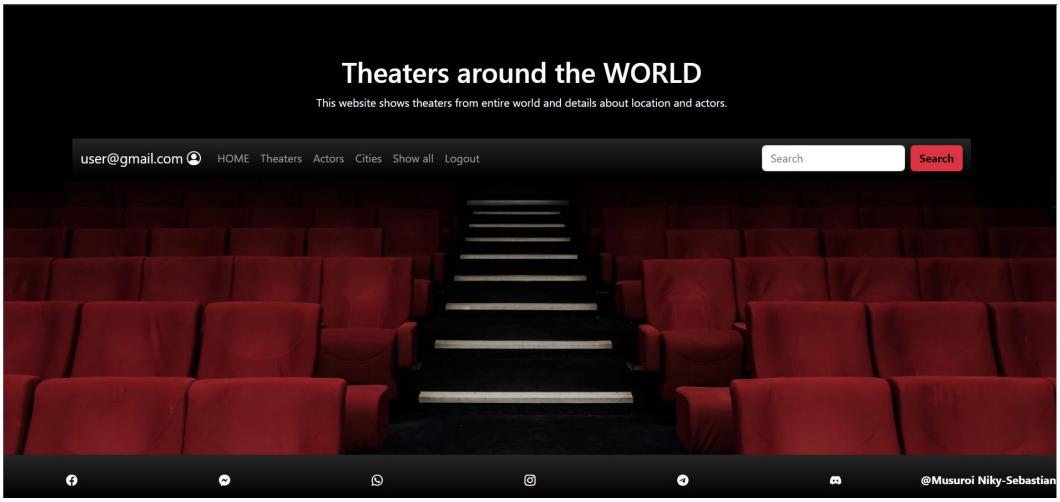
class=”row” : Asociază elementului clasa row.

Stilizarea butoanelor folosite:

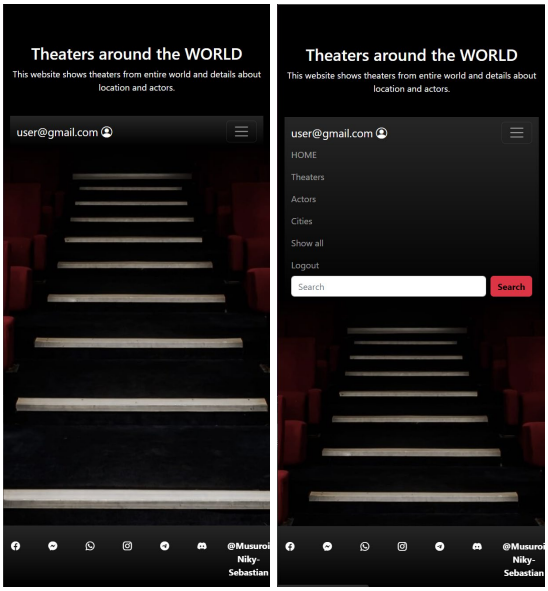
```
class="btn btn-danger text-black fw-bold"
```

Tot conținutul afișat pe pagina web este adaptabil pentru orice rezoluție de ecran, lucru posibil datorită framework-ului Bootstrap.

Pagina HOME ecran laptop:



Pagina HOME ecran smartphone:



CONCLUZII

Având în vedere importnața utilizării bazelor de date, consider acest proiect foarte util pentru înțelegerea mai amânunțită a bazelor de date și cum se gestionează acestea.

Pentru o utilizare corectă și fiabilă a unei baze de date este important să se urmărească realizarea unei arhitecturi ce oferă posibilitatea de a separa funcționalitățile, partea de backend de partea de frontend.

În prezent bazele sunt utilizare peste tot, fiecare companie cu caracter de producete, comercializare are nevoie să implementeze în sistemul lor o bază de date.

BIBLIOGRAFIE

[1] https://ro.wikipedia.org/wiki/MySQL

[2] https://www.nav.ro/blog/ce-este-mysql/

[3] https://ro.education-wiki.com/8106597-what-is-jsp

[4] https://ro.wikipedia.org/wiki/Servlet

[5] http://db-engines.com/en/ranking

[6] https://ro.wikipedia.org/wiki/Apache_Tomcat

[7] https://ro.wikipedia.org/wiki/HyperText_Markup_Language

[8] https://ro.frwiki.wiki/wiki/Bootstrap_(framework)

[9] Cursuri PIBD