Universitatea Politehnica din Bucuresti

Facultatea de Electronica,Telecomunicatii si Tehnologia Informatiei

Programarea Interfetelor Pentru Baze De Date

Tehnologie Hibernate

Elisei Stefan-Sergiu

Grupa-434C

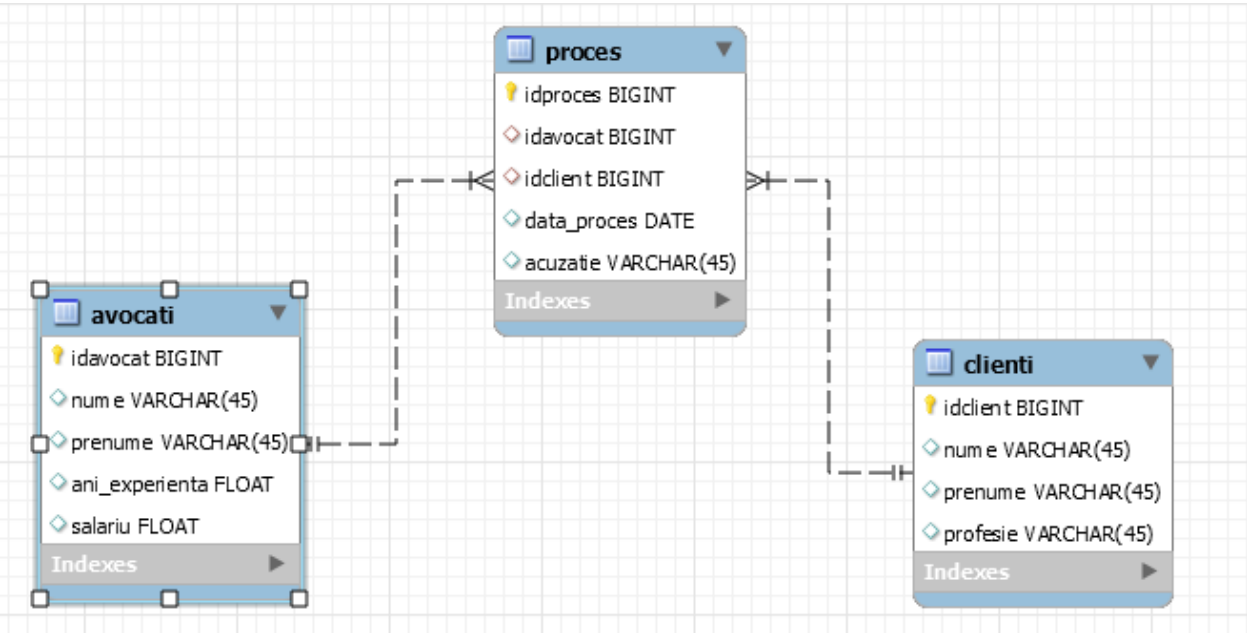
Cuprins

Baza de date

Tehnologia utilizata pentru realizarea bazei de date este MySQL.

MySQL este un sistem de gestionare a bazelor de date relaționale open source care este utilizat în principal pentru aplicațiile online. MySQL poate crea și gestiona baze de date foarte utile (cum ar fi informații despre angajați, inventar și multe altele).

Datele găzduite în structură sunt capabile să recunoască relațiile dintre informațiile stocate. Fiecare bază de date conține tabele. Fiecare tabel (denumit și o relație) conține una sau mai multe categorii de date stocate în coloane (denumite și atribute). Fiecare rând (denumit, de asemenea, o înregistrare sau „tuple”) conține o informație unică (altfel menționată ca și cheie) pentru categoriile definite în coloane.[1]



Baza de date pe care am realizat-o contine 3 tabele:

- avocati
- clienti
- proces

Asocierea intre tabelele avocati si clienti este de tipul M:N. Astfel,s-a introdus o noua tabela (“proces”) ,cu ajutorul careia s-a simplificat asocierea la avocati M:1 proces 1:N clienti.

Atributele celor 3 tabele sunt:

1.avocati

- idavocat,tip BIGINT,primary key
- nume, tip VARCHAR(45)
- prenume, tip VARCHAR(45)
- ani_experienta,tip FLOAT
- salariu,tip FLOAT

2.clienti

- idclient,tip BIGINT,primary key
- nume, tip VARCHAR(45)
- prenume, tip VARCHAR(45)
- profesie, tip VARCHAR(45)

3.proces

- idproces,tip BIGINT,primary key
- idavocat,tip BIGINT,foreign key
- idclient, tip BIGINT,foreign key
- data_proces,tip DATE
- acuzatie,tip VARCHAR(45)

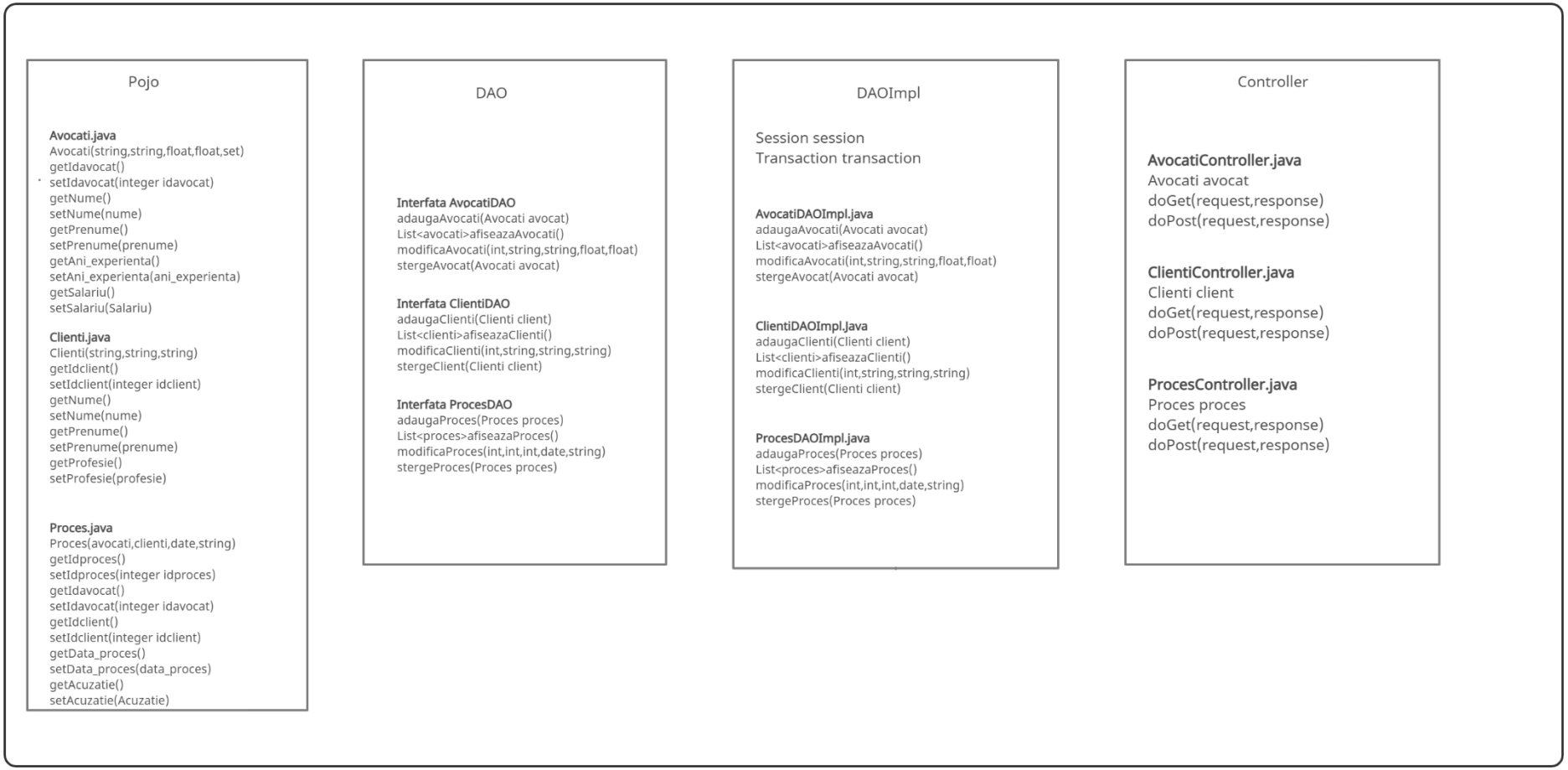
Tehnologia utilizata:Hibernate

Framework-ul Hibernate reprezintă un serviciu de mapare obiect-relaţional pentru limbajul de programare Java. Prin intermediu acestei platforme se asigură persistenţa datelor pentru aplicatiile web

Hibernate ORM permite utilizatorilor să dezvolte aplicaţii ale căror date vor fi disponibile şi după ce aplicaţia este oprită. Hibernate ORM se ocupă cu persistenţa datelor stocate în sisteme de gestiune a bazelor de date relaţionale. ORM(Object-Relational Mapping) reprezintă o tehnică de programare de conversie a datelor stocate în sisteme de gestiune a bazelor de date relaţionale şi limbaje de programare obiect-orientate precum Java. [2]

Arhitectura Hibernate este structurată pe mai multe niveluri, astfel încât să fie încapsulate cât mai multe dintre interfețele de programare pe care le utilizează, oferind aplicației servicii legate de obiectele persistente. [3]

Diagrama UML



Pojo-Plain Oriented Java Objects

Folderul pojo realizeaza maparea bazei de date. Astfel se trece de la tabelele avocati, clienti, proces la clasele avocati, clienti, proces.

In interiorul acestor clase, se gasesc Setters si Getters, precum si Constructori.

```
public Avocati(String nume, String prenume, Float ani_experienta,Float salariu, Set proceses) {
    this.nume = nume;
    this.prenume = prenume;
    this.ani_experienta=ani_experienta;
    this.salariu=salariu;
    this.proceses = proceses;
}

public Integer getIdavocat() {
    return idavocat;
}

public void setIdavocat(Integer idavocat) {
    this.idavocat = idavocat;
}

public String getNume() {
    return nume;
}

public void setNume(String nume) {
    this.nume = nume;
}

public String getPrenume() {
    return prenume;
}

public void setPrenume(String prenume) {
    this.prenume = prenume;
}
```

Fisierele Avocati.hbm.xml,Clienti.hbm.xml,Proces.hbm.xml asigura maparea corecta a bazei de date.

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="pojo.Avocati" table="avocati" catalog="tema_pibd"
        optimistic-lock="version">
        <id name="idavocat" type="java.lang.Integer">
            <column name="idavocat" />
            <generator class="identity" />
        </id>
        <property name="nume" type="string">
            <column name="nume" length="45" />
        </property>
        <property name="prenume" type="string">
            <column name="prenume" length="45" />
        </property>
        <property name="ani_experienta" type="float">
            <column name="ani_experienta" length="45" />
        </property>
        <property name="salariu" type="float">
            <column name="salariu" length="45" />
        </property>
        <set name="proceses" table="proces" inverse="true"
            lazy="true" fetch="select">
            <key>
                <column name="idavocat" />
            </key>
            <one-to-many class="pojo.Proces" />
        </set>
    </class>
</hibernate-mapping>
```

Fișierul hibernate.cfg.xml realizează configurarea internă a proiectului, iar fișierul hibernate.reveng.xml (reverse engineering) realizează echivalarea tipurilor de date din baza de date cu tipuri Hibernate.

DAO

In acest folder au fost create interfete in care s-au definit functiile ce vor fi implementate in DAOImpl.

Acest folder a fost creat pentru o mai buna sistematizare a codului.

```
 * To change this license header, choose License Headers in Project Properties.
package DAO;

import java.util.List;

/**
 *
 * @author vali
 */
public interface AvocatiDao {
    public void adaugaAvocati (Avocati avocat);
    public List<Avocati> afiseazaAvocati();
    public void modificaAvocati (int idavocat, String nume, String prenume, Float ani_experienta,Float salariu);
    public void stergeAvocat (Avocati avocat);
}
```

DAOImpl

In folderul DAOImpl are loc implementarea functiilor definite in cadrul DAO.

Astfel cu ajutorul acestor functii se realizeaza operatiile de Vizualizare,Adaugare,Modificare,Stergere de date.

Toate operatiile se realizeaza in cadrul unei tranzactii

Transaction transaction = session.beginTransaction();

La randul ei tranzactia are loc in cadrul unei sesiuni care trebuie initiata.

Session session = HibernateUtil.getSessionFactory().openSession();

```
 */
public class AvocatiDaoImpl implements AvocatiDao{

    public void adaugaAvocati(Avocati avocat) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction transaction = session.beginTransaction();
        session.save(avocat);
        transaction.commit();
        session.close();
    }

    public List<Avocati> afiseazaAvocati() {
        List<Avocati> listaAvocati = new ArrayList();
        Session session = HibernateUtil.getSessionFactory().openSession();
        org.hibernate.Query query = session.createQuery("From Avocati");
        listaAvocati = query.list();
        return listaAvocati;
    }
```

Controller

In folderul Controller se gasesc fisiere de tip Servlet.

Clientul transmite din interfata in mod dinamic cereri catre Servlet, iar Servletul trimite ,la fel dinamic, un raspuns catre client.

Totodata, aici se apeleaza tranzactiile din “DAOImpl”, iar legatura dintre interfata si cod se face prin crearea unor metode de tipul doGet() si doPost().

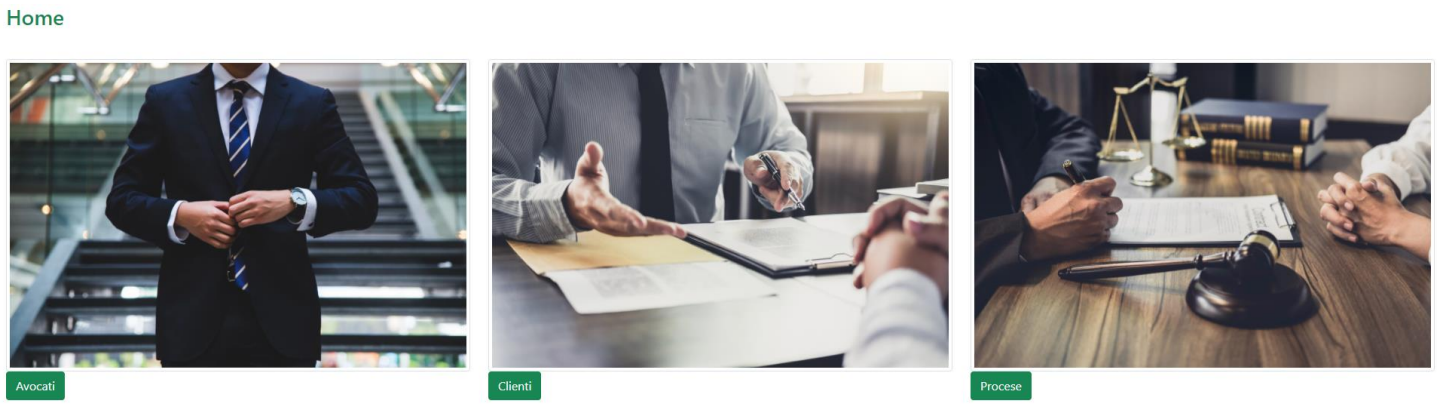
```
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        if (request.getParameter("adaugaAvocat") != null) {
            String nume = request.getParameter("nume");
            String prenume = request.getParameter("prenume");
            String ani_experienta2 = request.getParameter("ani_experienta");
            Float ani_experienta=Float.parseFloat(ani_experienta2);
            String salariu2 = request.getParameter("salariu");
            Float salariu=Float.parseFloat(salariu2);

            avocat.setNume(nume);
            avocat.setPrenume(prenume);
            avocat.setAni_experienta(ani_experienta);
            avocat.setSalariu(salariu);
            avocatDaoImpl.adaugaAvocati(avocat);
            RequestDispatcher rd = request.getRequestDispatcher("date_adaugate_avocati.jsp");
            rd.forward(request, response);
        }
    }
```

Interfata Web

Pagina principala a aplicatiei este implementata cu ajutorul fisierului index2.html.

Designul l-am realizat cu ajutorul site-ului Layoutit.com[4]

Este accesat urmatorul url :
http://localhost:8080/tema_pibd_2/index2.html




```
<div class="row">

    <div class="col-md-4">
        <img alt="Bootstrap Image Preview" src="https://englishlawyersjapan.com/wp-content/uploads/2019/09/corporate-law.jpg" class="img-thumbnail">
       <form action="AvocatiController" method="POST">
            <br><input class="btn btn-success" type="submit" name="afiseazaAvocati" value="Avocati">    <br>
        </form>

    </div>
    <div class="col-md-4">
        <img alt="Bootstrap Image Preview" src="https://scanners.us.fujitsu.com/-/media/project/scanners/blog/2019/08/istock-941906652.jpg?mw=1440&w=1440&hash=A01A16600BBF82E7
       <form action="ClientiController" method="POST">
            <br><input class="btn btn-success" type="submit" name="afiseazaClienti" value="Clienti">    <br>
        </form>
    </div>
    <div class="col-md-4">
        <img alt="Bootstrap Image Preview" src="https://www.criminaldefencelawyers.com.au/wp-content/uploads/2020/02/Criminal-lawyer-in-consultation-with-client-1440x960.jpg"
       <form action="ProcesController" method="POST">
            <br><input class="btn btn-success" type="submit" name="afiseazaProces" value="Procese">    <br>
        </form>
    </div>
</div>
<p></p>
```

Apasand pe unul dintre butoanele corespunzatoare tabelelor din baza de date, se vor vizualiza pe ecran datele existente in tabele.De exemplu,la apasarea butonului “Avocati”, se trimite request-ul “afiseazaAvocati”.Astfel se va apela functia afiseazaAvocati din AvocatiController.

```
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        if (request.getParameter("afiseazaAvocati") != null) {
            List<Avocati> listaAvocati = new ArrayList();
            listaAvocati = avocatDaoImpl.afiseazaAvocati();
            request.setAttribute("listaAvocati", listaAvocati);
            RequestDispatcher rd = request.getRequestDispatcher("tabela_Avocati.jsp");
            rd.forward(request, response);
        }
```

Dupa trimiterea raspunsului, se intra in fisierul tabela_Avocati.jsp, unde se afiseaza tabela avocati

```
<div class="row">
    <div class="col-md-12">
        <table class="table table-sm table-hover table-striped">
            <thead>
                <tr>

                    <th>
                        idavocat
                    </th>
                    <th>
                        Nume
                    </th>
                    <th>
                        Prenume
                    </th>
                    <th>
                        ani_experienta
                    </th>
                    <th>
                        salariu
                    </th>
                </tr>
            </thead>
        <c:forEach var="avocati" items="${listaAvocati}">
            <tr>
                <td>${avocati.idavocat}</td>
                <td>${avocati.nume}</td>
                <td>${avocati.prenume}</td>
                <td>${avocati.ani_experienta}</td>
                 <td>${avocati.salariu}</td>
            </tr>
        </c:forEach>
    </table>
```

Tabela Avocati

Home

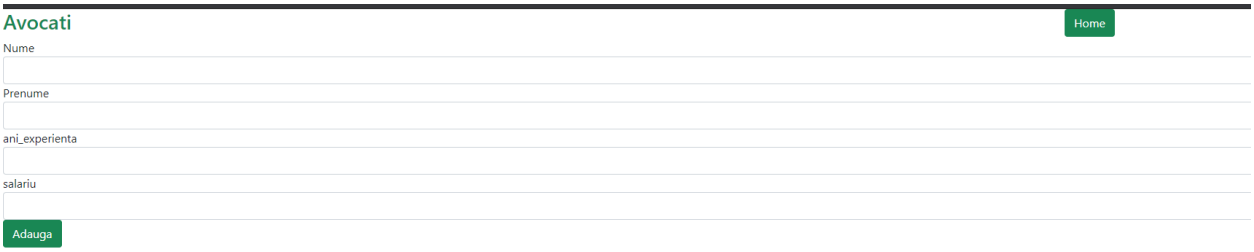
Adauga avocat nou

idavocat	Nume	Prenume	ani_experienta	salariu
39	Elisei	Stefan	10.0	9000.0
40	Popescul	Valentin	7.0	1999.0
41	Busila	Leonora	6.0	2800.0

Pe ecran apar afisate toate atributele tabelei, precum si operatia de adaugare de date noi si de intoarcere la pagina principala.

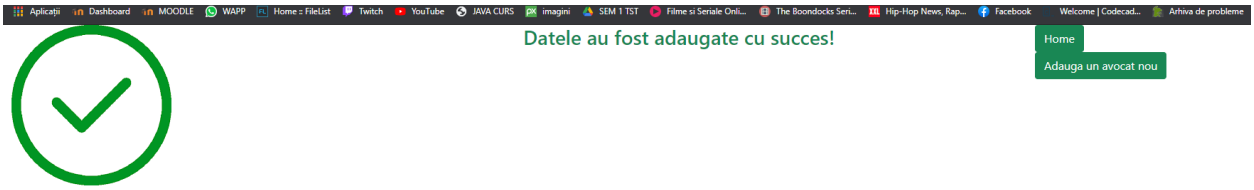
Operatia de adaugare de date noi se realizeaza prin fisierul adauga_Avocat.jsp, in care se cere introducerea variabilelor nume,prenume,ani_experienta,salariu,variabile ce vor fi folosite in functia adaugaAvocat, apelata in AvocatiController.

```
<div id="add">
    <form action="AvocatiController" method="GET">
    <div class="form-group">
        <table>
         <div class="form-group">

                        <label for="Nume">
            Nume
        </label>
        <input type="text" class="form-control" name="nume" />
                        <label for="Prenume">
            Prenume
        </label>
        <input  type="text" class="form-control" name="prenume" />
                        <label for="ani_experienta">
            ani_experienta
        </label>
        <input  type="text" class="form-control" name="ani_experienta" />
                        <label for="salariu">
            salariu
        </label>
        <input  type="text" class="form-control" name="salariu" />


    </div>
        <td><input type="submit" name="adaugaAvocat" value="Adauga" class="btn btn-success" ></td>

        </tr>
    </table>
    </div>
```



La adaugarea de noi date in tabela, apar campuri care trebuie completate cu datele noi pe care dorim sa le introducem.

Odata introduse datele, se apasa butonul de adauga si se primeste un mesaj de confirmare a datelor introduse.



Pentru operatiile de **modificare** si **stergere**, am creat 2 butoane, pe care odata la apasarea unuia dintre ele, utilizatorul este interogat pe care tabela doreste realizarea operatiei.

Voi continua exemplificarea cu cazul tabelei Avocati

Pentru **modificare** se alege tabela pe care se doreste modificarea(in cazul nostrum avocati), dupa care se apeleaza functia afiseazaAvocati_modificari in AvocatiController.

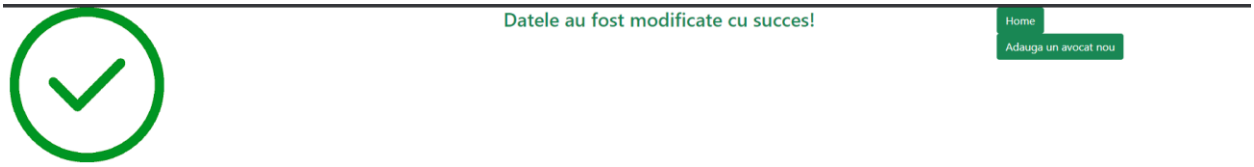
```
<div class="row">

        <div class="col-md-4">
           <form action="AvocatiController" method="POST">
                 <br><input class="btn btn-success" type="submit" name="afiseazaAvocati_modificari" value="Avocati">    <br>
            </form>

        </div>

        <div class="col-md-4">
           <form action="ClientiController" method="POST">
                 <br><input class="btn btn-success" type="submit" name="afiseazaClienti_modificari" value="Clienti">    <br>
            </form>
        </div>
        <div class="col-md-4">

           <form action="ProcesController" method="POST">
                 <br><input class="btn btn-success" type="submit" name="afiseazaProcese_modificari" value="Procese">    <br>
            </form>
        </div>
    </div>
        </div>
    </div>
</div>
```

```
63          if (request.getParameter("afiseazaAvocati_modificari") != null) {
64              List<Avocati> listaAvocati = new ArrayList();
65              listaAvocati = avocatDaoImpl.afiseazaAvocati();
66              request.setAttribute("listaAvocati", listaAvocati);
67              RequestDispatcher rd = request.getRequestDispatcher("tabela_modificari_Avocat.jsp");
68              rd.forward(request, response);
69          }
70
71          if (request.getParameter("modificaAvocat") != null) {
72              int id1 = Integer.parseInt(request.getParameter("idavocat"));
73              String nume = request.getParameter("nume");
74              String prenume = request.getParameter("prenume");
75              String ani_experienta2 = request.getParameter("ani_experienta");
76              Float ani_experienta=Float.parseFloat(ani_experienta2);
77              String salariu2 = request.getParameter("salariu");
78              Float salariu=Float.parseFloat(salariu2);
79              avocatDaoImpl.modificaAvocati(id1, nume, prenume, ani_experienta,salariu);
80              RequestDispatcher rd = request.getRequestDispatcher("date_modificate_avocati.jsp");
81              rd.forward(request, response);
82
83          }
84
```

Astfel, pe ecran se va afisa tabela Avocati cu datele existente impreuna cu o casuta in care se va alege id-ul pentru care vrem sa modificam datele.

Se introduc datele noi, dupa care se apasa pe butonul **modifica**

Dupa indeplinirea cererii de modificare, se returneaza raspunsul din AvocatiController, si se afiseaza conform fisierului date_modificate_avocati.jsp, un mesaj de confirmare al modificarilor.



Pentru **stergere**, similar se alege tabela pe care se doreste modificarea(in cazul nostrum avocati), dupa care se trimite cererea de stergere in AvocatiController si se apeleaza functia afiseazaAvocati_stergere().

```
<body>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9
<div class="container-fluid">
    <div class="row">
        <div class="col-md-12">
            <h3 class="text-center text-success">
                Ce tabela doriti sa stergeti?
                                <img alt="Bootstrap Image Preview" src="https://images-eu.ssl-images-amazon.com/images/I/41oQhTiOPIL._SY300_QL70_ML2_.jpg" />
            </h3>
<div class="row">

        <div class="col-md-4">
           <form action="AvocatiController" method="POST">
                <br><input class="btn btn-success" type="submit" name="afiseazaAvocati_stergere" value="Avocati">    <br>
            </form>

        </div>

        <div class="col-md-4">
           <form action="ClientiController" method="POST">
                <br><input class="btn btn-success" type="submit" name="afiseazaClienti_stergere" value="Clienti">    <br>
            </form>
        </div>
        <div class="col-md-4">

           <form action="ProcesController" method="POST">
                <br><input class="btn btn-success" type="submit" name="afiseazaProces_stergere" value="Procese">    <br>
            </form>
        </div>
    </div>
        </div>
    </div>
</div>
</body>
```

```
if (request.getParameter("afiseazaAvocati_stergere") != null) {
    List<Avocati> listaAvocati = new ArrayList();
    listaAvocati = avocatDaoImpl.afiseazaAvocati();
    request.setAttribute("listaAvocati", listaAvocati);
    RequestDispatcher rd = request.getRequestDispatcher("tabela_stergere_Avocat.jsp")
    rd.forward(request, response);
}
```

Tabela Avocati

Home

Adauga avocat nou

idavocat	Nume	Prenume	ani_experienta	salariu
39	Elisei	Stefan	10.0	9000.0
40	Popescul	Valentin	7.0	1999.0
41	Busila	Leonora	6.0	2800.0
43	Elisei Stefan	tes2	231131.0	3.13112992E8

43

Sterge

Astfel, pe ecran se va afisa tabela Avocati cu datele existente impreuna cu o casuta in care se va alege id-ul pentru care vrem sa stergem datele.

Dupa alegerea id-ului se apasa pe butonul **sterge.**

Dupa indeplinirea cererii de stergere, se returneaza raspunsul din AvocatiController, si se afiseaza conform fisierului date_sterse_avocati.jsp, un mesaj de confirmare al stergerilor.

Datele au fost sterse cu succes!

Home

Adauga un avocat nou

Bibliografie

- https://www.nav.ro/blog/ce-este-mysql/
- https://app.slack.com/client/T01CD8NKE67/C01CD2KAHH8
- https://www.tutorialspoint.com/hibernate/hibernate_architecture.htm
- https://www.layoutit.com/build