Proiect realizat pentru disciplina
PROGRAMAREA INTERFEȚELOR PENTRU BAZE DE DATE

Hibernate

Student
Tucă Mihnea-Luca-Ioan
434B

Profesor coordonator
Ș. I. Dr. Ing. Pupezescu Valentin

Universitatea Națională de Știință și Tehnologie Politehnică București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
February, 2024

Contents

List of Figures

1 Introducere

1.1 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 si își propune să demonstreze flexibilitatea și eficiența tehnologiei Hibernate[2] prin implementarea operațiilor CRUD - Create, Read, Update, Delete - într-o aplicație web interactivă.

1.2 Obiective

Acest proiect își propune să exploreze diverse tehnologii utilizate în dezvoltarea aplicațiilor web, inclusiv Hibernate pentru persistența datelor[2], JSP pentru partea de server[3], și HTML/CSS pentru structura și designul frontend. Scopul este de a construi o aplicație web robustă care să demonstreze cum tehnologiile backend și frontend pot fi integrate eficient.

Gestionarea Muzicienilor

- Crearea de noi înregistrări pentru muzicieni.
- Vizualizarea detaliilor muzicienilor existenți.
- Actualizarea informațiilor muzicienilor.
- Ștergerea înregistrărilor muzicienilor inactivi sau duplicați.

Administrarea Stilurilor Muzicale

- Adăugarea de noi stiluri muzicale în sistem.
- Afișarea listei stilurilor muzicale, cu detalii precum origini și caracteristici.
- Modificarea informațiilor despre stilurile muzicale (de exemplu, actualizarea descrierii).
- Eliminarea stilurilor muzicale care nu mai sunt relevante.

Organizarea Concertelor

- Planificarea concertelor pentru diferiți muzicieni sau grupuri.
- Vizualizarea detaliată a concertelor, inclusiv locația, data și ora.
- Actualizarea detaliilor concertelor, cum ar fi schimbarea locației sau a datei.
- Anularea concertelor programate.

1.3 Aplicabilitate

Aplicația poate fi folosită pentru a gestiona carierele muzicienilor și evenimentele live. Tabela "Muzicieni" stochează informații despre artiști, în timp ce "Stiluri Muzicale" și "Concerte" sunt utilizate pentru a clasifica genurile muzicale și pentru a organiza evenimentele. Utilizatorii pot gestiona cu ușurință programările, promovarea stilurilor muzicale și coordonarea detaliilor concertelor, oferind o platformă centralizată pentru dinamica industriei muzicale.

2 Tehnologii utilizate

2.1 Sistemul de gestiune a bazelor de date MySQL

MySQL[1] este un sistem de gestiune a bazelor de date relaționale (SGBDR), open-source, care folosește Structured Query Language (SQL) pentru gestionarea datelor. Este ales pentru flexibilitatea, fiabilitatea și performanța sa, fiind ideal atât pentru aplicații mici, cât și pentru sisteme mari și complexe. MySQL excellează în aplicații web datorită integrării sale ușoare cu limbaje de programare precum Java, oferind o soluție robustă pentru dezvoltarea web dinamică.

2.2 JSP

Java Server Pages (JSP) este o tehnologie esențială pentru dezvoltarea aplicațiilor web, oferind un amestec perfect între HTML[8] și codul Java. Prin capacitatea sa de a genera conținut web dinamic și posibilitatea de integrare cu JavaBeans și alte tehnologii Java, cum ar fi JDBC, JSP devine o alegere ideală pentru crearea de aplicații web complexe și interacțiuni bazate pe date. Avantajele sale, cum ar fi portabilitatea, scalabilitatea și eficiența în gestionarea interfețelor utilizator dinamice, îl fac nu doar practic, ci și o opțiune populară printre dezvoltatorii care doresc să creeze experiențe web bogate și personalizate.

2.3 HTML

HTML (HyperText Markup Language)[8] este limba de bază pentru crearea și structurarea paginilor web. Folosind o serie de elemente și atribute, HTML permite dezvoltatorilor să definească structura de conținut, cum ar fi paragrafe, titluri, linkuri și imagini, pe o pagină web. Această structură semantică nu numai că este esențială pentru prezentarea vizuală și organizarea conținutului în browser, dar joacă un rol crucial în accesibilitatea și optimizarea motorului de căutare (SEO). Fiind strâns integrată cu CSS pentru stilizare și JavaScript pentru funcționalități, HTML stă la baza dezvoltării web, permițând crearea de site-uri web interactive și atractive.

2.4 Apache Tomcat

Apache Tomcat este un container de servlet-uri open-source implementat în Java, care permite rularea aplicațiilor web dezvoltate cu tehnologia Java Servlet și JavaServer Pages (JSP). Este unul dintre cele mai populare servere web pentru aplicații Java, datorită ușurinței de utilizare, flexibilității și performanței solide. Tomcat acționează ca un server web sau un mediul de execuție pentru aplicațiile Java, gestionând cererile HTTP de la clienți și facilitând executarea servlet-urilor și JSP-urilor pentru generarea răspunsurilor dinamice

3 Descrierea aplicației

3.1 Creearea bazei de date

Tema se bazează pe crearea unei baze de date ce are doua tabele în asociere M:N, "Muzicieni" și "StiluriMuzicale".

Table Name: muzicieni Schema: mihnea

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
idmuzician	BIGINT	☑	☑	☐	☐	☐	☐	☑	☐	
nume	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
prenume	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
datanasterii	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
taraorigine	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
instrumentprincipal	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 3.1: Tabela Muzicieni

Table Name: stiluri_muzicale Schema: mihnea

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
idstil	BIGINT	☑	☑	☐	☐	☐	☐	☑	☐	
numestil	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
originestil	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
perioadapopularitate	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 3.2: Tabela Stiluri Muzicale

Asocierea M:N (Many-to-Many) este o relație de bază de date între două tabele care permite fiecărei înregistrări din prima tabelă să fie asociate cu mai multe înregistrări din a doua tabelă și invers. Această relație necesită crearea unei tabele intermediare pentru a gestiona această asociere. În cazul nostru, am creat tabela de legătura "Concerte". Am ales ca și cheie primara idconcert. În această nouă tabelă, atributele ce au fost selectate ca și chei primare pentru tabelele anterioare, vor deveni chei străine (FK) pentru tabela de legătură.[1]

Table Name: concerte Schema: mihnea

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
idconcert	BIGINT	☑	☑	☐	☐	☐	☐	☑	☐	
idmuzician	BIGINT	☐	☑	☐	☐	☐	☐	☐	☐	
idstil	BIGINT	☐	☑	☐	☐	☐	☐	☐	☐	
dataconcert	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
locatie	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
altedetalii	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 3.3: Tabela Concerte

Diagrama EER (Enhanced Entity-Relationship) a unei baze de date este o reprezentare vizuală complexă care ilustrează entitățile, relațiile și constrângerile acestora, oferind o vedere detaliată și extinsă a structurii și interconexiunilor din cadrul bazei de date.

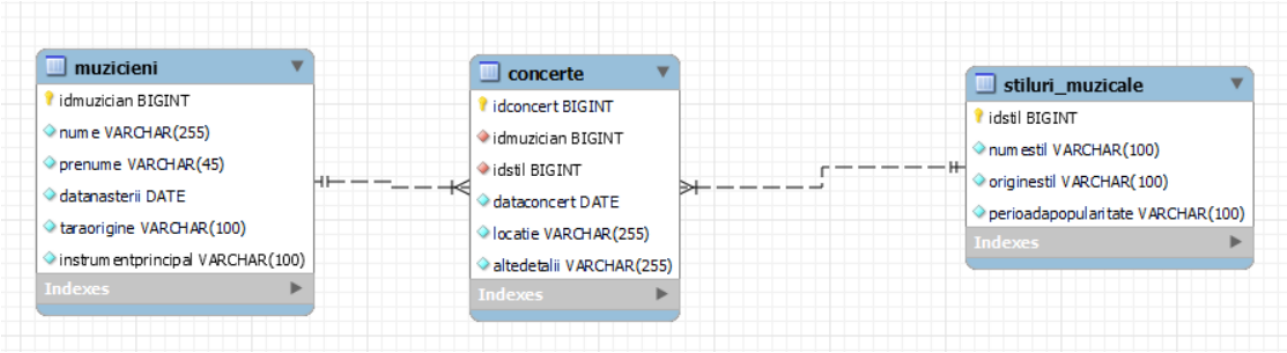

Figure 3.4: Diagrama EER (Enhanced Entity-Relationship)

3.2 Structura fisierelor

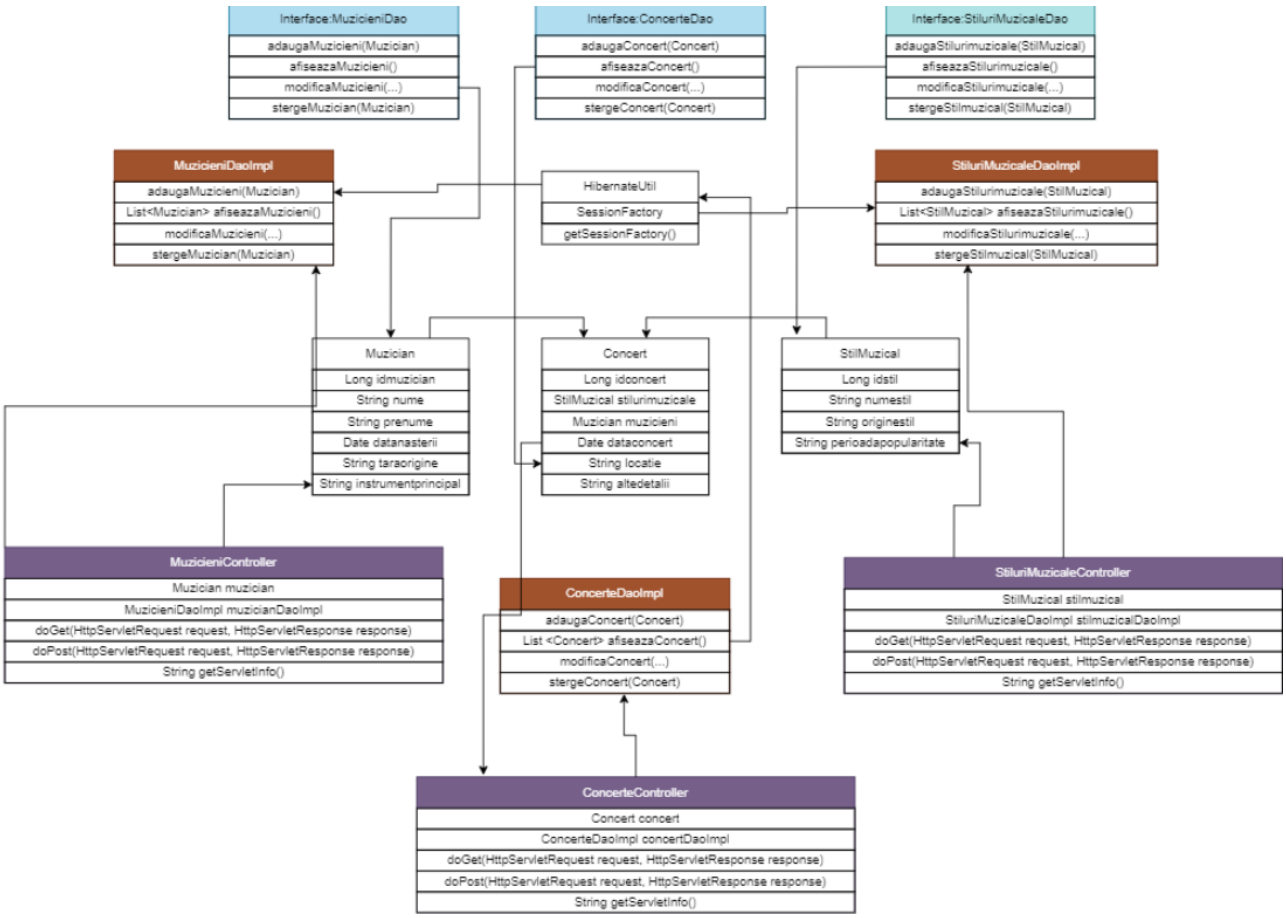

Figure 3.5: Diagrama implementării fișierelor ce alcătuiesc pagina web

3.3 Controller

Directorul Controller conține clasele care gestionează interacțiunile utilizatorului cu aplicația web. Acestea primesc cereri (requests) și trimit răspunsuri (responses).

- MuzicieniController.java: gestionează cererile legate de operațiunile cu muzicienii unui conert (afișarea, adăugarea sau actualizarea muzicienilor).
- StiluriMuzicaleController.java: gestionează cererile legate de stilurile muzicale.
- ConcerteController.java: gestionează cererile legate de concerte

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    if (request.getParameter("adaugaMuzicieni") != null) {
        String nume = request.getParameter("nume");
        String prenume = request.getParameter("prenume");
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
        Date datanasterii=null;
        try {
            datanasterii = df.parse(request.getParameter("datanasterii"));
        } catch (ParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        String taraorigine = request.getParameter("taraorigine");
        String instrumentprincipal = request.getParameter("instrumentprincipal");
        muzician.setNume(nume);
        muzician.setPrenume(prenume);
        muzician.setDatanasterii(datanasterii);
        muzician.setTaraorigine(taraorigine);
        muzician.setInstrumentprincipal(instrumentprincipal);
        muzicianDaoImpl.adaugaMuzicieni(muzician);
        List<Muzician> listaMuzicieni = muzicianDaoImpl.afiseazaMuzicieni();
        request.setAttribute("listaMuzicieni", listaMuzicieni);
        RequestDispatcher rd = request.getRequestDispatcher("tabela_Muzicieni.jsp");
        rd.forward(request, response);
    }
}
```

Figure 3.6: Functia adaugaMuzicieni() din Controller

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    if (request.getParameter("afiseazaMuzicieni") != null) {
        List<Muzician> listaMuzicieni = muzicianDaoImpl.afiseazaMuzicieni();
        request.setAttribute("listaMuzicieni", listaMuzicieni);
        RequestDispatcher rd = request.getRequestDispatcher("tabela_Muzicieni.jsp");
        rd.forward(request, response);
    }
```

Figure 3.7: Functia afiseazaMuzicieni() din Controller

```java
if (request.getParameter("modificaMuzicieni") != null) {
    Long id1 = Long.parseLong(request.getParameter("idmuzician"));
    String nume = request.getParameter("nume");
    String prenume = request.getParameter("prenume");
    Date datanasterii=null;
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
    try {
        datanasterii = df.parse(request.getParameter("datanasterii"));
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    String taraorigine = request.getParameter("taraorigine");
    String instrumentprincipal = request.getParameter("instrumentprincipal");
    muzicianDaoImpl.modificaMuzicieni(id1, nume, prenume, datanasterii, taraorigine, instrumentprincipal);
    List<Muzician> listaMuzicieni = muzicianDaoImpl.afiseazaMuzicieni();
    request.setAttribute("listaMuzicieni", listaMuzicieni);
    RequestDispatcher rd = request.getRequestDispatcher("tabela_Muzicieni.jsp");
    rd.forward(request, response);

}
```

Figure 3.8: Functia modificaMuzicieno() din controller

```java
if (request.getParameter("stergeMuzician") != null) {
    Long id2 = Long.parseLong(request.getParameter("idmuzician"));
    muzician.setIdmuzician(id2);
    muzicianDaoImpl.stergeMuzician(muzician);
    List<Muzician> listaMuzicieni = muzicianDaoImpl.afiseazaMuzicieni();
    request.setAttribute("listaMuzicieni", listaMuzicieni);
    RequestDispatcher rd = request.getRequestDispatcher("tabela_Muzicieni.jsp");
    rd.forward(request, response);
}
```

Figure 3.9: Functia stergeMuzician() din Controller

3.4 DAO (Data Access Object)

Directorul DAO conține interfețe care abstractizează accesul la date. Scopul acestora este de a ascunde detalii despre accesul la sursa de date.

- MuzicieniDao.java: Interfața pentru accesul la datele muzicienilor.
- StiluriMuzicaleDao.java: Interfața pentru accesul la datele stilurilor muzicale.
- ConcerteDao.java: Interfața pentru accesul la datele concertelor.

```java
public interface MuzicieniDao {
    public void adaugaMuzicieni (Muzician muzician);
    public List<Muzician> afiseazaMuzicieni();
    public void modificaMuzicieni (Long idmuzician, String nume, String prenume, Date datanasterii,
            String taraorigine, String instrumentprincipal);
    public void stergeMuzician (Muzician muzician);
}
```

Figure 3.10: interfata abstractizata MuzicieniDAO

3.5 DAOImpl

Directorul DAOImpl conține implementările interfețelor DAO. Acestea concretizează modul în care operațiunile de acces la date sunt efectuate.

- MuzicienieDaoImpl.java: Implementează operațiunile pentru accesul la datele muzicienilor folosind Hibernate.
- StiluriMuzicaleDaoImpl.java: Implementează operațiunile pentru accesul la datele stilurilor muzicale.
- ConcerteDaoImpl.java: Implementează operațiunile pentru accesul la datele concertelor.
- HibernateUtil.java: O clasă utilitară pentru a configura și a oferi accesul la sesiunea Hibernate, care este punctul de intrare în interacțiunea cu baza de date prin Hibernate.

```
public void adaugaMuzicieni(Muzician muzician) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    session.save(muzician);
    transaction.commit();
    session.close();
}
```

Figure 3.11: Functia adaugaMuzicieni() din DaoImpl

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

Figure 3.12: Functia afiseazaMuzicieni() din DaoImpl

```
public void modificaMuzicieni(Long idmuzician, String nume, String prenume, Date datanasterii, String taraorigine,
        String instrumentprincipal) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    Muzician detaliimuzicieni = (Muzician) session.load(Muzician.class, idmuzician);
    detaliimuzicieni.setNume(nume);
    detaliimuzicieni.setPrenume(prenume);
    detaliimuzicieni.setDatanasterii(datanasterii);
    detaliimuzicieni.setTaraorigine(taraorigine);
    detaliimuzicieni.setInstrumentprincipal(instrumentprincipal);
    session.update(detaliimuzicieni);
    transaction.commit();
    session.close();
}
```

Figure 3.13: Functia modificaMuzicieni() din DaoImpl

```java
public void stergeMuzician(Muzician muzician) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    session.delete(muzician);
    transaction.commit();
    session.close();
}
```

Figure 3.14: Functia stergeMuzician() din DaoImpl

```java
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Figure 3.15: Fisierul HibernateUtil

3.6 Pojo

Directorul pojo (Plain Old Java Objects) conține clasele entitate care reprezintă tabelele din baza de date. Acestea sunt utilizate de Hibernate pentru a mapea rândurile bazei de date la obiecte Java.

- Muzician.java: Clasa entitate pentru un tabel de muzicieni.
- StilMuzical.java: Clasa entitate pentru un tabel de stiljuri muzicale.
- Concert.java: Clasa entitate pentru un tabel de concerte.

```java
public class Muzician implements java.io.Serializable {

    private Long idmuzician;
    private String nume;
    private String prenume;
    private Date datanasterii;
    private String taraorigine;
    private String instrumentprincipal;
    private Set<Concert> concertes = new HashSet<>(0);

    public Muzician() {
    }

    public Muzician(String nume, String prenume, Date datanasterii, String taraorigine, String instrumentprincipal,
            Set<Concert> concertes) {
        this.nume = nume;
        this.prenume = prenume;
        this.datanasterii = datanasterii;
        this.taraorigine = taraorigine;
        this.instrumentprincipal = instrumentprincipal;
        this.concertes = concertes;
    }
```

Figure 3.16: Clasa Muzician si constructorul

```java
public Long getIdmuzician() {
    return this.idmuzician;
}

public void setIdmuzician(Long idmuzician) {
    this.idmuzician = idmuzician;
}

public String getNume() {
    return this.nume;
}

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

public String getPrenume() {
    return this.prenume;
}

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

Figure 3.17: O parte din metodele getter si setter ale clasei

3.7 Resurse XML

Fișierele XML sunt utilizate pentru configurarea Hibernate și mapearea obiectelor POJO la tabelele din baza de date.[2]

- *.hbm.xml: Fiecare din aceste fișiere este un fișier de mapare pentru Hibernate care asociază o clasă POJO cu un tabel din baza de date.
- hibernate.cfg.xml: Configurația generală pentru Hibernate, inclusiv detalii despre conexiunea la baza de date și alte proprietăți.
- hibernate.reveng.xml: Un fișier de inginerie inversă folosit probabil pentru a genera automat mapeările și clasele POJO pe baza schemei bazei de date.

```xml
<hibernate-mapping>
    <class name="pojo.Muzician" table="muzicieni" catalog="mihnea"
        optimistic-lock="version">
        <id name="idmuzician" type="java.lang.Long">
            <column name="idmuzician" />
            <generator class="identity" />
        </id>
        <property name="nume" type="string">
            <column name="nume" length="255" />
        </property>
        <property name="prenume" type="string">
            <column name="prenume" length="45" />
        </property>
        <property name="datanasterii" type="date">
            <column name="datanasterii" />
        </property>
        <property name="taraorigine" type="string">
            <column name="taraorigine" length="100" />
        </property>
        <property name="instrumentprincipal" type="string">
            <column name="instrumentprincipal" length="100" />
        </property>
        <set name="concertes" table="concerte" inverse="true"
            lazy="true" fetch="select">
            <key>
                <column name="idmuzician" />
            </key>
            <one-to-many class="pojo.Concert" />
        </set>
    </class>
</hibernate-mapping>
```

Figure 3.18: Fisierul de mapare pentru ”Muzicieni”

```xml
<hibernate-configuration>
  <session-factory>
      <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.query.factory_class">org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mihnea</property>
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password"></property>
    <mapping resource="pojo/Concerte.hbm.xml"/>
    <mapping resource="pojo/Muzicieni.hbm.xml"/>
    <mapping resource="pojo/StiluriMuzicale.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Figure 3.19: Configurația hibernate

```
<hibernate-reverse-engineering>
  <schema-selection match-catalog="mihnea"/>
  <table-filter match-name="stilurimuzicale"/>
  <table-filter match-name="muzicieni"/>
  <table-filter match-name="concerte"/>
</hibernate-reverse-engineering>
```

Figure 3.20: Fisier inginerie inversa

3.8 Frontend - Bootstrap

```
<ul class="list-unstyled">
    <li><a href="adauga_Muzician.jsp" class="btn btn-primary btn-block"><b>Adauga Muzicieni</b></a></li>
    <li><a href="adauga_StilMuzical.jsp" class="btn btn-primary btn-block mt-3"><b>Adauga Stiluri Muzicale</b></a></li>
    <li><a href="adauga_Concert.jsp" class="btn btn-primary btn-block mt-3"><b>Adauga Concerte</b></a></li>
</ul>
```

Figure 3.21: Stilizarea tabelelor din pagina web folosind Bootstrap

Tabela Muzicieni:

IdMuzician:	Nume:	Prenume:	Data Nasterii:	Tara Origine:	Instrument Principal:
1	Popescu	Ion	2024-01-17	România	Chitară
2	Ionescu	Maria	1985-03-15	România	Pian
3	Vasile	Andrei	1992-08-10	România	Vioară

Figure 3.22: Tabela muzicieni in pagina web

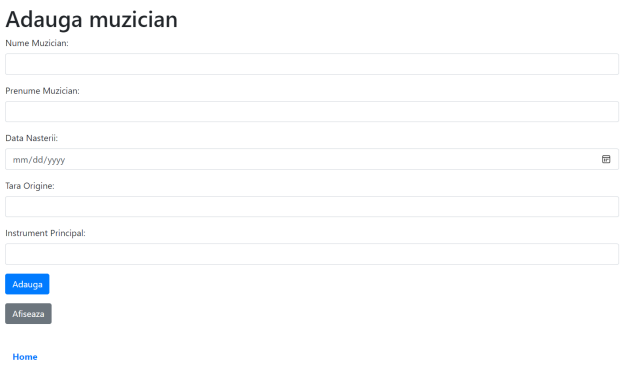

Figure 3.23: Formularul pentru adaugare

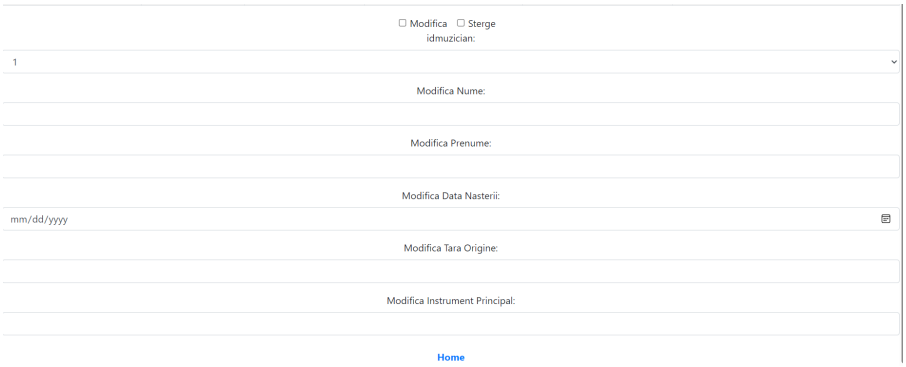

Figure 3.24: Formularul pentru modificare si stergere

4 Fluxul Aplicației

- **Interacțiunea Utilizatorului**: Utilizatorul interacționează cu aplicația printr-o interfață (o pagină web) și trimite o cerere către un controller.
- **Controller**: Controllerul primește cererea și determină ce acțiune să efectueze.
- **DAO**: Controllerul apelează o metodă corespunzătoare dintr-un DAO pentru a accesa datele necesare.
- **DAOImpl**: Implementarea DAO interacționează cu baza de date prin Hibernate folosind clasele POJO și fișierele de configurare și mapeare XML. Răspunsul: Odată ce datele sunt preluate și/sau procesate, controllerul trimite un răspuns înapoi la utilizator.

5 Concluzii

Prin implementarea operațiunilor fundamentale CRUD (Create, Read, Update, Delete), aplicația oferă o soluție directă pentru catalogarea artiștilor, clasificarea genurilor muzicale și planificarea evenimentelor live, facilitând astfel administrarea eficientă a informațiilor în domeniul muzical. Această abordare minimală și focalizată permite utilizatorilor să mențină și să acceseze ușor baza de date, făcând această aplicație esențială pentru organizatorii de concerte și entuziaștii muzicii.

6 Bibliografie

References

[1] MySQL. (2023). *MySQL Documentation*. https://dev.mysql.com/doc/.

[2] Hibernate. (2023). *Hibernate Documentation*. https://hibernate.org/orm/documentation/.

[3] Oracle. (2023). *Java Platform, Standard Edition & Java Development Kit Version Documentation*. https://docs.oracle.com/javase/.

[4] Baeldung. (2023). *The Persistence Layer with Hibernate Tutorial*. https://www.baeldung.com/hibernate.

[5] JavaTpoint. (2023). https://www.javatpoint.com/hibernate-tutorial.

[6] Bootstrap. (2023). *Bootstrap Documentation*. https://getbootstrap.com/docs/.

[7] W3Schools. (2023). *SQL Tutorial*. https://www.w3schools.com/sql/.

[8] MDN Web Docs. (2023). *HTML: Hypertext Markup Language*. https://developer.mozilla.org/en-US/docs/Web/HTML.

[9] Codecademy. (2023). *Learn JavaServer Pages*. https://www.codecademy.com/learn/learn-jsp.