UNIVERSITATEA „POLITEHNICA” BUCURESTI

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

PROIECT – Programarea interfetelor pentru baze de date

2020-2021

Documentatie pentru tehnologia Hibernate

Student: Dobrin Cosmin-Iulian

Grupa: 432G

Profesor indrumator: Pupezescu Valentin

Cuprins:

1. Tema proiectului

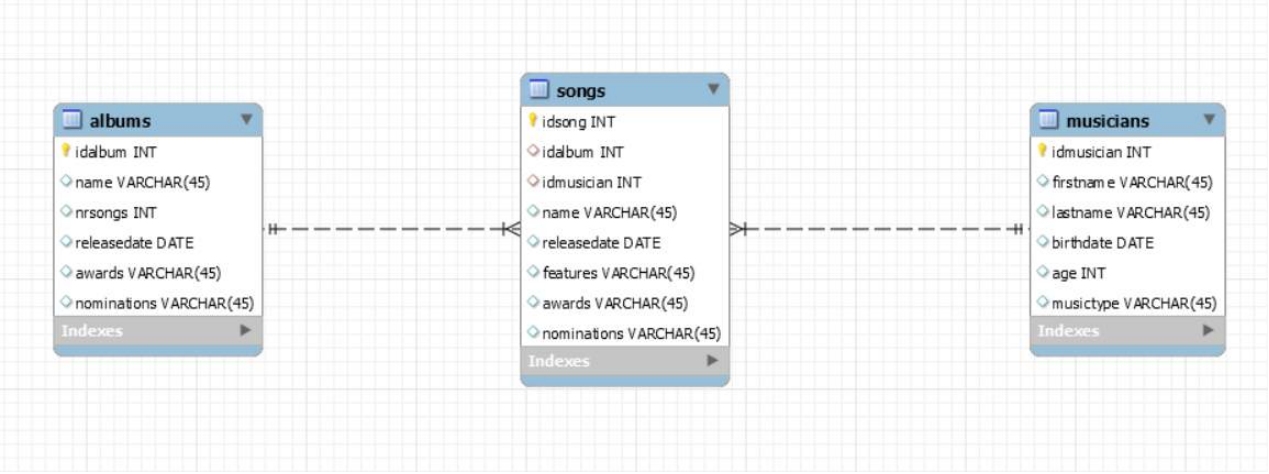
Proiectul are la baza o baza de date folosind structura de relationare M:N intre tabelele „albums” si „musicians”. Folosind aceste tabele, impreuna cu tabela de legatura „songs”, a fost creata baza de date „music”, care formeaza fundatia aplicatiei web „Music_Hibernate”. In aceasta aplicatie se pot adauga albume, muzicieni sau melodii, impreuna cu alte informatii despre acestea. De asemenea, se pot sterge sau modifica date despre acestea.

2. Tehnologia folosita

Aceasta aplicatie a fost dezvoltata cu ajutorul framework-ului Hibernate[0] care permite efectuarea operatiilor de baza CRUD[1] pe toate tabelele bazei de date. Aplicatia a fost construita in Eclipse[2], iar baza de date a fost dezvoltata in sistemul de gestiune a bazelor de date MySql[3].

3. Baza de date

Diagrama logica a bazei de date „music” este urmatoarea:



Tabelele sunt urmatoarele:

* **albums** (idalbum, name, nrsongs, releasedate, awards, nominations);

* **musicians** (idmusician, firstname, lastname, birthdate, age, musictype);

* **songs** (idsong, idalbum, idmusician, name, releasedate, features, awards, nominations);

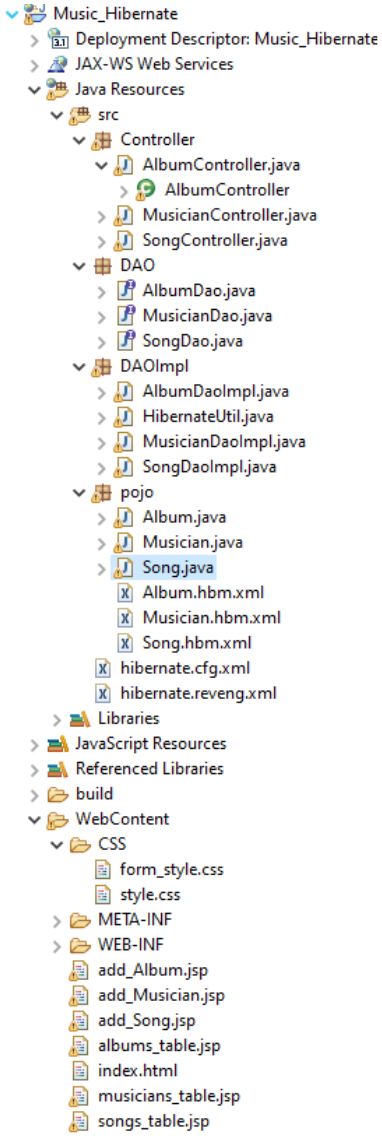
Intre tabelele *albums* si *musicians* se realizeaza asocierea M:N prin intermediul tabelei de legatura *songs*. Astfel, in aceasta tabela vom avea doua chei straine(FK): *idalbum* si *idmusician*. De asemenea, toate id-urile in baza de date au acelasi tip de date(INT), campurile *idalbum*(din tabela *albums*), *idsong* (din tabela *songs*) si *idmusician*(din tabela *musician*) sunt chei primare, iar in tabela *songs* avem doua chei straine la care s-a setat optiunea CASCADE la stergere.

4. Alegerea tehnologiei:

Avem nevoie de o tehnologie care sa rezolve problema conectarii limbajului obiect-orientat cu limbajul relational, intrucat lucrul cu ambele tipuri de software devine complicat si costisitor. Astfel, o idee buna ar fi sa folosim procesul de mapare obiectual-relationala, cunoscuta si sub denumirea de *object-relational mapping* sau *ORM*[4]. Asadar, folosind acest model, putem sa gestionam datele prin obiectele Java[5], reducand astfel decalajul intre modelul relational si cel orientat pe obiecte. Alegem tehnologia Hibernate[0] deoarece aceasta ne ofera un cadru de lucru in care folosim modelul obiectual-relational si de asemenea, este una din tehnologiile studiate la curs.

5. Segmente de cod insotite de explicatii

Structura proiectului este urmatoarea:



Aplicatia are la baza o arhitectura de tip MVC(Model-View-Controller)[6]. Astfel, codul ce tine de interfata este separat de codul ce tine de logica si procesare a datelor. Componenta „Model” este alcatuita din continutul pachetelor „pojo”, „DAO” si „DAOImpl”.

Pachetul „pojo”[8] este format din trei clase java si trei fisiere XML[7]. Fiecare clasa java reprezinta cate o tabela din baza de date. Coloanele tabelelor sunt inlocuite de variabile avand un tip de data corespunzator. Clasele sunt construite in aceeasi maniera, toate prezentand un constructor cu parametri, un constructor fara parametri, metode de tip „getter” si „setter” pentru fiecare variabila. O astfel de clasa poate fi observata mai jos:

```
package pojo;

import java.util.Date;

public class Song implements java.io.Serializable {

	private Integer idsong;
	private Musician musician;
	private Album album;
	private String name;
	private Date releasedate;
	private String features;
	private String awards;
	private String nominations;

	public Song() {}

	public Song(Musician musician, Album album, String name, Date releasedate, String features,
			String awards, String nominations) {
		this.musician = musician;
		this.album = album;
		this.name = name;
		this.releasedate = releasedate;
		this.features = features;
		this.awards = awards;
		this.nominations = nominations;
	}

	public Integer getIdsong() {
		return this.idsong;
	}
	public void setIdsong(Integer idsong) {
		this.idsong = idsong;
	}

	public Musician getMusicians() {
		return this.musician;
	}
	public void setMusicians(Musician musician) {
		this.musician = musician;
	}

	public Album getAlbums() {
		return this.album;
	}
	public void setAlbums(Album album) {
		this.album = album;
	}

	public String getName() {
		return this.name;
	}
	public void setName(String name) {
		this.name = name;
```

```
        }

        public Date getReleasedate() {
                return this.releasedate;
        }
        public void setReleasedate(Date releasedate) {
                this.releasedate = releasedate;
        }

        public String getFeatures() {
                return this.features;
        }
        public void setFeatures(String features) {
                this.features = features;
        }

        public String getAwards() {
                return this.awards;
        }
        public void setAwards(String awards) {
                this.awards = awards;
        }

        public String getNominations() {
                return this.nominations;
        }
        public void setNominations(String nominations) {
                this.nominations = nominations;
        }
}
```

Fisierele XML[7] au rol de mapare, indicand astfel cum se face legatura intre una din clasele definite anterior si tabela corespunzatoare din baza de date. Pentru fiecare clasa de tip „pojo”[8] exista un fisier „.hbm.xml” care realizeaza maparea intre clasa si tabela. Un astfel de fisier este atasat mai jos:

```
<?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.Song" table="songs" catalog="music" optimistic-lock="version">
        <id name="idsong" type="java.lang.Integer">
            <column name="idsong" />
            <generator class="identity" />
        </id>
        <many-to-one name="albums" class="pojo.Album" fetch="select">
            <column name="idalbum" />
        </many-to-one>
        <many-to-one name="musicians" class="pojo.Musician" fetch="select">
            <column name="idmusician" />
        </many-to-one>
        <property name="name" type="string">
            <column name="name" length="45" />
        </property>
        <property name="releasedate" type="date">
            <column name="releasedate" length="10" />
        </property>
        <property name="features" type="string">
            <column name="features" length="45" />
        </property>
        <property name="awards" type="string">
            <column name="awards" length="45" />
        </property>
        <property name="nominations" type="string">
            <column name="nominations" length="45" />
```

```
        </property>
    </class>
</hibernate-mapping>
```

Pachetul „DAO” este format din trei interfete, scopul sau fiind de a ajuta la standardizarea proiectului. Implementarea acestor interfete se va gasi in pachetul „DAOImpl”. Interfetele sunt construite in aceeasi maniera, fiecare avand patru metode corespunzatoare operatiunilor care pot fi facute pentru fiecare obiect: adaugare, afisare, modificare si stergere. Denumirea de DAO se trage de la „Data acces object”[9], un model care separa stratul de date din clasele „pojo” de stratul ce tine de persistenta, mai exact baza noastra de date. O astfel de interfata este prezentata mai jos:

```
package DAO;


import java.util.List;

import pojo.Song;

import java.util.Date;

import pojo.Musician;

import pojo.Album;


public interface SongDao {


        public void addSong(Song song);

        public List<Song> showSongs();

        public void modifySong(Integer idsong, Album album,

                                                Musician musician, String name,

                                                Date releasedate, String features,

                                                String awards, String nominations);

        public void deleteSong(Song song);

}
```

Pachetul „DAOImpl” contine implementarile interfetelor din pachetul „DAO” si in plus, o clasa numita „HibernateUtil”. Din nou, toate clasele ce tin de implementarea interfetelor sunt construite in mod asemanator, continand definitiile celor patru metode declarate anterior. Prima metoda este cea de adaugare:

```
public void addSong(Song song) {
                Session session = HibernateUtil.getSessionFactory().openSession();
                Transaction transaction = session.beginTransaction();
                session.save(song);
                transaction.commit();
                session.close();
        }
```

Aceasta metoda este folosita pentru a adauga un obiect de tip „Song”, preluat de la clasa „SongController”, si a il introduce in baza de date. Pentru a se realiza acest lucru se va genera o sesiune prin clasa de utilitate „HibernateUtil”, folosind metoda „getSessionFactory()”, urmand sa o deschidem folosind metoda „openSession”. Sesiunea reprezinta un mijloc de comunicare cu baza de date. Dupa deschiderea sesiunii, urmeaza creerea si inceperea unei tranzactii, folosind sesiunea anterior creata: „session.beginTransaction()”. Tranzactia reprezinta un set de instructiuni care trebuie executate in mod atomic. Altfel spus, fie se executa toate instructiunile, fie niciuna. Folosind sesiunea anterior creata salvam obiectul de tip „Song” primit ca parametru: „session.save(song)”. In urma salvarii obiectului, nu ne ramane decat sa apelam metoda „commit()” pentru a finaliza tranzactia si sa incheiem sesiunea: „session.close()”.

A doua metoda este cea de afisare:

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

Aceasta metoda este folosita pentru a afisa datele din tabela. Pentru inceput se creeaza o lista de obiecte de tip „Song”, lista ce va fi returnata in final de catre metoda. Apoi, similar ca in metoda de adaugare, se deschide o sesiune si se incepe o tranzactie. Se creeaza o interogare „org.hibernate.query.Query” si o initializam prin sesiunea anterior creata, apeland metoda „createQuery(“From Song”)”. Parametrul dat functiei „createQuery()” este scris folosind limbajul „HQL” sau „Hibernate Query Language”, un limbaj similar „SQL” cu ajutorul caruia putem face interogari. Datele preluate din baza de date vor fi transmise sub forma de lista obiectului de tip „List<Song>” creat anterior. In final, vom incheia tranzactia si vom inchide sesiunea, similar metodei de adaugare care a fost descrisa mai sus.

A treia metoda este cea de modificare:

```
public void modifySong(Integer idsong, Album album, Musician musician,
                        String name, Date releasedate, String features, String awards, String nominations)
{

                Session session = HibernateUtil.getSessionFactory().openSession();
                Transaction transaction = session.beginTransaction();
                Song songDetails;
                songDetails = (Song) session.load(Song.class, idsong);
                songDetails.setAlbums(album);
                songDetails.setMusicians(musician);
                songDetails.setName(name);
                songDetails.setReleasedate(releasedate);
                songDetails.setFeatures(features);
                songDetails.setAwards(awards);
                songDetails.setNominations(nominations);
                session.update(songDetails);
                transaction.commit();
                session.close();
        }
```

Aceasta metoda este folosita pentru a modifica datele din tabela. Similar metodelor precedente, se deschide o sesiune si se initializeaza o tranzactie. Se creeaza un obiect de tip „Song” si este mai apoi initializat cu obiectul al carui Id este primit ca parametru, prin apelarea metodei „session.load(Song.class, idsong)”. Folosind metodele de tip „setter” din clasa „Song” vom atribui noile valori pe care utilizatorul doreste sa le dea variabilelor obiectului cu acel Id. Apeland metoda „session.update(songDetails)” se vor face schimbarile la nivelul bazei de date. Similar metodelor precedente, se incheie tranzactia si se inchide sesiunea.

Ultima metoda este cea de stergere:

```
public void deleteSong(Song song) {
                Session session = HibernateUtil.getSessionFactory().openSession();
                Transaction transaction = session.beginTransaction();
                session.delete(song);
                transaction.commit();
                session.close();
        }
```

Aceasta metoda este folosita pentru a sterge datele din tabela. Similar metodelor precedente, se deschide o sesiune si se initializeaza o tranzactie. Folosind sesiunea anterior creata stergem obiectul de tip „Song” primit ca parametru: „session.delete(song)”. In urma stergerii obiectului, nu ne ramane decat sa apelam metoda „commit()” pentru a finaliza tranzactia si sa incheiem sesiunea: „session.close()”. Putem observa o asemanare puternica intre metoda de adaugare si cea de stergere, singura diferenta din corpul functiei fiind metoda apelata prin sesiune: „session.save(song)” vs. „session.delete(song)”.

Singura clasa din acest pachet care nu este construita in maniera prezentata anterior, este clasa „HibernateUtil”:

```
package DAOImpl;


import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.SessionFactory;




public class HibernateUtil {


private static final SessionFactory sessionFactory;


    static {
```

```
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder =
                        new StandardServiceRegistryBuilder().applySettings(configuration.getProperties());
        sessionFactory = configuration.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;
    }
}
```

Pe scrut, aceasta clasa are rolul de a oferi o modalitate convenabila de a genera sesiuni prin metoda statica „getSessionFactory()”, „ascunzand” configurarea din spatele generarii unei sesiuni.

Pachetul Controller este format din 3 clase java, avand rolul de a separa componenta „Model” de componenta „View”, manipuland datele din „Model” si actualizand interfata. Toate cele trei clase sunt construite in mod similar. Astfel, in fiecare clasa avem doua metode importante care raspund in momentul in care utilizatorul lanseaza cereri dinamice catre server.

Prima metoda este cea folosita la adaugarea de noi date in baza de date:

```
@Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {

                if (request.getParameter("addSong") != null) {

                        int IAE = 0;
                        Integer idalbum = java.lang.Integer.parseInt(request.getParameter("idalbum"));
                        Integer idmusician =
java.lang.Integer.parseInt(request.getParameter("idmusician"));
                        Session session = HibernateUtil.getSessionFactory().openSession();
                        Album album = (Album) session.get(Album.class, idalbum);
                        Musician musician = (Musician) session.get(Musician.class, idmusician);
                        Date releasedate = null;
                        try {
```

```
                releasedate = Date.valueOf(request.getParameter("releasedate"));
            } catch(IllegalArgumentException e) {
                e.printStackTrace();
                IAE = 1;
            }

            if (IAE == 0) {
                String name = request.getParameter("name");
                String features = request.getParameter("features");
                String awards = request.getParameter("awards");
                String nominations = request.getParameter("nominations");

                song.setAlbums(album);
                song.setMusicians(musician);
                song.setName(name);
                song.setReleasedate(releasedate);
                song.setFeatures(features);
                song.setAwards(awards);
                song.setNominations(nominations);
                songDaoImpl.addSong(song);
            }

            RequestDispatcher rd = request.getRequestDispatcher("add_Song.jsp");
            rd.forward(request, response);
        }
    }
```

Aceasta metoda comunica cu fisierul „add_Song.jsp”, de unde primeste datele introduse de catre utilizator. Folosind metoda „request.getParameter()” se extrag datele introduse din fiecare camp, sub forma de sir de caractere, urmand sa fie folosite pentru initializarea variabilelor (care reprezinta coloanele din tabela corespunzatoare clasei in care se lucreaza), dupa conversia corespunzatoare. Variabila „IAE” joaca rol de fanion si ajuta la tratarea exceptiei „IllegalArgumentException”. Acest lucru este posibil prin folosirea expresiei „try & catch”, unde se incearca atribuirea unei valori introduse de utilizator la un atribut cu format fix. In cazul in care formatul nu este cel corespunzator, va fi „aruncata” o exceptie, caz in care fanionul va fi setat (IAE = 1). In cazul in care fanionul nu va fi setat, se vor face in continuare restul initializarilor si se vor folosi apoi metode de tip „setter” pentru a modela obiectul de tip „Song” dupa cerintele utilizatorului. Acest obiect va fi mai apoi trimis ca si parametru metodei „addSong()” accesata printr-un obiect de tip „SongDaoImpl”. Aceasta metoda va adauga datele oferite de obiect in baza de date. In final, folosim un „RequestDispatcher” si metoda „forward()” pentru a naviga mai departe la pagina „add_Song.jsp”.

A doua metoda este folosita pentru 3 tipuri de operatii: afisare, modificare, stergere. Fata de prima metoda „doGet()”, aceasta nu manipuleaza doar baza de date, ci si interfata. Metoda este urmatoarea:

```
@Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        if (request.getParameter("showSongs") != null) {
            List<Song> songList = new ArrayList();
            songList = songDaoImpl.showSongs();
            request.setAttribute("songList", songList);
```

```
                RequestDispatcher rd = request.getRequestDispatcher("songs_table.jsp");
                rd.forward(request, response);
        }

        if (request.getParameter("modifySong") != null) {
                Integer id1 = Integer.parseInt(request.getParameter("idsong"));
                Integer idalbum = Integer.parseInt(request.getParameter("idalbum"));
                Integer idmusician = Integer.parseInt(request.getParameter("idmusician"));
                Session session = HibernateUtil.getSessionFactory().openSession();
                Album album = (Album) session.get(Album.class, idalbum);
                Musician musician = (Musician) session.get(Musician.class, idmusician);
                String name = request.getParameter("name");
                String features = request.getParameter("features");
                String awards = request.getParameter("awards");
                String nominations = request.getParameter("nominations");

                int IAE = 0;
                Date releasedate = null;
                try {
                        releasedate = Date.valueOf(request.getParameter("releasedate"));
                } catch (IllegalArgumentException e) {
                        e.printStackTrace();
                        IAE = 1;
                }

                if (IAE == 0) {
                        songDaoImpl.modifySong(id1, album, musician, name, releasedate, features,
                                        awards, nominations);
                }

                RequestDispatcher rd = request.getRequestDispatcher("add_Song.jsp");
                rd.forward(request, response);

        }

        if (request.getParameter("deleteSong") != null) {
                Integer id2 = Integer.parseInt(request.getParameter("idsong"));
                song.setIdsong(id2);
                songDaoImpl.deleteSong(song);
                RequestDispatcher rd = request.getRequestDispatcher("add_Song.jsp");
                rd.forward(request, response);
        }
}
```

Aceasta metoda comunica cu fisierul „add_Song.jsp”, pentru afisarea tabelei, dar si cu fisierul „songs_table.jsp” pentru eventuale modificari sau stergeri. Metoda „doPost()” este formata din trei afirmatii de tip „if (){}”, fiecare ocupandu-se de o operatie, in functie de cererea trimisa de fisierul „.jsp”.

Prima afirmatie manipuleaza cererile de afisare:

```
if (request.getParameter("showSongs") != null) {
                List<Song> songList = new ArrayList();
                songList = songDaoImpl.showSongs();
                request.setAttribute("songList", songList);
                RequestDispatcher rd = request.getRequestDispatcher("songs_table.jsp");
                rd.forward(request, response);
        }
```

Este creata o lista de obiecte de tip „Song” careia ii este mai apoi atribuita o lista cu datele actuale din baza de date, folosind metoda „showSongs” disponibila prin componenta „DAO”. Lista creata o vom trimite ca parametru metodei „setAttribute()” pentru a putea fi actualizata interfata. In final,

folosim un „RequestDispatcher” si metoda „forward()” pentru a naviga mai departe la pagina „songs_table.jsp”.

A doua afirmatie are rol in operatiile de modificare:

```
if (request.getParameter("modifySong") != null) {
                        Integer id1 = Integer.parseInt(request.getParameter("idsong"));
                        Integer idalbum = Integer.parseInt(request.getParameter("idalbum"));
                        Integer idmusician = Integer.parseInt(request.getParameter("idmusician"));
                        Session session = HibernateUtil.getSessionFactory().openSession();
                        Album album = (Album) session.get(Album.class, idalbum);
                        Musician musician = (Musician) session.get(Musician.class, idmusician);
                        String name = request.getParameter("name");
                        String features = request.getParameter("features");
                        String awards = request.getParameter("awards");
                        String nominations = request.getParameter("nominations");

                        int IAE = 0;
                        Date releasedate = null;
                        try {
                                releasedate = Date.valueOf(request.getParameter("releasedate"));
                        } catch (IllegalArgumentException e) {
                                e.printStackTrace();
                                IAE = 1;
                        }

                        if (IAE == 0) {
                                songDaoImpl.modifySong(id1, album, musician, name, releasedate, features,
                                                awards, nominations);
                        }

                        RequestDispatcher rd = request.getRequestDispatcher("add_Song.jsp");
                        rd.forward(request, response);

                }
```

Aceasta afirmatie este similara cu cea din metoda „doGet()”, singura diferenta fiind reprezentata de inlocuirea metodei „addSong()” cu metoda „modifySong()”. De aceasta data nu se mai folosesc metode de tip „setter” pentru modelarea obiectului, intrucat acesta deja exista. Tot ce ne ramane de facut este sa trimitem variabilele initializate cu noile valori dorite de utilizator ca si parametrii ai metodei „modifySong()”, care se va ocupa de modificarea obiectului de tip „Song”.

A treia afirmatie are rol in operatiile de stergere:

```
if (request.getParameter("deleteSong") != null) {
                        Integer id2 = Integer.parseInt(request.getParameter("idsong"));
                        song.setIdsong(id2);
                        songDaoImpl.deleteSong(song);
                        RequestDispatcher rd = request.getRequestDispatcher("add_Song.jsp");
                        rd.forward(request, response);
                }
```

Se creeaza o variabila de tip „Integer” pentru stocarea id-ului obiectului de tip „Song” pe care utilizatorul doreste sa il stearga. Obiectul de tip „Song” primeste id-ul printr-o metoda de tip „setter” urmand ca mai apoi sa fie trimis si el ca parametru in metoda „deleteSong()”. Aceasta metoda va realiza stergerea datelor din baza de date. . In final, folosim un „RequestDispatcher” si metoda „forward()” pentru a naviga mai departe la pagina „songs_table.jsp”.

Pe langa aceste doua metode importante, in fiecare clasa exista o metoda menita sa returneze o scruta descriere pentru servlet:

```
@Override
        public String getServletInfo() {
                return "Short description";
        }
```

Pe langa aceste patru pachete, mai pot fi observate doua fisiere de tip „.xml” cu rol in configurare:

„hibernate.cfg.xml”:

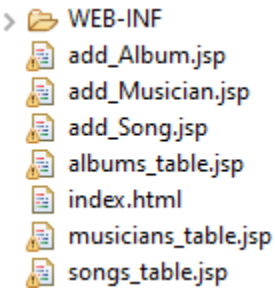
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
      <property name="hibernate.hbm2ddl.auto">update</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</pro
perty>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/music</property>
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="hibernate.connection.username">dbmusicdb</property>
    <property name="hibernate.connection.password">CosminPass1234.</property>
    <mapping resource="pojo/Song.hbm.xml"/>
    <mapping resource="pojo/Album.hbm.xml"/>
    <mapping resource="pojo/Musician.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

„hibernate.reveng.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse Engineering DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-reverse-engineering-3.0.dtd">
<hibernate-reverse-engineering>
  <schema-selection match-catalog="music"/>
  <table-filter match-name="musicians"/>
  <table-filter match-name="albums"/>
  <table-filter match-name="songs"/>
</hibernate-reverse-engineering>
```

Aceste fisiere cuprind proprietati necesare pentru configurarea conexiunii cu baza de date precum: driver JDBC, nume de utilizator, parola. De asemenea, sunt mentionate fisierele necesare pentru mapare. De asemenea, in fisierul „hibernate.reveng.xml” se specifica schema si tabelele care sunt procesate.

In dosarul „WebContent” pe langa fisierul „index.html”, se pot observa cate trei fisiere „.jsp” pentru adaugare de noi date in tabela si trei fisiere „.jsp” pentru vizualizarea tabelelor si executarea operatiunilor de modificare sau stergere:



Denumirea „JSP” provine de la „Java server pages”. In astfel de fisiere vom gasi in mare parte o sintaxa HTML la care se adauga taguri JSP, care fac conexiunea la servlet.

Fisierul „add_Song.jsp” se prezinta astfel:

```
<%@page import="DAOImpl.MusicianDaoImpl"%>
<%@page import="DAOImpl.AlbumDaoImpl"%>
<%@page import="pojo.Musician"%>
<%@page import="java.util.ArrayList"%>
<%@page import="pojo.Album"%>
<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
    <link rel="stylesheet" href="CSS/form_style.css">
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Song</title>
        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    </head>
    <body>
<%
        AlbumDaoImpl albumDaoImpl = new AlbumDaoImpl();
        MusicianDaoImpl musicianDaoImpl = new MusicianDaoImpl();
        List<Album> albumList = new ArrayList();
        albumList = albumDaoImpl.showAlbums();
        List<Musician> musicianList = new ArrayList();
        musicianList = musicianDaoImpl.showMusicians();
        request.setAttribute("albumList", albumList);
        request.setAttribute("musicianList", musicianList);
%>
        <div id="add">
            <h1> Add a new song </h1>
            <form action="SongController" method="GET">
                <table align="center">
                    <tr>
                        <td> Album: </td>
                        <td>
                            <select name="idalbum">
                                <c:forEach items="${albumList}" var="albums">
                                    <option value="${albums.idalbum}">${albums.idalbum},
                                     ${albums.name}, ${albums.nrsongs}, ${albums.releasedate},
                                      ${albums.awards}, ${albums.nominations}</option>
                                </c:forEach>
                            </select>
                        </td>
                    </tr>
                    <tr>
                        <td> Musician: </td>
```

```
                        <td>
                            <select name="idmusician">
                                <c:forEach items="${musicianList}" var="musicians">
                                    <option value="${musicians.idmusician}">${musicians.idmusician},
                                     ${musicians.firstname}, ${musicians.lastname},
${musicians.birthdate},
                                      ${musicians.age}, ${musicians.musictype}</option>
                                </c:forEach>
                            </select>
                        </td>
                    </tr>
                    <tr>
                        <td> Name: </td>
                        <td><input type="text" name="name"></td>
                    </tr>
                    <tr>
                        <td> Release date: </td>
                        <td><input type="text" name="releasedate" placeholder="yyyy-mm-dd"></td>
                    </tr>
                    <tr>
                        <td> Features: </td>
                        <td><input type="text" name="features"></td>
                    </tr>
                     <tr>
                        <td> Awards: </td>
                        <td><input type="text" name="awards"></td>
                    </tr>
                     <tr>
                        <td> Nominations: </td>
                        <td><input type="text" name="nominations"></td>
                    </tr>

                    <tr>
                        <td><input type="submit" name="addSong" value="Add"></td>

                    </tr>
                </table>
            </form>
        </div>

        <form action="SongController" method="POST">
        <p align="center">
            <input type="submit" name="showSongs" value="Show">
            </p>
        </form>
        <br>
        <br>
        <br>
        <br>
        <p align="center">
        <a href="index.html"><b>Home</b></a>
        </p>
    </body>
</html>
```

Se observa comunicarea cu clasa de tip controller prin metodele „GET” si „POST”. Similar sunt construite si fisierele „add_Album.jsp” si „add_Musician.jsp”.

Fisierul „songs_table.jsp” este cel prin care se afiseaza tabela impreuna cu optiunile de modificare si stergere de date. Intrucat in acest context nu se adauga noi date in tabela, se foloseste doar metoda „POST” pentru a comunica cu clasa controller. Fisierul arata astfel:

```jsp
<%@page import="DAOImpl.MusicianDaoImpl"%>
<%@page import="DAOImpl.AlbumDaoImpl"%>
<%@page import="pojo.Musician"%>
<%@page import="java.util.ArrayList"%>
<%@page import="pojo.Album"%>
<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="CSS/style.css">
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Songs</title>
        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
        <script>
            $(document).ready(function () {
                $("#deleteSong").hide();
                $("#modifySong").hide();

                $('#update').click(function() {
                    $("#modifySong").toggle(this.checked);
                });

                $('#delete').click(function() {
                    $("#deleteSong").toggle(this.checked);
                });
            });
        </script>

    </head>
    <body>
        <%
            AlbumDaoImpl albumDaoImpl = new AlbumDaoImpl();
            MusicianDaoImpl musicianDaoImpl = new MusicianDaoImpl();
            List<Album> albumList = new ArrayList();
            albumList = albumDaoImpl.showAlbums();
            List<Musician> musicianList = new ArrayList();
            musicianList = musicianDaoImpl.showMusicians();
            request.setAttribute("albumList", albumList);
            request.setAttribute("musicianList", musicianList);
        %>
        <h1 align="center"> Songs:</h1>

        <table border="1" align="center">
        <thead>
            <tr>
                <th><b>IdSong:</b></th>
                <th><b>IdAlbum:</b></th>
                <th><b>Album name:</b></th>
                <th><b>Number of songs:</b></th>
                <th><b>Album release date:</b></th>
                <th><b>Album awards:</b></th>
                <th><b>Album nominations:</b></th>
                <th><b>IdMusician:</b></th>
                <th><b>First name:</b></th>
                <th><b>Last name:</b></th>
                <th><b>Birthdate:</b></th>
                <th><b>Age:</b></th>
                <th><b>Music type:</b></th>
                <th><b>Song name:</b></th>
                <th><b>Song release date:</b></th>
                <th><b>Song features:</b></th>
                <th><b>Song awards:</b></th>
                <th><b>Song nominations:</b></th>
            </tr>
            </thead>
            <tbody>
            <c:forEach var="songs" items="${songList}">
```

```
                <tr>
                    <td>${songs.idsong}</td>
                    <td>${songs.albums.idalbum}</td>
                    <td>${songs.albums.name}</td>
                    <td>${songs.albums.nrsongs}</td>
                    <td>${songs.albums.releasedate}</td>
                    <td>${songs.albums.awards}</td>
                    <td>${songs.albums.nominations}</td>
                    <td>${songs.musicians.idmusician}</td>
                    <td>${songs.musicians.firstname}</td>
                    <td>${songs.musicians.lastname}</td>
                    <td>${songs.musicians.birthdate}</td>
                    <td>${songs.musicians.age}</td>
                    <td>${songs.musicians.musictype}</td>
                    <td>${songs.name}</td>
                    <td>${songs.releasedate}</td>
                    <td>${songs.features}</td>
                    <td>${songs.awards}</td>
                    <td>${songs.nominations}</td>
                </tr>
            </c:forEach>
            </tbody>
        </table>
        <br>
        <p align="center">
            <a href="index.html" class="homeLink"><b>Home</b></a>
        </p>
        <form action="SongController" method="POST">
            <p align="center">
                Modify: <input type="checkbox" id="update">
                Delete: <input type="checkbox" id="delete"
                onclick="document.getElementById('idalbum').disabled = this.checked;
                         document.getElementById('idmusician').disabled = this.checked;
                         document.getElementById('name').disabled = this.checked;
                         document.getElementById('releasedate').disabled = this.checked;
                         document.getElementById('features').disabled = this.checked;
                         document.getElementById('awards').disabled = this.checked;
                         document.getElementById('nominations').disabled = this.checked;"><br><br>
                IdSong: <br>
                <select name="idsong">
                    <c:forEach items="${songList}" var="songs">
                        <option value="${songs.idsong}">${songs.idsong}</option>
                    </c:forEach>
                </select>
                <br>
                IdAlbum: <br>
                <select name="idalbum">
                    <c:forEach items="${albumList}" var="albums">
                        <option value="${albums.idalbum}">${albums.idalbum}, ${albums.name},
                         ${albums.nrsongs}, ${albums.releasedate}, ${albums.awards},
${albums.nominations}</option>
                    </c:forEach>
                </select>
                <br>
                IdMusician: <br>
                <select name="idmusician">
                    <c:forEach items="${musicianList}" var="musicians">
                        <option value="${musicians.idmusician}">${musicians.idmusician},
                         ${musicians.firstname}, ${musicians.lastname}, ${musicians.birthdate},
                          ${musicians.age}, ${musicians.musictype}</option>
                    </c:forEach>
                </select>
                <br>
                Modify song name: <br><input id="name" type="text" name="name"><br>
                Modify song release date: <br><input id="releasedate" type="text" name="releasedate"
placeholder="yyyy-mm-dd"> <br>
                Modify song features: <br><input id="features" type="text" name="features"> <br>
                Modify song awards: <br><input id="awards" type="text" name="awards"> <br>
                Modify song nominations: <br><input id="nominations" type="text" name="nominations"> <br>
```

```
                <button type="submit" id="modifySong" name="modifySong"> Modify</button> <br> <br>
                <button type="submit" id="deleteSong" name="deleteSong"> Delete </button>
            </p>
        </form>

    </body>
</html>
```

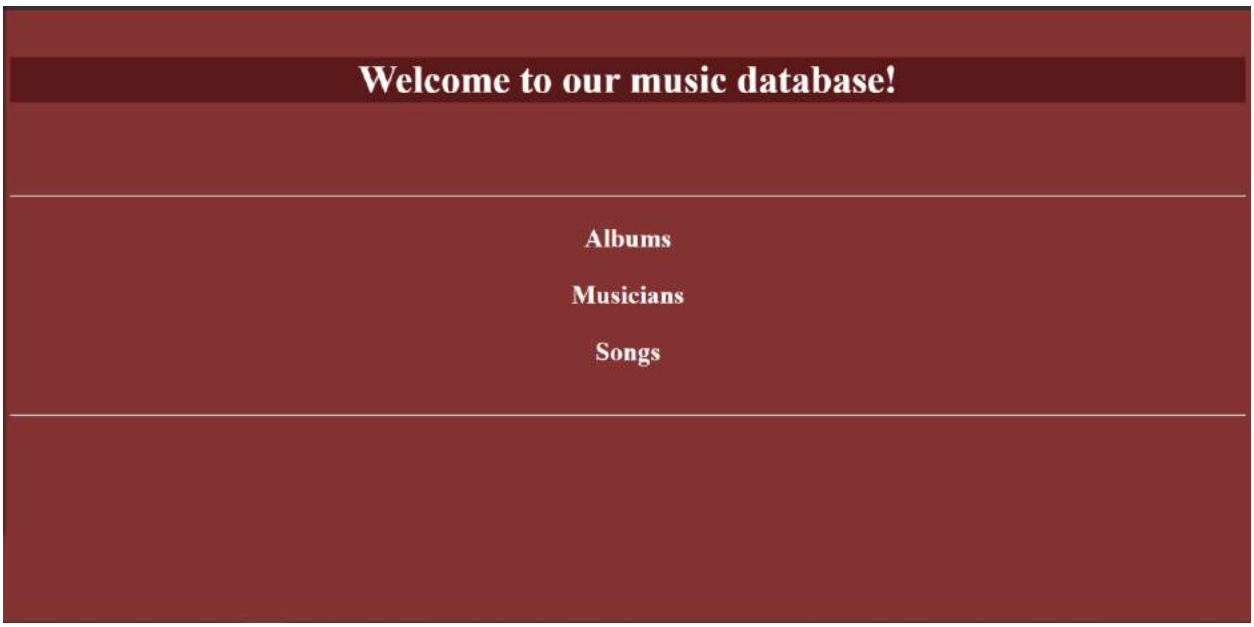
De mentionat mai este sectiunea urmatoare:

```
<script>
            $(document).ready(function () {
                $("#deleteSong").hide();
                $("#modifySong").hide();

                $('#update').click(function() {
                    $("#modifySong").toggle(this.checked);
                });

                $('#delete').click(function() {
                    $("#deleteSong").toggle(this.checked);
                });
            });
        </script>
```

In acest caz se foloseste limbajul JavaScript, mai exact libraria JQuery, pentru a manipula vizibilitatea butoanelor de modificare/stergere, in functie de checkbox-ul care este bifat.

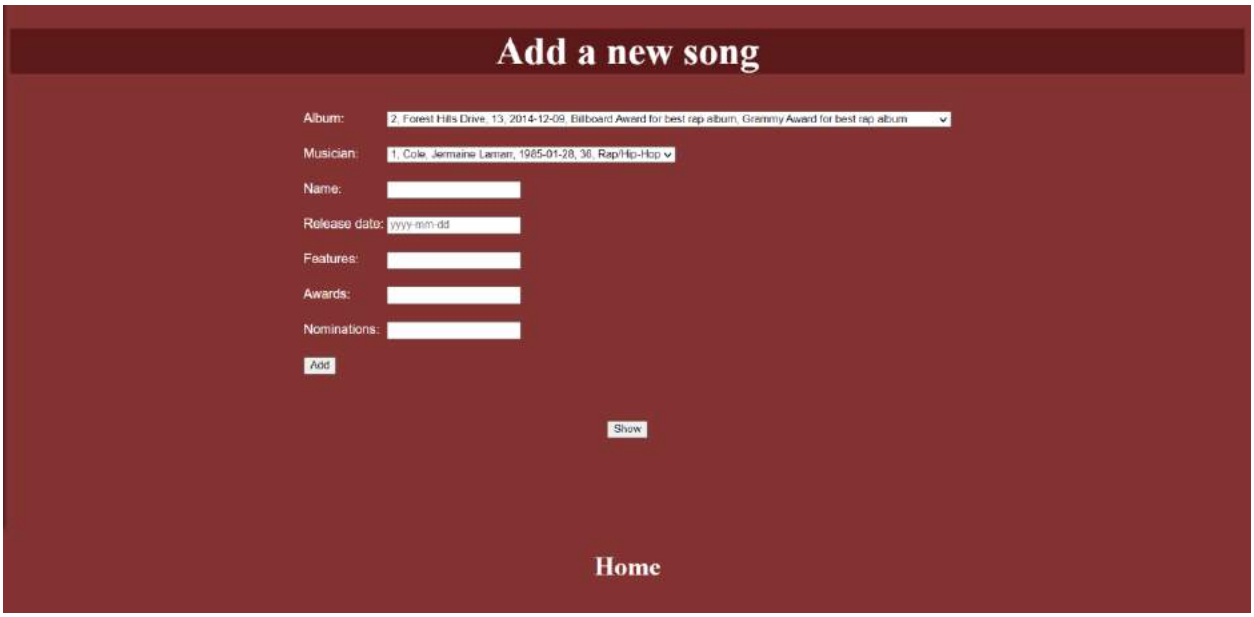
6. Folosirea aplicatiei

Aplicatia este deschisa prin rularea fisierului „index.html”. Astfel urmatoarea pagina ar trebui sa fie deschisa intr-un browser de internet:



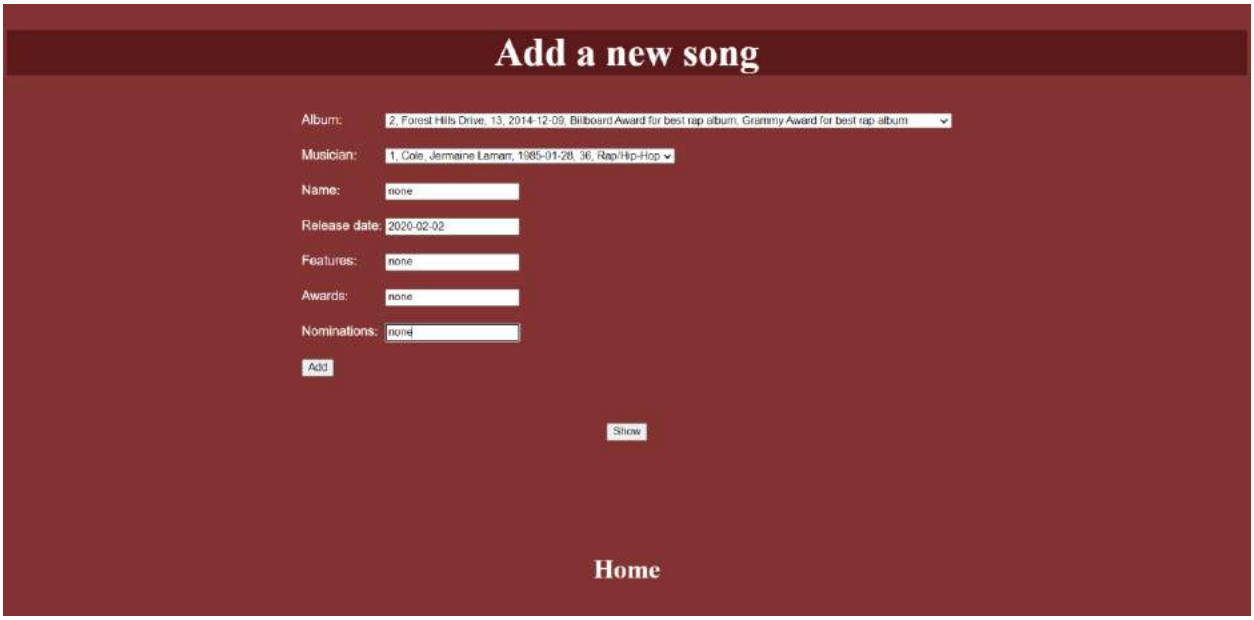
Aceasta pagina prezinta un meniu din care se poate naviga catre trei formulare pentru adaugarea unui nou album, muzician sau cantec.

Printr-un click pe link-ul dorit, spre exemplu „songs”, se ajunge la formularul urmator:

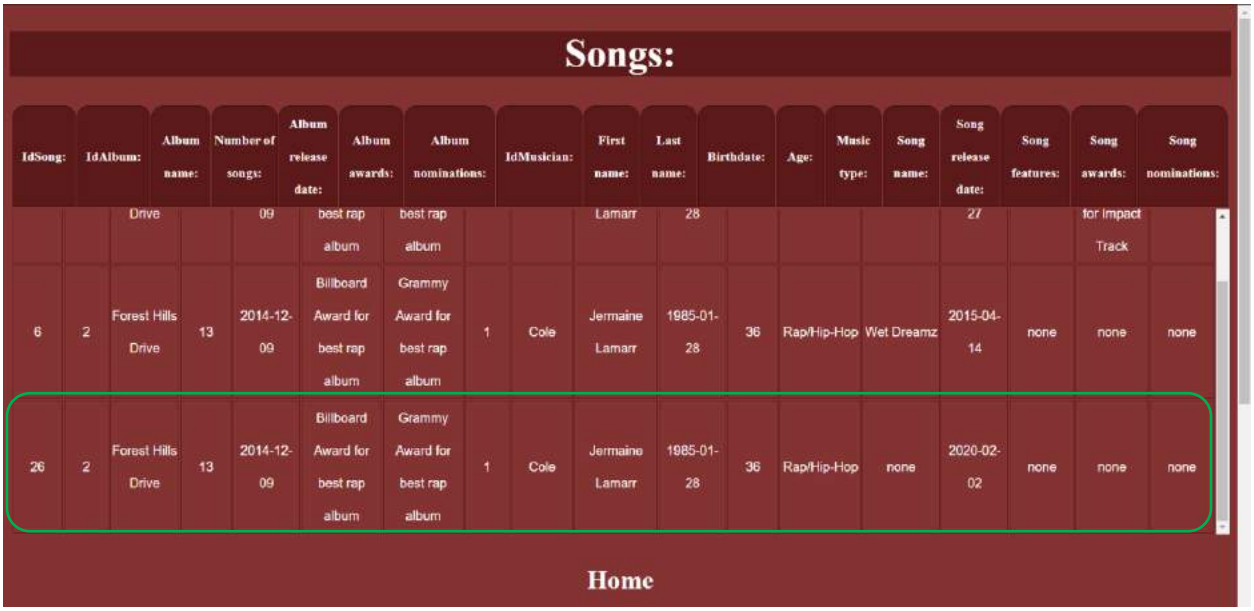


Din acest punct, se poate naviga catre o tabela in care sunt prezentate toate cantecele din baza de date. De asemenea, se poate completa formularul si dupa apasarea butonului „Add”, cantecul va fi adaugat in baza de date si in tabela.

Formularul completat:



Apasarea butonului etichetat „Show”, va duce la afisarea urmatoarei pagini, unde se poate observa cantecul adaugat in pasul anterior:

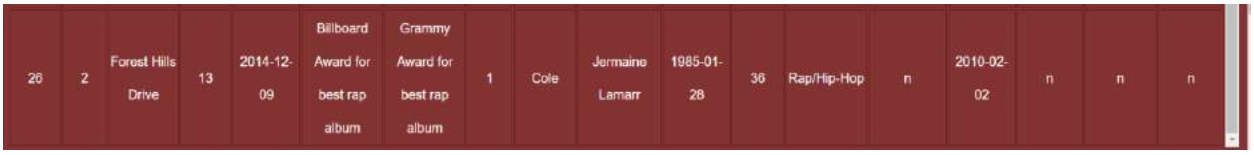


IdSong:	IdAlbum:	Album name:	Number of songs:	Album release date:	Album awards:	Album nominations:	IdMusician:	First name:	Last name:	Birthdate:	Age:	Music type:	Song name:	Song release date:	Song features:	Song awards:	Song nominations:
		Drive		09	best rap album	best rap album		Lamarr	28					27		for Impact Track	
6	2	Forest Hills Drive	13	2014-12-09	Billboard Award for best rap album	Grammy Award for best rap album	1	Cole	Jermaine Lamarr	1985-01-28	36	Rap/Hip-Hop	Wet Dreamz	2015-04-14	none	none	none
26	2	Forest Hills Drive	13	2014-12-09	Billboard Award for best rap album	Grammy Award for best rap album	1	Cole	Jermaine Lamarr	1985-01-28	36	Rap/Hip-Hop	none	2020-02-02	none	none	none

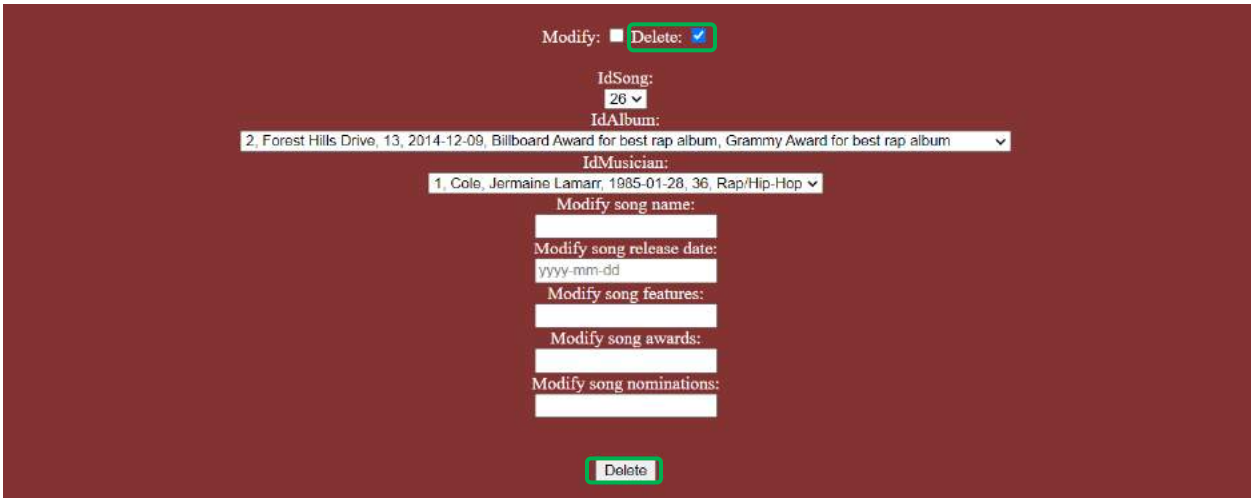
De asemenea, daca se navigheaza in partea de jos a paginii, se poate observa link-ul de intoarcere in meniul principal, optiunile de modificare („Modify”) sau de stergere („Delete”), cat si un formular ce va fi completat doar in cazul in care se doreste modificarea unui cantec.

Pentru modificarea cantecului tocmai adaugat, se selecteaza id-ul cantecului respectiv si se completeaza campurile cu noile valori. Se bifeaza optiunea „Modify” si apoi se apasa pe butonul „Modify”:

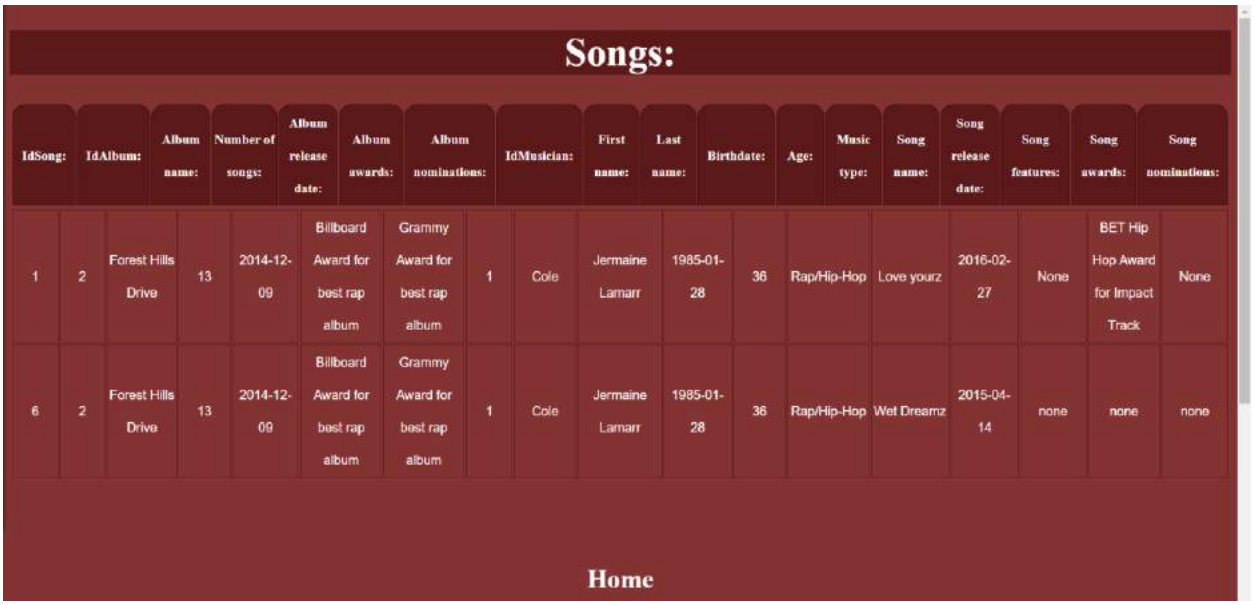
Se va naviga automat la pagina de adaugare a unui nou cantec. Daca se apasa din nou butonul „Show”, se vor observa in tabela modificarile facute:



Pentru stergerea acestui cantec, se bifeaza optiunea „Delete”, se selecteaza id-ul cantecului si se apasa butonul „Delete”:



Din nou, se va naviga automat catre pagina de adaugare a unui nou cantec. Pentru a vizualiza schimbarile, se apasa din nou pe butonul „Show”. Se poate observa ca acel cantec a fost sters din tabela si din baza de date:



Songs:

IdSong:	IdAlbum:	Album name:	Number of songs:	Album release date:	Album awards:	Album nominations:	IdMusician:	First name:	Last name:	Birthdate:	Age:	Music type:	Song name:	Song release date:	Song features:	Song awards:	Song nominations:
1	2	Forest Hills Drive	13	2014-12-09	Billboard Award for best rap album	Grammy Award for best rap album	1	Cole	Jermaine Lamarr	1985-01-28	36	Rap/Hip-Hop	Love yourz	2016-02-27	None	BET Hip Hop Award for Impact Track	None
6	2	Forest Hills Drive	13	2014-12-09	Billboard Award for best rap album	Grammy Award for best rap album	1	Cole	Jermaine Lamarr	1985-01-28	36	Rap/Hip-Hop	Wet Dreamz	2015-04-14	none	none	none

Home

Similar se procedeaza si cu celelalte doua tabele.

Bibliografie:

[0] - https://hibernate.org/

[1] - https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[2] - https://www.eclipse.org/downloads/

[3] - https://www.mysql.com/

[4] - https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping

[5] - https://www.java.com/en/

[6] - https://ro.wikipedia.org/wiki/Model-view-controller

[7] - https://ro.wikipedia.org/wiki/XML

[8] - https://en.wikipedia.org/wiki/Plain_old_Java_object

[9] - https://en.wikipedia.org/wiki/Data_access_object