



Universitatea POLITEHNICA din București

Facultatea de electronică, Telecomunicații și Tehnologia Informației



SOUND ARCHIVES

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

STUDENT: **ANDRONE IONUȚ**

GRUPA: **432C**

CUPRINS

1) Tema proiectului	pag. 3
2) Descrierea sistemului de gestiune a bazelor de date MySQL	pag. 3
1. Ce este MySQL.....	
2. Cum administreaza MySQL bazele de date.....	
3. De ce este atat de util MySQL.....	
3) Tehnologia Hibernate utilizata in dezvoltarea aplicatiei	pag. 4
1. Ce este Hibernate.....	
2. Arhitectura tehnologiei Hibernate.....	
4) Limbajul HTML si utilizarea sa in aplicatii	pag. 5
1. Ce este HTML.....	
2. In ce scop utilizam limbajul HTML.....	
5) Descrierea aplicatiei	pag. 5
1. Baza de date	pag. 5
2. Diagrama logica a bazei de date (Diagrama ERD)	pag. 7
3. Diagrama UML pentru aplicatia dezvoltata	pag. 8
4. Functionalitatea aplicatiei	pag. 9
1. Arhitectura proiectului.....	pag. 9
2. Implementarea functiilor.....	pag. 10
3. Partea de front-end.....	pag. 21
6) Concluzii	pag. 22
7) Bibliografie	pag. 22

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 precum JSP, Hibernate, JPA, .NET, Python+Django, Python+Flask, etc.

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

Pentru tema individuală primită, am ales 2 tehnologii diferite: JSP și Hibernate. În această prezentare ne vom orienta atenția către tehnologia Hibernate. Asocierea pentru tabelele din bază de date să fie de M:N.

2) Descrierea sistemului de gestiune a bazelor de date MySQL

2.1. Ce este MySQL?

MySQL reprezintă un sistem de gestionare a bazelor de date relationale 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), la fel ca alte sisteme, cum ar fi popularul Microsoft Access. În timp ce Microsoft Access, MySQL și alte sisteme de gestionare a bazelor de date servesc scopuri similare (de a găzdui datele), utilizarea diferă foarte mult. [1]

MySQL este componenta integrată a platformelor LAMP sau WAMP (Linux/Windows-Apache-MySQL-PHP/Perl/Python). Popularitatea sa ca aplicație web este strâns legată de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. În multe cărți de specialitate este precizat faptul că MySQL este mult mai ușor de învățat și folosit decât multe din aplicațiile de gestiune a bazelor de date, drept exemplu comanda de ieșire fiind una simplă și evidentă: „exit” sau „quit”. [2]

2.2. Cum administrează MySQL bazele 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. [2]

2.3. De ce este atat de util MySQL?

Baza de date MySQL este folosita in principal ca mijloc de a stoca date pentru aplicatii mari, bazate pe web. Site-uri precum WordPress, iStock, GitHub, Facebook, NASA, Marina SUA, Tesla, Scholastic, Spotify, YouTube, Netflix, Glasses Direct, Symantec (si multe altele) folosesc baza de date MySQL ca mijloc de stocare a datelor pe din interiorul sau exteriorul site-urilor web și serviciilor interne. [1]

3) Tehnologia Hibernate utilizata in dezvoltarea aplicatiei

3.1. Ce este Hibernate?

Cunoscut oficial sub denumirea de Hibernate ORM, acesta este un instrument de cadru relational de obiecte sau cadru pentru limbajul de programare Java. Este software-ul gratuit sub GNU Lesser General Public License 2.1, care isi propune sa ofere utilizatorului cadrul pentru a gestiona maparea impedantelor relaționale obiecte.

De asemenea, acesta gestioneaza accesuri persistente la baze de date cu functii de gestionare a obiectelor la nivel inalt, fiind un cadru de Java middleware folosit pentru maparea relationala obiect și pentru realizarea persistentei eficiente a obiectelor.[3]

3.2. Arhitectura tehnologiei Hibernate

Arhitectura tehnologiei Hibernate cuprinde:

- Un cod de aplicație Java ce constă in toate clasele, variabilele și obiectele care definesc logica de afaceri a aplicației. Aceste clase comunică cu Hibernate.
- Principiile Hibernate, stocarea, salvarea sau recuperarea obiectelor claselor noastre, comunicând cu stratul serverului de baze de date.
- Utilizarea API-ului de bază Java, Java Database Connectivity (JDBC), Java Transaction API (JTA), Java Naming și Directory Interface (JNDI) pentru a comunica cu baza de date cu scopul de a afla starea unui obiect executând citirea, crearea, actualizarea, ștergerea (operațiile CRUD).[2]

[illegible]

Ce este asocierea M:N?

Asocierea M:N (mai-multi-la-mai-multi) are ca si caracteristica faptul ca fiecarui element inregistrat intr-o tabela i se poate fi asociate mai multe elemente din cealalta tabela si invers. [5]

De exemplu in cazul nostru, un album poate fi asociat mai multor genuri muzicale, asa cum si unui gen muzical ii pot fi asociate mai multe albume.

Ce reprezinta tabela de legatura?

Pentru a crea o relație mai-multi-la-mai-multi, trebuie sa se creeze o a treia tabela denumita deseori tabela de joctiune, care imparte relatia mai-multi-la-mai-multi în doua relatii unu-la-mai-multi. In cazul nostru, am ales ca si tabela de legatura tabela **artisti**.

In aceasta noua tabela, atributele ce au fost selectate ca si chei primare pentru tabelele anterioare vor deveni chei straine (FK) pentru tabela de legatura **artisti**.




Table Name:

Charset/Collation:

Schema: **tema**

Engine: **InnoDB**

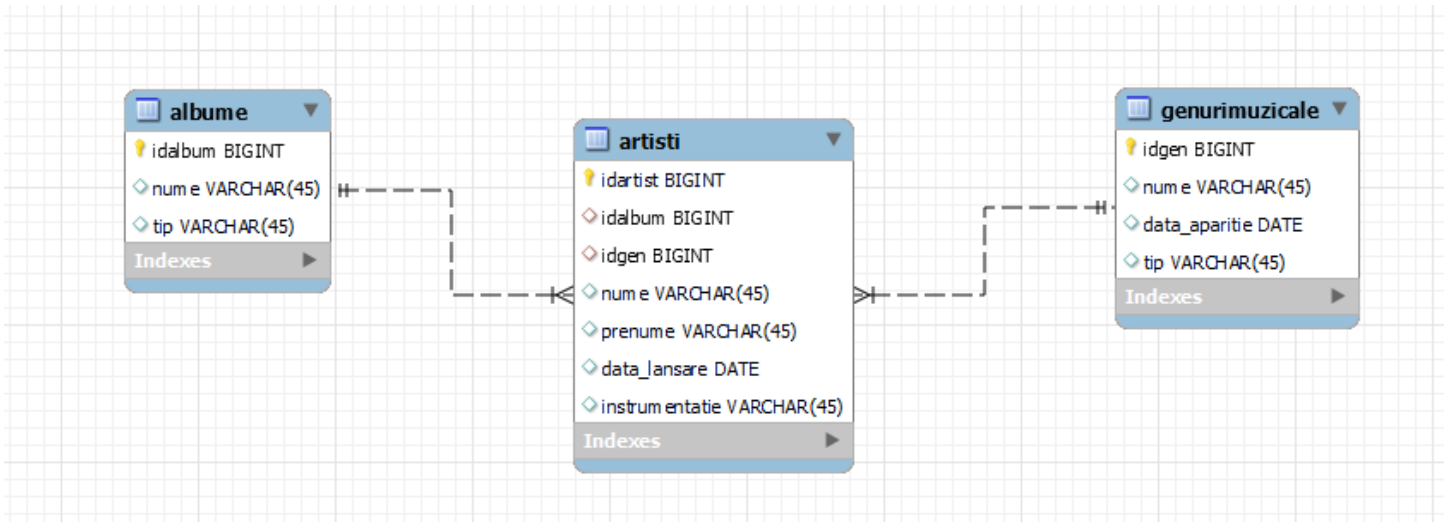
Comments:

Foreign Key Name	Referenced Table	Column	Referenced Column
fk_artisti_1	tema, albume		
fk_artisti_2	tema, genurimuzicale		
<input type="text"/>			

Pentru aceasta tabela, am ales atributul idartist ca si cheie primara. Restul atributelor sunt cheile straine idalbum si idgen si: nume, prenume, data_lansare, trupa.

[illegible]

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

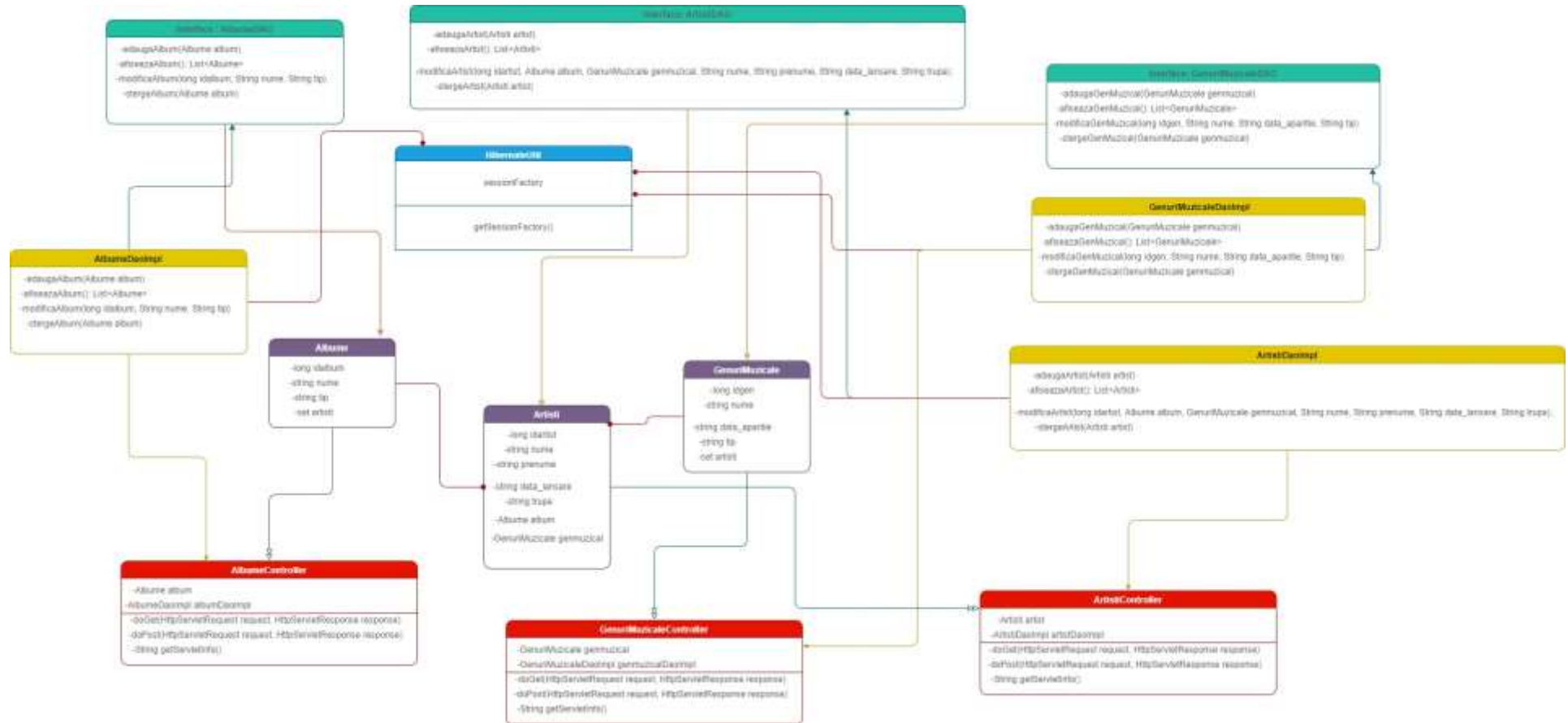


De ce este importanta diagrama logica a bazei de date?

Aceasta ofera o reprezentare logica detaliata a datelor celor 3 tabele, a relatiilor dintre ele.

Ce relatii exista intre cele trei tabele?

- ➔ Intre **albume** si **artisti** exista o asociere de tip 1:N
- ➔ Intre **albume** si **genurimuzicale** exista o asociere de tip M:N
- ➔ Intre **genurimuzicale** si **artisti** exista o asociere de tip 1:N

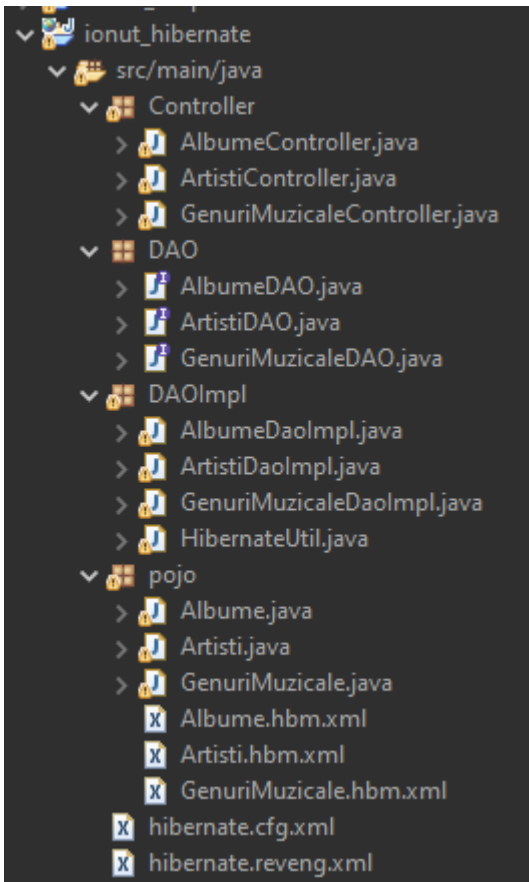


5.4. Functionalitatea aplicatiei

5.4.1. Arhitectura proiectului

Am ales o structura particulara a proiectului pentru a marca separarea functionalitatii aplicatiei noastre in ceea ce priveste partea de back-end.

Arhitectura proiectului este de tip Model View Controller (MVC).



Asadar putem spune ca proiectul e constituit din:

- **Partea de model:** identificata in POJO, unde se face maparea. In clasele din acest package, cele mai importante elemente sunt functiile de GET si SET. Cand se fac operatii pe obiectele din clasa, modificarile se propaga in baza de date.

- **Partea de functionalitate:** este reprezentata de urmatoarele pachete: Controller, DAO, DaoImpl.

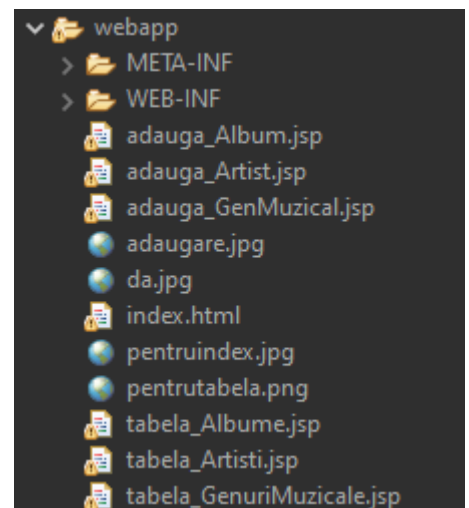
Aici se fac operatiile de baza pe obiectele din clase, din acest motiv, aceasta parte este considerata partea dinamica a proiectului. In DAO avem interfete pentru standardizarea denumirilor ce au fost implementate in DAOImpl, unde toate operatiile de aici sunt facute in cadrul unor tranzactii.

Controller-ul este implementat sub firma de SERVLET.

Totodata o alta parte importanta este cea de **vizualizare**, implementata prin JSP, prin limbajul HTML ce contureaza in totalitate front-end-ul proiectului.

Ce este JSP?

JSP este o tehnologie care permite dezvoltatorilor sa genereze pagini web dinamice, utilizand fragmente de cod Java, introduse in pagini HTML.



Hibernate functioneaza ca un serviciu de mapare (ORM). Incercam sa mapam tabelele si toate datele pe care le avem in baza noastra de date, apoi sa le transfiguram in partea de OBIECT-ORIENTAT, unde complexitatea creste si unde avem libertate in utilizarea metodelor.

5.4.2. Implementarea functiilor

In pachetul POJO, luam ca si exemplu clasele Album.java si GenuriMuzicale.java . In primul rand, se declara attributele, ce vor fi mapate si vor fi utilizate in cadrul functiilor aplicatiei:

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

    private Long idalbum;
    private String nume;
    private String tip;

    private Set artisti = new HashSet(0);

    public Album() {
    }

    public Album(String nume, String tip, Set artisti) {
        this.nume = nume;
        this.tip = tip;
        this.artisti = artisti;
    }
}

public class GenuriMuzicale implements java.io.Serializable {

    private Long idgen;
    private String nume;
    private String data_aparitie;
    private String tip;
    private Set artisti = new HashSet(0);

    public GenuriMuzicale() {
    }

    public GenuriMuzicale(String nume, String data_aparitie, String tip, Set artisti) {
        this.nume = nume;
        this.tip = tip;
        this.data_aparitie = data_aparitie;
        this.artisti = artisti;
    }
}
```

Echivalentul liniilor din tabelele noastre (albume, genurimuzicale, artisti) vor fi obiectele instantia la clase.

Totodata pentru fiecare atribut din cele doua clase ne trebuie cate o metoda de GET si SET:

```
public Long getIdalbum() {
    return this.idalbum;
}

public void setIdalbum(Long idalbum) {
    this.idalbum = idalbum;
}

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

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

public String getTip() {
    return this.tip;
}

public void setTip(String tip) {
    this.tip = tip;
}

public Long getIdgen() {
    return this.idgen;
}

public void setIdgen(Long idgen) {
    this.idgen = idgen;
}

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

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

public String getTip() {
    return this.tip;
}

public void setTip(String tip) {
    this.tip = tip;
}

public String getData_aparitie() {
    return this.data_aparitie;
}

public void setData_aparitie(String data_aparitie) {
    this.data_aparitie = data_aparitie;
}
```

Se creeaza fisierele de tip xml pentru functia de configurare. Aici se va face maparea propriu-zisa. De exemplu:

```
<hibernate-mapping>
<class name="pojo.Albume" table="albume" catalog="tema" optimistic
<id name="idalbum" type="java.lang.Long">
  <column name="idalbum" />
  <generator class="identity" />
</id>
<property name="nume" type="string">
  <column name="Nume" length="45" />
</property>
<property name="tip" type="string">
  <column name="Tip" />
</property>
<set name="artisti" table="artisti" inverse="true" lazy="true"
  fetch="select">
  <key>
    <column name="idalbum" />
  </key>
  <one-to-many class="pojo.Artisti" />
</set>
</class>
</hibernate-mapping>
```

```
<hibernate-mapping>
<class name="pojo.GenuriMuzicale" table="genurimuzicale" catalog=
<id name="idgen" type="java.lang.Long">
  <column name="idgen" />
  <generator class="identity" />
</id>
<property name="nume" type="string">
  <column name="Nume" length="45" />
</property>
<property name="data_aparitie" type="string">
  <column name="Data_aparitie" length="45" />
</property>
<property name="tip" type="string">
  <column name="Tip" length="45" />
</property>
<set name="artisti" table="artisti" inverse="true" lazy="true"
  fetch="select">
  <key>
    <column name="idgen" />
  </key>
  <one-to-many class="pojo.Artisti" />
</set>
</class>
</hibernate-mapping>
```

Partea de obiect-orientat este marcata chiar prin: `<property name="nume" type="string">`

Partea de relationare este reprezentata ca si exemplu de `<column name="Nume" length="45" />`

Totodata, aplicatia este una tranzactionala.

Ce este o tranzactie?

O tranzactie este o unitate logica de prelucrare indivizibila (atomica) a datelor unei baze de date, prin care se asigura consistenta acesteia. In principiu, orice executie a unui program care acceseaza o baza de date poate fi considerata o tranzactie, daca baza de date este într-o stare consistenta, atat inainte, cat si dupa executie. O tranzactie trebuie sa asigure consistenta bazei de date indiferent daca a fost executata individual sau concurent cu alte tranzactii, precum si in conditiile in care au aparut erori in cursul executiei tranzactiei.[6]

Care sunt proprietatile tranzactiei?

- **Atomicitatea** : este proprietatea unei tranzacții de a reprezenta o unitate de execuție indivizibilă, adică de a executa "totul sau nimic". Dacă o tranzacție este întreruptă dintr-o cauză oarecare, atunci sistemul SGBD va asigura, după eliminarea cauzei care a întrerupt executarea tranzacției, fie completarea și validarea tranzactiei, fie abandonarea tranzacției și anularea tuturor efectelor acțiunilor acesteia până în momentul întreruperii apărute.
- **Consistentă** : referă proprietatea acesteia de a efectua modificări corecte ale bazei de date. Cu alte cuvinte, o tranzacție transformă baza de date dintr-o stare consistentă în altă stare consistentă

- **Izolarea:** este proprietatea unei tranzacții de a face vizibile modificările efectuate numai după ce a fost validată (committed). Dacă în acest timp sunt executate alte tranzacții concurente, acestea nu "văd" modificările parțiale efectuate de tranzacția respectivă până în momentul încheierii-validării tranzacției.
- **Durabilitatea:** este proprietatea prin care, după validarea unei tranzacții, modificările efectuate de aceasta în baza de date nu vor mai fi pierdute datorită unor defectări ulterioare a sistemului. Proprietatea de durabilitate este asigurată prin metode de refacere (recovery) ale sistemului SGBD.[6]

În pachetul DAOImpl, clasa HibernateUtil ne ajuta sa obtinem o sesiune pentru a face modificarile in ceea ce priveste datele din baza de date.

```
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;
    }
}
```

Fisierul hibernate.cfg.xml este unul de configurare. Aici putem observa configurarea driver-ului de MySQL, partea de conectica (prin API), si totodata maparea legata de clasele din POJO.

```
5 <property name="hibernate.hbm2ddl.auto">update</property>
6 <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
7 <property name="hibernate.show_sql">true</property>
8 <property name="hibernate.query.factory_class">org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory</property>
9 <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/tema</property>
10 <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
```

Cum se afiseaza datele?

Trebuie sa ne indreptam atentia catre fisierul index.html, unde am implementat functia de afisare pentru datele din tabelele noastre.

Am implementat 3 butoane ce ne vor redirectiona catre paginile ce contin tabele cu date: tabela_Albume.jsp , tabela_GenuriMuzicale.jsp, tabela_Artisti.jsp.



```
18 <form action="GenuriMuzicaleController" method="POST">
19 <button class="btn btn-primary" type="submit" name="afiseazaGenMuzical" value="Afiseaza" style="text-decoration:none;margin: 106px -2px 4px 886px;margin-top: 171px;margin
20 </form>
21 <form action="AlbumeController" method="POST">
22 <button class="btn btn-primary" type="submit" name="afiseazaAlbum" value="Afiseaza" style="text-decoration:none;margin: 106px -2px 4px 886px;margin-top: 163px;margin-right: -2px;
23 </form>
24 <form action="ArtistiController" method="POST">
25 <button class="btn btn-primary" type="submit" name="afiseazaArtist" value="Afiseaza" style="text-decoration:none;margin: 106px -2px 4px 886px;margin-top: 113px;margin-right: -2px
26 </form>
27
```

Pentru fiecare buton implementat, avem o caracteristica de “submit” ce ne va face legatura cu functiile din pachetul Controller (prin afiseazaGenMuzical, afiseazaAlbum, afiseazaArtist). De exemplu pentru afisarea albumelor inregistrate in baza de date, in AlbumeController functia de afisare este implementata astfel:

```
if (request.getParameter("afiseazaAlbum") != null) {
    List<Album> listaAlbume = new ArrayList();
    listaAlbume = albumDaoImpl.afiseazaAlbum();
    request.setAttribute("listaAlbume", listaAlbume);
    RequestDispatcher rd = request.getRequestDispatcher("tabela_Albume.jsp");
    rd.forward(request, response);
}
```

Se creeaza o lista de obiecte de tip Album. Aceasta lista trebuie sa fie umpluta de obiecte de acest tip, asadar se face legatura cu DAOImpl, unde se creeaza o sesiune, in cadrul careia se face o interogare (“From Albume”). Aceasta este echivalenta cu comanda din MySQL **select * from**.

Aici se creeaza lista de obiecte “albume” ce continue toate datele din tabela “albume” din baza de date, apoi aceasta este returnata.


```

24 public List<Album> afiseazaAlbum() {
25     List<Album> listaAlbum = new ArrayList();
26     Session session = HibernateUtil.getSessionFactory().openSession();
27     org.hibernate.Query query = session.createQuery("From Album");
28     listaAlbum = query.list();
29     return listaAlbum;
30 }

```

În cadrul clasei AlbumController se seteaza un atribut asociat, iar obiectele vor fi trimise pentru vizualizare:

```
request.setAttribute("listaAlbum", listaAlbum);
```

```
RequestDispatcher rd = request.getRequestDispatcher("tabela_Album.jsp");
```

Datele vor fi vizualizate în cadrul paginii tabela_Album:

Pentru afisare se foloseste tag-ul `<c:forEach>` Cu ajutorul variabilei "album", vom itera lista de albume.

Prin `${album.nume}` si `${album.tip}` se preiau attributele, numele lor de la fiecare obiect si se realizeaza afisarea.

```

<c:forEach var="album" items="${listaAlbum}">
  <tr style="text-align: center; box-shadow: 0px 0px 9px;">
    <td style="width: 75.656px; filter: contrast(08) saturate(53%); transform: rotate(0deg); border-top-left-radius: 0px; font-family: Acme, sans-serif;">${album.idalbum}</td>
    <td style="width: 227.078px; font-family: Acme, sans-serif;">${album.nume}</td>
    <td style="font-family: Acme, sans-serif;">${album.tip}</td>
  </tr>

```

Datele din tabela albume vor fi afisate astfel după ce vom apăsa butonul:

Home Genuri muzicale înregistrate Artiști înregistrați		
Id	Nume album	Tipul albumului
5	At Last!	album de studio
6	LSD	album de studio
7	Glee: The Graduation	coloana sonora
8	Dancing Queen	album tribut
9	New Jersey	album de studio
10	Dawn of the Black Hearts	album live
11	Blood on the Dance Floor	album remixat
12	Trio	album de studio

Cum se adauga datele?

Luam ca si exemplu fisierul `adauga_GenMuzical.jsp` in cadrul caruia se face adaugarea genurilor muzicale pe care dorim sa le inregistram.

In primul rand am creat un formular pentru a prelua parametrii, si de a-i adauga in baza de date:

```
<form action="GenuriMuzicaleController" method="GET">
<button class="Flicker" type="submit" id="inregistra" name="adaugaGenMuzical" value="Adauga" style="margin-top: 25px;margin-right: 25px;margin-left: 20px;padding: 15px 12px;width: 230px;" />
</div>
<div style="height: 72px;border: 1px solid black;background-color: #e0e0e0;margin-top: 10px;margin-right: 10px;margin-left: 10px;">
</div>
</div>
<div style="width: 550px;height: 30px;font-weight: bold;font-size: 20px;font-family: Lucida Handwriting;text-align: center"> Numele genului muzical:
<input class="no-outline" type="text" name="nume">
</div>
<div style="width: 550px;height: 30px;font-weight: bold;font-size: 20px;font-family: Lucida Handwriting;text-align: center"> Data aparitiei genului muzical:
<input class="no-outline" type="text" name="data_aparitie">
</div>
<div style="width: 550px;height: 30px;font-weight: bold;font-size: 20px;font-family: Lucida Handwriting;text-align: center"> Tipul genului muzical:
<input class="no-outline" type="text" name="tip">
</div>
</form>
```

Cand apasam butonul (SUBMIT), se va face legatura cu servlet-ul `GenuriMuzicaleController` si prin metoda `GET` vom obtine parametrii doriti.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    if (request.getParameter("adaugaGenMuzical") != null) {
        String nume = request.getParameter("nume");
        String data_aparitie = request.getParameter("data_aparitie");
        String tip = request.getParameter("tip");
        genmuzical.setNume(nume);
        genmuzical.setData_aparitie(data_aparitie);
        genmuzical.setTip(tip);
        if(nume != "" && data_aparitie != "" && tip != "") {
            genmuzicalDaoImpl.adaugaGenMuzical(genmuzical);
            List<GenuriMuzicale> listaGenuriMuzicale = new ArrayList();
            listaGenuriMuzicale = genmuzicalDaoImpl.afiseazaGenMuzical();
            request.setAttribute("listaGenuriMuzicale", listaGenuriMuzicale);
            RequestDispatcher rd = request.getRequestDispatcher("tabela_GenuriMuzicale.jsp");
            rd.forward(request, response);
        }
        else {
            RequestDispatcher rd = request.getRequestDispatcher("adauga_GenMuzical.jsp");
            rd.forward(request, response);
        }
    }
}
```

Linia 27 pune in evidenta daca am apasat butonul respectiv. Daca este apasat, se preiau datele din interfata pe care le-am adaugat, dar cu conditia ca toate casutele au fost completate:

```
if(nume != "" && data_aparitie != "" && tip != "")
```

Daca una dintre casute nu este completata, atunci datele nu vor fi adaugate.

Prin comanda de la linia 35, adaugam obiectul in baza noastra de date, daca conditia este respectata.

Funcția propriu-zisă de adăugare este implementată în GenuriMuzicaleDaoImpl astfel:

```
12 public class GenuriMuzicaleDaoImpl implements GenuriMuzicaleDAO{
13
14     public void adaugaGenMuzical(GenuriMuzicale genmuzical) {
15         Session session = HibernateUtil.getSessionFactory().openSession();
16         Transaction transaction = session.beginTransaction();
17         session.save(genmuzical);
18         transaction.commit();
19         session.close();
20     }
21 }
```

Operația se face într-o sesiune. Se începe o tranzacție la linia 15. Prin `session.save(genmuzical);` datele obiectului au fost duse până în baza de date. Prin `transaction.commit();` tranzacția a fost validată, astfel încât va fi imposibilă întoarcerea la operațiile de start transaction. La linia 19, tranzacția se închide.

Va rugăm să introduceți datele unui nou gen muzical.

Numele genului muzical:

Data apariției genului muzical:

Tipul genului muzical:

Back la home

Adauga genul muzical

După ce datele au fost adăugate, apăsând pe butonul de adăugare. Vom fi redirectionați pe pagina ce conține tabelul completat:

Adauga un gen muzical

☐ Modificati linia selectata

☐ Stergeti linia selectata

Selectati linia: 49

Modifica numele genului muzical

Modifica data aparitiei genului muzical

Modifica tipul genului muzical

	Nume gen muzical	Data aparitiei genului muzical	Tipul genului muzical
1	Jazz	Secolul XX	Modern
2	Rock	Secolul XX - Secolul XXI	Modern
3	Dans	Anul 1997	Vocal/Gitară
4	Folk	Secolul XX	Modern
5	Classical	1800	Religioasă/Baladă
6	Saxofon	Secolul XX/XXI	Străvechi
7	Class	1900	Modern
8	Country	Secolul XX	Modern

Cum se modifica si cum se sterg datele?

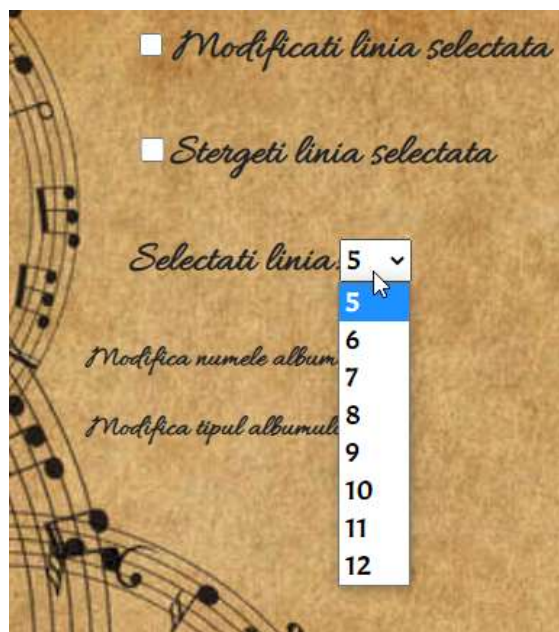
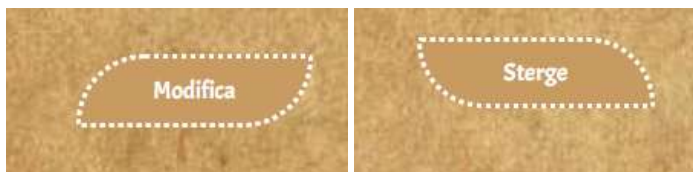
Luam ca si exemplu modificarea si stergerea datelor unui album, asadar ne vom orienta atentia catre fisierul tabela_Albume.jsp .

Se creeaza un nou formular diferit de cel in care aplicam functia de adaugare. In noul formular, prin metoda POST facem legatura cu AlbumController prin alegerea a doua functii: modificareAlbum (pentru modificarea unor date) sau stergereAlbum(pentru stergerea datelor). La functia de stergere exista o particularitate, si anume ca am dezactivat completarea casutelor, deoarece dorim doar sa stergem niste date:

```
32 <p style="margin-left: 34px;padding: 12px;margin-top: 4px;font-family: Allura, serif;font-weight: bold;font-size: 30px;"><input type="checkbox" id="update" style="width: 30px;height: 15px;" style="margin-left: 30px;vertical-align: middle;" /> Modificati linia selectata
33 <p style="margin-left: 34px;padding: 12px;margin-top: 4px;font-family: Allura, serif;font-weight: bold;font-size: 30px;"><input type="checkbox" id="delete" style="width: 30px;height: 15px;" style="margin-left: 30px;vertical-align: middle;" /> Stergeti linia selectata
34 <p style="margin-left: 34px;padding: 12px;margin-top: 4px;font-family: Allura, serif;font-weight: bold;font-size: 30px;">Selectati linia:<select name="idalbum" style="width: 50px;vertical-align: middle;">
35 <option value="${listaAlbum}" var="album">
36 </option>
37 </select>
38 </p>
39 </div>
```

Prin tag-ul `<c:forEach>` si prin "select" am parcurs lista de albume si prin "option" am selectat idalbum ca si element principal dupa care vrem sa realizem stergerea sau modificarea.

O data selectata optiunea de modificare (dupa ce am completat datele in casutele de input) sau cea de stergere si linia pe care dorim sa facem operatia, in josul paginii va aparea un buton (tip SUBMIT):



Cand apasam, se va face legatura cu functiile din clasa AlbumController:

```
if (request.getParameter("modificaAlbum") != null) {
    long id1 = java.lang.Long.parseLong(request.getParameter("idalbum"));
    String nume = request.getParameter("nume");
    String tip = request.getParameter("tip");
    if(nume != "" && tip != "") {
        albumDaoImpl.modificaAlbum(id1, nume, tip);
        List<Album> listaAlbum = new ArrayList();
        listaAlbum = albumDaoImpl.afiseazaAlbum();
        request.setAttribute("listaAlbum", listaAlbum);
        RequestDispatcher rd = request.getRequestDispatcher("tabela_Albume.jsp");
        rd.forward(request, response);
    }
    else {
        RequestDispatcher rd = request.getRequestDispatcher("adauga_Album.jsp");
        rd.forward(request, response);
    }
}
```

Daca butonul pentru modificare este apasat datele vor fi preluate, insa cu conditia ca acestea sa fie complete, adica casutele sa fie completate integral. Ca si in cazul adaugarii si afisarii, de aici se face legatura cu AlbumDaoImpl:

```

public void modificaAlbum(long idalbum, String nume, String tip) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    Albume detaliialbume = (Albume) session.load(Albume.class, idalbum);
    detaliialbume.setNume(nume);
    detaliialbume.setTip(tip);
    session.update(detaliialbume);
    transaction.commit();
    session.close();
}

```

Se creeaza din nou o sesiune, se incepe o tranzactie avand ca urmare incarcarea datelor obiectului selectat prin alegerea unui "idalbum" din interfata. Pe obiect se seteaza noile valori pe care le scriem in casutele din interfata, si se realizeaza un update pe sesiune cu detaliile obiectului modificat, apoi se valideaza tranzactia urmand ca sesiunea sa fie inchisa.

In cazul in care butonul pentru stergere este apasat, legatura va fi stabilita cu functia stergeAlbum din AlbumeController:

```

if (request.getParameter("stergeAlbum") != null) {
    long id2 = Long.parseLong(request.getParameter("idalbum"));
    album.setIdalbum(id2);
    albumDaoImpl.stergeAlbum(album);
    RequestDispatcher rd = request.getRequestDispatcher("adauga_Album.jsp");
    rd.forward(request, response);
}

```

Se preluеaza id-ul albumului si in AlbumeDaoImpl se face din nou o sesiune, o tranzactie, iar obiectul este sters din baza de date:

```

public void stergeAlbum(Albume album) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    session.delete(album);
    transaction.commit();
    session.close();
}

```

Ce particularitati are tabela de legatura in ceea ce priveste operatiile ce se pot realiza?

Desi majoritatea operatiilor pe care le-am realizat in ceea ce priveste tabela artisti seamana cu cele facute pe albume si genurimuzicale, functiile difera putin deoarece apar si asocieri cu cele doua tabele.

In fisierul adauga_Artist.jsp, lista de albume si lista de genuri muzicale sunt incarcate cu functiile din DAOImpl:

```

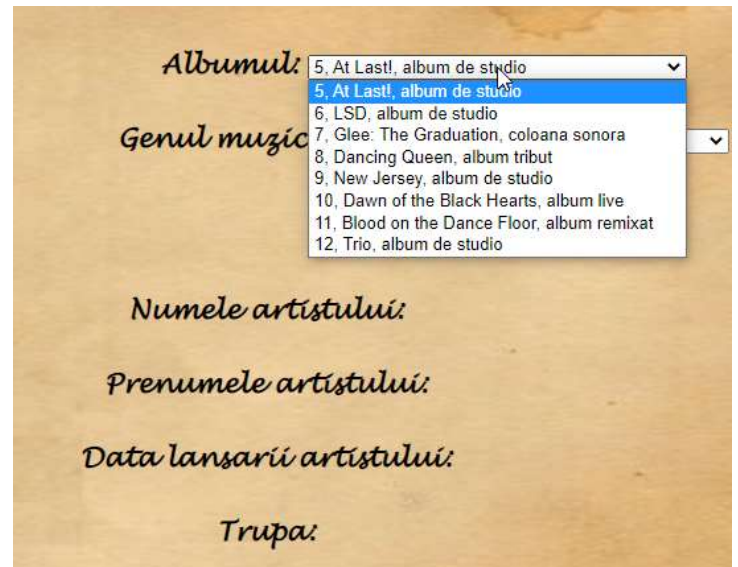
AlbumeDaoImpl albumDaoImpl = new AlbumeDaoImpl();
GenuriMuzicaleDaoImpl genmuzicalDaoImpl = new GenuriMuzicaleDaoImpl();
List<Albume> listaAlbume = new ArrayList();
listaAlbume = albumDaoImpl.afiseazaAlbum();
List<GenuriMuzicale> listaGenuriMuzicale = new ArrayList();
listaGenuriMuzicale = genmuzicalDaoImpl.afiseazaGenMuzical();
request.setAttribute("listaAlbume", listaAlbume);
request.setAttribute("listaGenuriMuzicale", listaGenuriMuzicale);

```

Cu ajutorul ultimelor functii, se incarca efectiv in pagina listele de albume si genuri muzicale.

In fisierul pentru adaugare am implementat 2 drop downs. Parcurgem lista de genuri muzicale respectiv, de albume.

Am implementat niste casute (input) pentru adaugarea numelui artistului, prenumelui, data lansarii si trupa din care face parte:



```
<p style="width: 956px; height: 36px; font-weight: bold; font-size: 20px; font-family: Lucida Handwriting; text-align: center"> Numele artistului:
    <input class="no-outline" type="text" name="nume">
</p>
<p style="width: 956px; height: 36px; font-weight: bold; font-size: 20px; font-family: Lucida Handwriting; text-align: center"> Prenumele artistului:
    <input class="no-outline" type="text" name="prenume">
</p>
<p style="width: 956px; height: 36px; font-weight: bold; font-size: 20px; font-family: Lucida Handwriting; text-align: center"> Data lansarii artistului:
    <input class="no-outline" type="text" name="data_lansare">
</p>
<p style="width: 956px; height: 36px; font-weight: bold; font-size: 20px; font-family: Lucida Handwriting; text-align: center"> Trupa:
    <input class="no-outline" type="text" name="trupa">
</p>
```

Cand apasam butonul (SUBMIT) se face legatura cu metoda GET din ArtistiController:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    if (request.getParameter("adaugaArtist") != null) {
        // prelucrare parametrilor de intrare
        long idgen = java.lang.Long.parseLong(request.getParameter("idgen"));
        long idalbum = java.lang.Long.parseLong(request.getParameter("idalbum"));
        Session session = HibernateUtil.getSessionFactory().openSession();
        GenuriMuzicale genmuzical = (GenuriMuzicale) session.get(GenuriMuzicale.class, idgen);
        Albume album = (Albume) session.get(Albume.class, idalbum);

        String nume = request.getParameter("nume");
        String prenume = request.getParameter("prenume");
        String data_lansare = request.getParameter("data_lansare");
        String trupa = request.getParameter("trupa");

        artist.setGenurimuzicale(genmuzical);
        artist.setAlbume(album);
        artist.setNume(nume);
        artist.setPrenume(prenume);
        artist.setData_lansare(data_lansare);
        artist.setTrupa(trupa);
        if (nume != "" && prenume != "" && data_lansare != "" && trupa != "") {
            artistDaoImpl.adaugaArtist(artist);
            List<Artist> listaArtisti = new ArrayList();
            listaArtisti = artistDaoImpl.afiseazaArtist();
            request.setAttribute("listaArtisti", listaArtisti);
            RequestDispatcher rd = request.getRequestDispatcher("tabela_Artisti.jsp");
            rd.forward(request, response);
        }
        else {
            RequestDispatcher rd = request.getRequestDispatcher("adauga_Artist.jsp");
            rd.forward(request, response);
        }
    }
}
```

Se creeaza o noua sesiune, urmand apoi incarcarea in obiectele album si genmuzical datele primite din interfata, prin intermediul cheilor idalbum si idgen.

Cu metodele de SET se construiesc obiectul artist.

Se face legatura cu clasa ArtistiDaoImpl din pachetul DAOImpl si se face "push" pana in baza de date. Se creeaza o noua sesiune si o noua tranzactie asemanatoare cu exemplele anterioare.

```
public void modificaArtist(long idartist, Albume album, GenuriMuzicale genmuzical, String nume, String prenume, String data_lansare, String trupa) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transaction = session.beginTransaction();
    Artisti detaliartisti;
    detaliartisti = (Artisti) session.load(Artisti.class, idartist);
    detaliartisti.setGenurimuzicale(genmuzical);
    detaliartisti.setAlbume(album);
    detaliartisti.setNume(nume);
    detaliartisti.setPrenume(prenume);
    detaliartisti.setData_lansare(data_lansare);
    detaliartisti.setTrupa(trupa);
    session.update(detaliartisti);
    transaction.commit();
    session.close();
}
```

Afisarea se face linie cu linie folosind din nou tag-ul <c:forEach>

```
<thead>
<script>
<forEach var="artisti" items="${listaArtisti}">
<tr style="text-align: center; box-shadow: 4px 4px 3px #ccc">
<td style="width: 75.616px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
<td style="width: 227.078px; float: left; text-align: left; text-decoration: none; border-top: 1px solid #ccc; border-left: 1px solid #ccc; border-right: 1px solid #ccc; border-bottom: 1px solid #ccc; padding: 5px 5px 5px 5px;">
</td>
</tr>
</script>
</c:forEach>
</thead>
```

Datele din tabela de Artisti vor fi afisate astfel:

Adauga datele unui artist				Id	Numele artistului	Prenumele artistului	Data lansarii artistului	Trupa	Numele albumului	Genul muzical
Modificati linia selectata				11	Jones	Etta	1954	Nu face parte	At Last!	Jazz
Stergeți linia selectată				12	Porter	Duffy	1967	Nu face parte	True	Country
Selectati linia: 15				13	Mantel	Cary	2009	Blas	Blas: The Production	Pop
Modificati albumul: 5				14	Bryon	David	1984	Ben Zvi	New Jersey	Rock
Modificati genul muzical: 49				15	Ranash	Linda	1949	Nu face parte	True	Country
Modifica numele artistului:				16	Jackson	Michael	1971	Nu face parte	Blood on the Dance Floor	Pop
Modifica prenumele artistului:				17	Bergasi	John Francis	1983	Ben Zvi	New Jersey	Rock
Modifica data lansarii artistului:				18	Scritson	Charlye	1965	Nu face parte	Dancing Queen	Pop
Modifica trupa:				19	Furter	Sia	1990	Nu face parte	LSD	Pop

In cazul operatiilor de modifica si stergere, acestea se aseamana cu cele realizate in tabelele albume si genurimuzicale, cu exceptia ca pentru functia de modificare am mai adaugat 2 drop downs pentru selectarea albumului si genului muzical pe care dorim sa il modificam:

```
<c:forEach items="${listaalbume}" var="albume">
<option value="${albume.idalbum}">${albume.idalbum}, ${albume.nume}, ${albume.tip}</option>
</c:forEach>
</select>
<p style="margin-left: 100px; padding: 12px; margin-top: 8px; font-family: Allura, serif; font-weight: bold; font-size: 30px;">Modificati genul muzical:
<c:forEach items="${listaGenuriMuzicale}" var="genurimuzicale">
<option value="${genurimuzicale.idgen}">${genurimuzicale.idgen}, ${genurimuzicale.nume}, ${genurimuzicale.data_aparitiei}, ${genurimuzicale.tip}</option>
</c:forEach>
</select>
```

5.4.3. Partea de front-end

Pentru un aspect placut al interfeței, am folosit limbajul HTML și totodată am folosit și programul Bootstrap Studio. Acesta este o aplicație web de design și dezvoltare web. Acesta oferă un număr mare de componente pentru crearea de pagini cu răspuns, inclusiv anteturi, subsoluri, galerii și prezentări de diapozitive.

Pentru a putea utiliza funcțiile acestui program, în Eclipse am declarat:

```
13 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-18me44u0q781YhFLdVkuhf7AU6auU8tT94wHftj0brCEXSU1oBaayL2Qn" crossorigin="anonymous">
14 <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
15 <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Acne&display=swap">
16 <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Arnonim&display=swap">
17 <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Allura&display=swap">
18 <link rel="stylesheet" href="assets/css/styles.css">
19 <link rel="stylesheet" href="style.css">
20 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/3.5.2/animate.min.css">
```

În același timp, am utilizat diferite animații pentru background, folosind coduri extrase din diferite programe, coduri pe care le-am adaptat pentru aplicația mea. Un exemplu este:

```
178 rcles{
179   position: absolute;
180   top: 0;
181   left: 0;
182   width: 100%;
183   height: 100%;
184   overflow: hidden;
185
186
187   rcles div{
188     position: absolute;
189
190     list-style: none;
191     width: 20px;
192     height: 20px;
193     background: rgba(255, 255, 255, 0.2);
194     animation: animate 25s linear infinite;
195     bottom: -150px;
196
197
198
199   rcles div:nth-child(1){
200     left: 25%;
201     width: 80px;
202     height: 80px;
203     animation-delay: 0s;
204
205
206
207   rcles div:nth-child(2){
208     left: 10%;
209     width: 20px;
210     height: 20px;
211     animation-delay: 2s;
212     animation-duration: 12s;
213
214
215   rcles div:nth-child(3){
216     left: 70%;
217     width: 20px;
218     height: 20px;
219     animation-delay: 4s;
220
```

```
237   rcles div:nth-child(6){
238     left: 75%;
239     width: 110px;
240     height: 110px;
241     animation-delay: 3s;
242
243
244   rcles div:nth-child(7){
245     left: 35%;
246     width: 150px;
247     height: 150px;
248     animation-delay: 7s;
249
250
251   rcles div:nth-child(8){
252     left: 50%;
253     width: 25px;
254     height: 25px;
255     animation-delay: 15s;
256     animation-duration: 45s;
257
258
259   rcles div:nth-child(9){
260     left: 20%;
261     width: 15px;
262     height: 15px;
263     animation-delay: 2s;
264     animation-duration: 35s;
265
266
267   rcles div:nth-child(10){
268     left: 85%;
269     width: 150px;
270     height: 150px;
271     animation-delay: 0s;
272     animation-duration: 11s;
273
274
275
276
277   yframes animate {
278
279     0%{
280       transform: translateY(0) rotate(0deg);
281       opacity: 1;
282       border-radius: 0;
283     }
284
```

6) Concluzii

În prezent bazele de date sunt utilizate practic peste tot, fiecare companie, întreprindere mai ales cu caracter de producere, comercializare are nevoie și siguranța implementează în sistemul lor o bază de date. Pentru o utilizare fiabilă și corectă a unei baze de date este important să se urmărească realizarea unei arhitecturi ce oferă posibilitatea de a separa funcționalitățile, și anume partea de back-end de partea de front-end.

Totodată, aplicația realizată în tehnologia Hibernate și prezentată în cadrul acestui proiect, numită “Sound Archives”, permite utilizatorului să efectueze operațiile cerute pe baza de date creată în MySQL: afișare, adăugare, modificare și ștergere.

7) Bibliografie

- <https://www.nav.ro/blog/ce-este-mysql/> [1]
- <https://ro.wikipedia.org/> [2]
- <https://ro.education-wiki.com/> [3]
- <https://web.ceiti.md/lesson.php?id=1> [4]
- Cursurile PIBD [5]
- <https://ftp.utcluj.ro/pub> [6]