



Uniwersytet Rzeszowski
Wydział Nauk Ścisłych i Technicznych
Aleksandra Cyboron
134899
Informatyka Rok 1
Projekt Spis Filmów

Spis treści

1. Opis projektu.....	2
1.2 Wybrane Technologie:	2
2. Bazy danych.....	2
2.1 Diagram ERD.....	2
2.2 Relacje między tabelami	2
2.3 Konfiguracja połączenia z bazą danych	3
2.4 Inicjalizacja Hibernate	4
2.5 Klasy encji.....	5
2.5.1 Klasa encji Film.....	5
2.5.2 Klasa encji Uzytkownik	6
2.5.3 Klasa encji Rezyser	7
2.5.4 Klasa encji Kategoria.....	8
2.6 Dostęp do danych.....	9
2.6.1 Dostęp do danych klasy Film	9
2.6.2 Dostęp do danych klasy Uzytkownik.....	11
2.6.3 Dostęp do danych klasy Rezyser.....	12
2.6.4 Dostęp do danych klasy Kategoria	13
3. Funkcjonalność aplikacji	15
3.1 Logowanie	15
3.2 Spis filmów.....	18
3.3 Usuniecie rezysera	22
3.4 Usuniecie kategorii.....	25
3.5 Rejestracja	27
4. Interfejs graficzny	31
4.1 CSS	31
4.2 Logowanie.fxml.....	32
5. Repozytorium Projekt	32

1. Opis projektu

Jest aplikacją desktopową stworzoną w celu ułatwienia użytkownikom przeglądania, dodawania, edytowania oraz usuwania informacji o filmach. System umożliwia również zarządzanie kategoriami filmów, reżyserami oraz kontami użytkowników. Główne funkcjonalności obejmują logowanie, rejestrację oraz panel administracyjny z przeglądem wszystkich danych.

1.2 Wybrane Technologie:

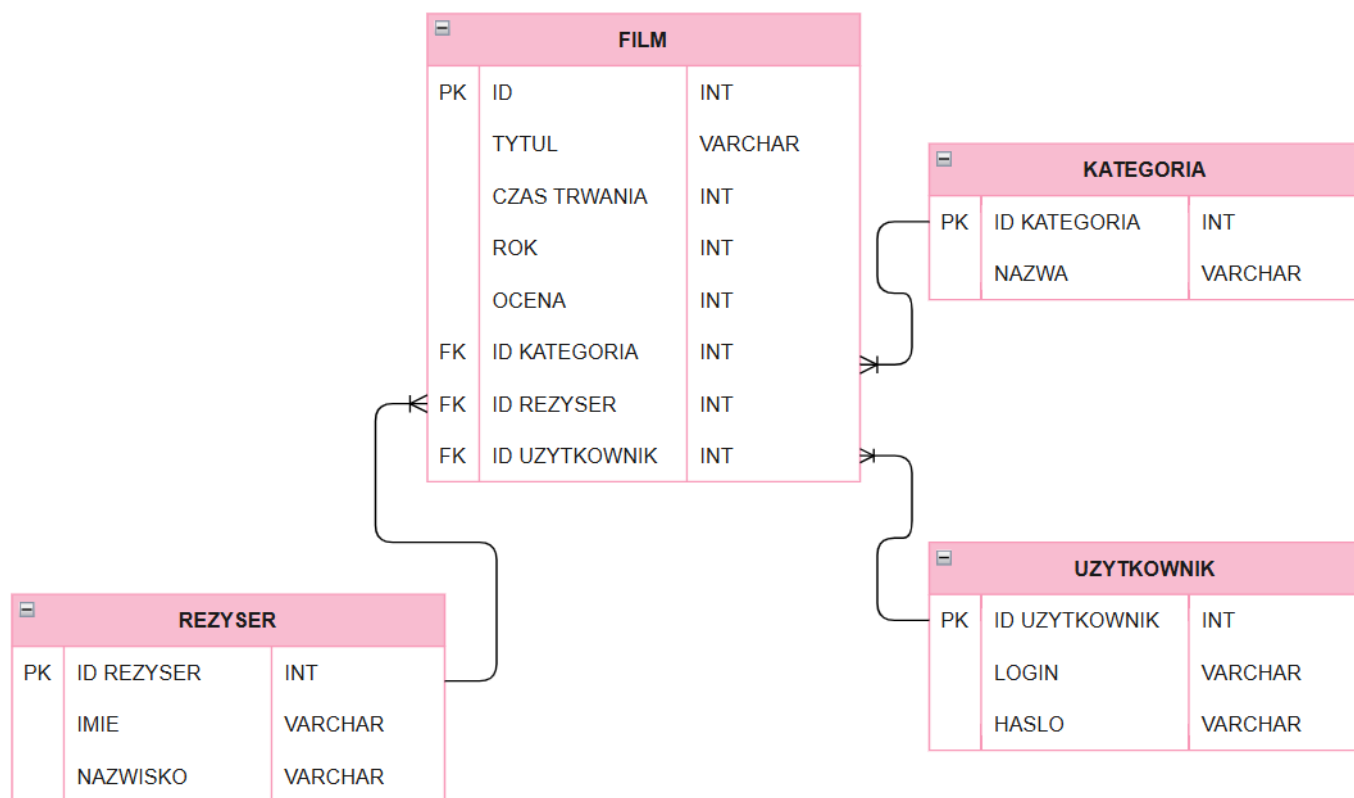
Baza danych: PostgreSQL

Biblioteka: JavaFX

Framework: Hibernate

2. Bazy danych

2.1 Diagram ERD



Rysunek 1: Diagram ERD

2.2 Relacje między tabelami

Uzytkownik [1 : n] Film – jeden użytkownik może dodać wiele filmów.

Kategoria [1 : n] Film – jedna kategoria może obejmować wiele filmów.

Rezyser [1 : n] Film – jeden reżyser może być przypisany do wielu filmów.

2.3 Konfiguracja połączenia z bazą danych

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4     "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5
6 <hibernate-configuration>
7     <session-factory>
8
9         <!-- Połączenie z bazą danych PostgreSQL -->
10        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
11        <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/bazaFilmy</property>
12        <property name="hibernate.connection.username">baza_filmy_haslo</property>
13        <property name="hibernate.connection.password">bazadanych</property>
14
15        <!-- Dialekt PostgreSQL -->
16        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
17
18        <!-- Pokazuj zapytania SQL w konsoli -->
19        <property name="hibernate.show_sql">true</property>
20        <property name="hibernate.format_sql">true</property>
21
22        <!-- Automatyczne tworzenie tabel: validate / update / create / create-drop -->
23        <property name="hibernate.hbm2ddl.auto">update</property>
24
25        <!-- Mapowanie klas encji -->
26        <mapping class="projektFilmy.baza.Film"/>
27        <mapping class="projektFilmy.baza.Kategoria"/>
28        <mapping class="projektFilmy.baza.Reżyser"/>
29        <mapping class="projektFilmy.baza.Uzytkownik"/>
30
31    </session-factory>
32</hibernate-configuration>
```

Rysunek 2: Plik konfiguracyjny Hibernate (hibernate.cfg.xml)

2.4 Inicjalizacja Hibernate

```
1 package projektFilmy.narzedzia;
2
3 import org.hibernate.SessionFactory;
4 import org.hibernate.cfg.Configuration;
5 import projektFilmy.baza.Film;
6 import projektFilmy.baza.Kategoria;
7 import projektFilmy.baza.Rezyser;
8
9 public class KonfiguracjaHibernate { 16 usages
10
11     private static final SessionFactory sessionFactory = buildSessionFactory(); 1 usage
12
13     // Tworzy SessionFactory na podstawie pliku konfiguracyjnego hibernate.cfg.xml
14     private static SessionFactory buildSessionFactory() { 1 usage
15     try {
16         return new Configuration()
17             .configure( resource: "hibernate.cfg.xml") // ładuje plik konfiguracyjny
18             .addAnnotatedClass(Film.class) // Mapuje klasę Film
19             .addAnnotatedClass(Kategoria.class) // Mapuje klasę Kategoria
20             .addAnnotatedClass(Rezyser.class) // Mapuje klasę Rezyser
21             .buildSessionFactory(); // Tworzy SessionFactory
22     } catch (Throwable ex) {
23         System.err.println("Bład tworzenia SessionFactory: " + ex);
24         throw new ExceptionInInitializerError(ex);
25     }
26 }
27
28 // Zwraca globalna instancje SessionFactory
29 public static SessionFactory getSessionFactory() { return sessionFactory; }
32
33 // Zamyka SessionFactory przy zamknięciu aplikacji
34 public static void shutdown() { getSessionFactory().close(); }
37 }
```

Rysunek 3: Klasa odpowiedzialna za inicjalizację połączenia (KonfiguracjaHibernate.java)

2.5 Klasy encji

2.5.1 Klasa encji Film

```
1 package projektFilmy.baza;
2
3 import jakarta.persistence.*; // @Entity, @Table....
4
5 @Entity // Informuje Hibernate, że ta klasa jest mapowana na tabelę w bazie danych
6 @Table(name = "filmy") // Nazwa tabeli w bazie danych, do której ta klasa jest przypisana
7 public class Film {
8
9     @Id // Określa pole jako klucz główny (primary key)
10    @GeneratedValue(strategy = GenerationType.IDENTITY) // Autoinkrementacja w bazie danych
11    private Long id;
12
13    private String tytuł; 3 usages
14    private int rok; 3 usages
15    private int czasTrwania; 2 usages
16    private double ocena; 2 usages
17
18    @ManyToOne // Relacja wiele filmów do jednej kategorii 2 usages
19    @JoinColumn(name = "id_kategoria") // Klucz obcy w tabeli FILMY wskazujący na kategorię
20    private Kategoria kategoria;
21
22    @ManyToOne // Relacja wiele filmów do jednego reżysera 2 usages
23    @JoinColumn(name = "id_rezyser") // Klucz obcy w tabeli FILMY wskazujący na reżysera
24    private Rezyser rezyser;
25
26    public Long getId() { return id; }
27
28
29
30    public void setId(Long id) { this.id = id; }
31
32
33
34    public String getTytuł() { return tytuł; }
35
36
37
38    public void setTytuł(String tytuł) { this.tytuł = tytuł; }
39
40
41
```

Rysunek 4: Klasa Film – encja odwzorowująca tabelę FILM

2.5.2 Klasa encji Uzytkownik

```
1 package projektFilmy.baza;
2
3 // Importuje wszystkie adnotacje JPA potrzebne do mapowania encji (np. @Entity, @Id, @Column itd.)
4 import jakarta.persistence.*;
5
6 @Entity // Oznacza klasę jako encję - będzie odwzorowana na tabelę w bazie danych
7 @Table(name = "uzytkownicy") // Ustawia nazwę tabeli w bazie danych na "uzytkownicy"
8 public class Uzytkownik {
9
10     @Id // Oznacza pole jako klucz główny (Primary Key)
11     @GeneratedValue(strategy = GenerationType.IDENTITY) // Wartość będzie generowana automatycznie (autoinkrementacja)
12     private int id;
13
14     @Column(unique = true, nullable = false) // Kolumna 'login' musi być unikalna i nie może być pusta 4 usages
15     private String login;
16
17     @Column(nullable = false) // Kolumna 'haslo' nie może być pusta 3 usages
18     private String haslo;
19
20     // Konstruktor domyślny - wymagany przez Hibernate
21     public Uzytkownik() {
22     }
23
24     // Konstruktor z parametrami - używany np. przy rejestracji użytkownika
25     public Uzytkownik(String login, String haslo) { 1 usage
26         this.login = login;
27         this.haslo = haslo;
28     }
29
30     @Override
31     public String toString() {
32         return login; // Reprezentacja tekstowa obiektu - wyświetla login
33     }
```

Rysunek 5: Klasa Uzytkownik – encja odwzorowująca tabelę UZYTKOWNIK

2.5.3 Klasa encji Reżyser

```
1 package projektFilmy.baza;
2 import jakarta.persistence.*;
3 import java.util.List; // Umożliwia korzystanie z listy obiektów - potrzebne do relacji jeden-do-wielu (1:N) np. List<Film>
4
5 @Entity // Oznacza klasę jako encję - będzie odwzorowana na tabelę w bazie danych
6 @Table(name = "rezyser") // Ustawia nazwę tabeli w bazie danych na "rezyser"
7 public class Rezyser {
8
9     @Id // Oznacza pole jako klucz główny (Primary Key)
10    @GeneratedValue(strategy = GenerationType.IDENTITY) // Automatyczne generowanie ID (np. autoinkrementacja)
11    private Long id;
12
13    private String imię; 3 usages
14    private String nazwisko; 3 usages
15
16    @OneToMany(mappedBy = "rezyser", cascade = CascadeType.ALL) 2 usages
17    // Relacja jeden-do-wielu - jeden reżyser może mieć wiele filmów.
18    // 'mappedBy = "rezyser"' oznacza, że to pole jest odwzorowaniem relacji z klasy Film.
19    // Cascade = ALL oznacza, że operacje na reżyserze będą wpływać na jego filmy (np. usunięcie)
20    private List<Film> filmy;
21
22    public Long getId() { return id; }
23
24    public void setId(Long id) { this.id = id; }
25
26    @Override
27    public String toString() {
28        return imię + " " + nazwisko; // Reprezentacja tekstowa obiektu Rezyser
29    }
30
31    public String getImię() { return imię; }
32
33    public void setImię(String imię) { this.imię = imię; }
```

Rysunek 6: Klasa Rezyser – encja odwzorowująca tabelę REZYSER

2.5.4 Klasa encji Kategoria

```
1 package projektFilmy.baza;
2 import jakarta.persistence.*;
3 import java.util.List;
4
5 @Entity // Oznacza klasę jako encję - będzie odwzorowana na tabelę w bazie danych
6 @Table(name = "kategoria") // Ustawia nazwę tabeli w bazie danych na "kategoria"
7 public class Kategoria {
8
9     @Id // Oznacza pole jako klucz główny (Primary Key)
10    @GeneratedValue(strategy = GenerationType.IDENTITY) // Automatyczne generowanie ID (np. autoinkrementacja)
11    private Long id;
12    private String nazwa; 3 usages
13
14    @OneToMany(mappedBy = "kategoria", cascade = CascadeType.ALL) 2 usages
15    // Relacja jeden-do-wielu - jedna kategoria może mieć wiele filmów.
16    // 'mappedBy = "kategoria"' oznacza, że to pole odwzorowuje relację z klasy Film.
17    // Cascade = ALL oznacza, że operacje na kategorii będą wpływać na przypisane filmy (np. usuwanie)
18    private List<Film> filmy;
19
20    public Long getId() { return id; }
21
22    public void setId(Long id) { this.id = id; }
23
24    @Override
25    public String toString() { return nazwa; // Reprezentacja tekstowa kategorii - zwraca tylko nazwę }
26
27    public String getNazwa() { return nazwa; }
28
29    public void setNazwa(String nazwa) { this.nazwa = nazwa; }
30
31    public List<Film> getFilmy() { return filmy; }
32
33    public void setFilmy(List<Film> filmy) { this.filmy = filmy; }
```

Rysunek 7: Klasa Kategoria – encja odwzorowująca tabelę KATEGORIA

2.6 Dostęp do danych

2.6.1 Dostęp do danych klasy Film

```
1 package projektFilmy.baza;
2 import org.hibernate.Session; // Import klasy Hibernate Session - reprezentuje połączenie z bazą danych
3 import org.hibernate.Transaction; // Import klasy Transaction - pozwala zarządzać transakcjami w Hibernate
4 import projektFilmy.narzedzia.KonfiguracjaHibernate; // Import klasy udostępniającej SessionFactory (połączenie z bazą)
5 import java.util.List;
6
7 public class FilmDAO { 4 usages
8
9     // Pobiera wszystkie rekordy typu Film z bazy danych
10    public static List<Film> pobierzWszystkie() { 1 usage
11        // Otwiera nową sesję Hibernate do wykonania operacji SELECT
12        try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
13            // HQL: zapytanie do wszystkich obiektów typu Film (czyli: SELECT * FROM filmy)
14            return session.createQuery("from Film", Film.class).list();
15        }
16    }
17
18    // Dodaje nowy film do bazy danych
19    public static void dodaj(Film film) { 1 usage
20        // Otwiera sesję do połączenia z bazą
21        try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
22            // Rozpoczęcie transakcji - wszystkie operacje od teraz są "tymczasowe",
23            // dopóki nie zostaną zatwierdzone (commit). Dzięki temu możliwe jest
24            // wykonanie operacji atomowo - czyli albo całość się powiedzie, albo nie.
25            Transaction tx = session.beginTransaction();
26
27            // Zapisanie obiektu Film do bazy (Hibernate wykona INSERT)
28            session.persist(film);
29
30            // Zatwierdzenie transakcji - dane trafiają na stałe do bazy danych
31            tx.commit();
32        }
33    }
```

Rysunek 8: Klasa FilmDAO – operacje dodania na tabeli FILM

```

35 // Aktualizuje istniejący rekord filmu
36 public static void edytuj(Film film) { 1 usage
37     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
38         // Rozpoczęcie transakcji - aby zmiany były bezpieczne i spójne
39         Transaction tx = session.beginTransaction();
40
41         // Aktualizacja danych obiektu Film (Hibernate wykona UPDATE)
42         session.merge(film);
43         tx.commit(); // Zatwierdzenie zmian w bazie danych
44     }
45 }
46
47 // Usuwa film z bazy danych
48 @ public static void usun(Film film) { 1 usage
49     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
50         // Rozpoczęcie transakcji - usuwanie też wymaga bezpieczeństwa transakcyjnego
51         Transaction tx = session.beginTransaction();
52
53         // Pobranie obiektu z bazy po ID - Hibernate musi najpierw go załadować
54         Film filmDoUsuniecia = session.get(Film.class, film.getId());
55
56         if (filmDoUsuniecia != null) {
57             // Zerowanie relacji - żeby nie było konfliktu z kluczami obcymi (FK)
58             filmDoUsuniecia.setRezyser(null);
59             filmDoUsuniecia.setKategoria(null);
60
61             // Usunięcie obiektu z bazy (Hibernate wykona DELETE)
62             session.remove(filmDoUsuniecia);
63         }
64
65         tx.commit(); // Zatwierdzenie operacji usunięcia
66     }

```

Rysunek 9: Klasa FilmDAO – operacje edycji i usunięcia na tabeli FILM

2.6.2 Dostęp do danych klasy Uzytkownik

```
1 package projektFilmy.baza;
2 import org.hibernate.Session; // Import klasy Hibernate Session - połączenie z bazą danych
3 import org.hibernate.Transaction; // Import klasy Transaction - pozwala zarządzać transakcjami (BEGIN, COMMIT)
4 import org.hibernate.query.Query; // Import klasy Query - służy do tworzenia zapytań HQL
5 import projektFilmy.narzedzia.KonfiguracjaHibernate;
6
7 public class UzytkownikDAO { 4 usages
8
9     // Rejestruje nowego użytkownika w bazie danych, jeśli login nie jest już zajęty
10    public static boolean zarejestruj(String login, String haslo) { 1 usage
11        try (Session sesja = KonfiguracjaHibernate.getSessionFactory().openSession()) {
12
13            Transaction tx = sesja.beginTransaction(); // Rozpoczęcie transakcji - zapis do bazy wymaga transakcji
14
15            // Tworzenie zapytania HQL: sprawdzanie, czy użytkownik o danym loginie już istnieje
16            // "from Uzytkownik" - HQL operuje na nazwach klas
17            Query<Uzytkownik> zapytanie = sesja.createQuery(
18                s: "from Uzytkownik where login = :login", Uzytkownik.class);
19            zapytanie.setParameter(s: "login", login); // Przekazanie parametru do zapytania
20
21
22            if (!zapytanie.getResultList().isEmpty()) {
23                return false; // Jeśli lista wyników nie jest pusta, login już istnieje - rejestracja nieudana
24            }
25
26            Uzytkownik nowy = new Uzytkownik(login, haslo); // Tworzymy nowego użytkownika i zapisujemy do bazy
27            sesja.persist(nowy); // Hibernate wykona INSERT
28
29            tx.commit(); // Zatwierdzenie transakcji - dane trafią do bazy
30            return true; // Rejestracja zakończona powodzeniem
31        }
32    }
33
34    // Próbuje zalogować użytkownika - zwraca true, jeśli login i hasło się zgadzają
35    public static boolean zaloguj(String login, String haslo) { 1 usage
36        try (Session sesja = KonfiguracjaHibernate.getSessionFactory().openSession()) {
37
38            // Tworzymy zapytanie HQL, które sprawdzi, czy istnieje użytkownik o podanym loginie i hasle
39            Query<Uzytkownik> zapytanie = sesja.createQuery(
40                s: "from Uzytkownik where login = :login and haslo = :haslo", Uzytkownik.class);
41
42            // Ustawiamy parametry zapytania
43            zapytanie.setParameter(s: "login", login);
44            zapytanie.setParameter(s: "haslo", haslo);
45
46            // Jeśli lista wyników nie jest pusta, logowanie zakończone sukcesem
47            return !zapytanie.getResultList().isEmpty();
48        }
49    }
50 }
51
```

Rysunek 10: Klasa UzytkownikDAO – obsługa kont użytkowników

2.6.3 Dostęp do danych klasy Reżyser

```
7 public class RezyserDAO { 5 usages
9 // Pobiera wszystkich reżyserów z bazy danych
10 public static List<Rezyser> pobierzWszystkich() { 3 usages
11     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
12         // HQL: zapytanie do wszystkich encji typu Rezyser
13         return session.createQuery("from Rezyser", Rezyser.class).list();
14     }
15 }
16 // Zwraca istniejącego reżysera, jeśli istnieje; w przeciwnym razie dodaje nowego do bazy
17 @ public static Rezyser znajdzLubDodaj(String tekst) { 2 usages
18
19     // Przeszukiwanie istniejących reżyserów po imieniu i nazwisku
20     for (Rezyser r : pobierzWszystkich()) {
21         if ((r.getImie() + " " + r.getNazwisko()).equalsIgnoreCase(tekst)) {
22             return r; // Zwraca istniejącego reżysera
23         }
24     }
25
26     // Jeśli nie znaleziono, dzieli podany tekst na imię i nazwisko
27     String[] czesci = tekst.split(" ", limit: 2);
28     String imie = czesci.length > 0 ? czesci[0] : "";
29     String nazwisko = czesci.length > 1 ? czesci[1] : "";
30
31     Rezyser nowy = new Rezyser(); // Tworzenie nowego obiektu reżysera
32     nowy.setImie(imie);
33     nowy.setNazwisko(nazwisko);
34
35     // Zapisanie nowego reżysera do bazy danych
36     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
37         Transaction tx = session.beginTransaction(); // Rozpoczęcie transakcji
38         session.persist(nowy); // INSERT nowego obiektu
39         tx.commit(); // Zatwierdzenie transakcji
40     }
41
42     return nowy;
43
44 public static boolean dodaj(String tekst) { 1 usage new *
45     // Sprawdź, czy reżyser już istnieje
46     for (Rezyser r : pobierzWszystkich()) {
47         if ((r.getImie() + " " + r.getNazwisko()).equalsIgnoreCase(tekst)) {
48             return false; // reżyser istnieje
49         }
50     }
51
52     // Rozdzielenie tekstu na imię i nazwisko
53     String[] czesci = tekst.trim().split(" ", limit: 2);
54     String imie = czesci.length > 0 ? czesci[0] : "";
55     String nazwisko = czesci.length > 1 ? czesci[1] : "";
56
57     // Dodaj nowego reżysera
58     Rezyser nowy = new Rezyser();
59     nowy.setImie(imie);
60     nowy.setNazwisko(nazwisko);
61
62     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
63         Transaction tx = session.beginTransaction();
64         session.persist(nowy);
65         tx.commit();
66     }
67
68     return true;
69 }
```

Rysunek 11: Na rysunku widać metody klasy RezyserDAO do obsługi danych reżyserów. Metoda pobierzWszystkich() zwraca listę wszystkich reżyserów z bazy danych. Metoda znajdzLubDodaj(String tekst) najpierw szuka reżysera na podstawie imienia i nazwiska, a jeśli go nie znajdzie — dzieli tekst na części, tworzy nowy obiekt Rezyser, ustawia dane i zapisuje go do bazy.

Rysunek 12: Metoda dodaj(String tekst) odpowiada za dodanie nowego reżysera do bazy danych. Najpierw sprawdza, czy reżyser już istnieje, porównując pełne imię i nazwisko. Następnie dzieli wprowadzony tekst na imię i nazwisko (na podstawie spacji), tworzy nowy obiekt Rezyser, ustawia jego dane, a na końcu zapisuje go do bazy przy użyciu Hibernate.

```

// Usuwa reżysera z bazy danych, ale tylko jeśli nie ma przypisanych filmów
public static boolean usun(String tekst) { 1 usage ± a-0130 *
    try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
        Reżyser znaleziony = session.createQuery(
            s: "from Reżyser r where lower(concat(r.imie, ' ', r.nazwisko)) = :tekst", Reżyser.class)
            .setParameter(s: "tekst", tekst.toLowerCase())
            .uniqueResult();

        if (znaleziony == null) return false;

        Long ileFilmow = session.createQuery(
            s: "select count(f) from Film f where f.reżyser.id = :id", Long.class)
            .setParameter(s: "id", znaleziony.getId())
            .uniqueResult();

        if (ileFilmow != null && ileFilmow == 0) {
            Transaction tx = session.beginTransaction();
            session.remove(session.contains(znaleziony) ? znaleziony : session.merge(znaleziony));
            tx.commit();
            return true;
        }
    }
    return false;
}

```

Rysunek 13: Metoda `usun(String tekst)` próbuje usunąć reżysera na podstawie jego imienia i nazwiska, ale tylko wtedy, gdy nie ma on przypisanych filmów. Najpierw wyszukuje reżysera w bazie, a następnie sprawdza liczbę powiązanych z nim filmów. Jeśli liczba filmów wynosi 0, rozpoczyna transakcję i usuwa reżysera. W przeciwnym razie zwraca `false`, nie dopuszczając do usunięcia

2.6.4 Dostęp do danych klasy `Kategoria`

```

4 import projektFilmy.narzedzia.KonfiguracjaHibernate; // Import klasy konfigurującej połączenie z bazą danych (SessionFactory)
5 import java.util.List;
6
7 public class KategoriaDAO { 5 usages
8
9     // Pobiera wszystkie rekordy typu Kategoria z bazy danych
10    public static List<Kategoria> pobierzWszystkie() { 3 usages
11        try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
12            // HQL: zapytanie do wszystkich encji Kategoria
13            return session.createQuery(s: "from Kategoria", Kategoria.class).list();
14        }
15    }
16
17    // Zwraca istniejącą kategorię o podanej nazwie lub dodaje nową, jeśli nie istnieje
18    @ public static Kategoria znajdzLubDodaj(String tekst) { 2 usages
19        // Sprawdzamy, czy dana kategoria już istnieje (ignorując wielkość liter)
20        for (Kategoria k : pobierzWszystkie()) {
21            if (k.getNazwa().equalsIgnoreCase(tekst)) {
22                return k; // Znaleziona - zwracamy istniejącą
23            }
24        }
25        Kategoria nowa = new Kategoria(); // Jeśli nie znaleziono - tworzymy nową kategorię
26        nowa.setNazwa(tekst);
27
28        // Zapisujemy nową kategorię do bazy danych
29        try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
30            Transaction tx = session.beginTransaction(); // Rozpoczęcie transakcji
31            session.persist(nowa); // Zapisanie nowej encji (INSERT)
32            tx.commit(); // Zatwierdzenie transakcji
33        }
34        return nowa;
35    }
}

```



```

36 public static boolean dodaj(String nazwa) { 1 usage new *
37     for (Kategoria k : pobierzWszystkie()) {
38         if (k.getNazwa().equalsIgnoreCase(nazwa)) {
39             return false;
40         }
41     }
42
43     Kategoria nowa = new Kategoria();
44     nowa.setNazwa(nazwa);
45
46     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
47         Transaction tx = session.beginTransaction();
48         session.persist(nowa);
49         tx.commit();
50     }
51
52     return true;
53 }

```

Rysunek 14: Metoda `dodaj(String nazwa)` sprawdza, czy kategoria o podanej nazwie już istnieje (ignorując wielkość liter). Jeśli nie istnieje, tworzy nowy obiekt `Kategoria`, ustawia jego nazwę, a następnie zapisuje go do bazy danych przy użyciu Hibernate i transakcji. Jeśli taka kategoria już istnieje, metoda zwraca `false`.

```

55 @ public static boolean usun(String nazwa) { 1 usage a-0130 *
56     try (Session session = KonfiguracjaHibernate.getSessionFactory().openSession()) {
57         Kategoria znaleziona = session.createQuery(
58             s: "from Kategoria k where lower(k.nazwa) = :nazwa", Kategoria.class)
59             .setParameter(s: "nazwa", nazwa.toLowerCase())
60             .uniqueResult();
61
62         if (znaleziona == null) return false;
63
64         Long ileFilmow = session.createQuery(
65             s: "select count(f) from Film f where f.kategoria.id = :id", Long.class)
66             .setParameter(s: "id", znaleziona.getId())
67             .uniqueResult();
68
69         if (ileFilmow != null && ileFilmow == 0) {
70             Transaction tx = session.beginTransaction();
71             session.remove(session.contains(znaleziona) ? znaleziona : session.merge(znaleziona));
72             tx.commit();
73             return true;
74         }
75     }
76
77     return false;
78 }

```

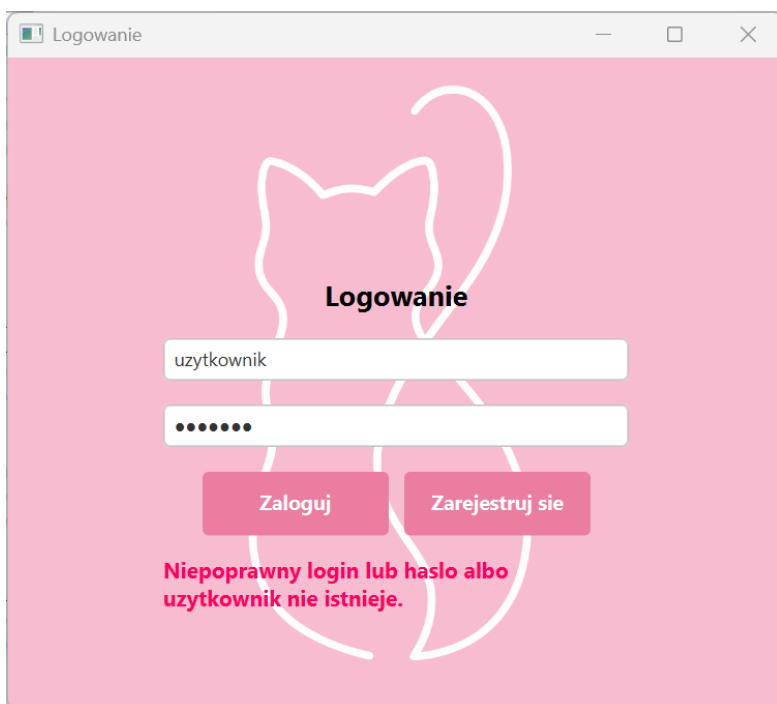
Rysunek 15: Metoda `usun(String nazwa)` próbuje usunąć kategorię z bazy danych, ale tylko wtedy, gdy nie jest przypisana do żadnego filmu. Wyszukuje kategorię po nazwie (ignorując wielkość liter), a następnie sprawdza liczbę filmów z nią powiązanych. Jeśli jest ich zero, rozpoczyna transakcję i usuwa kategorię z bazy. W przeciwnym przypadku zwraca `false`.

3. Funkcjonalność aplikacji

3.1 Logowanie



Rysunek 16: Wygląd widoku logowania



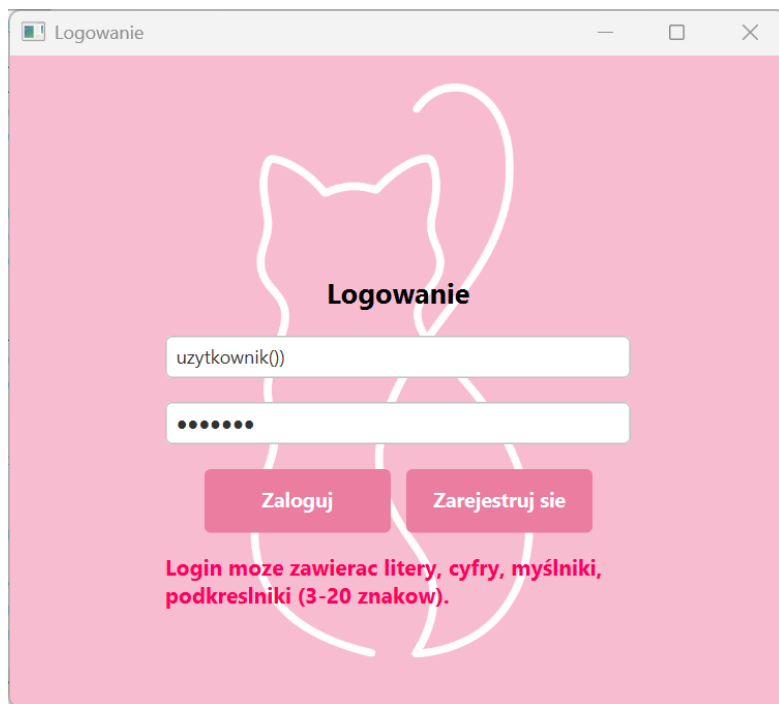
Rysunek 17: Wygląd widoku logowania, gdy uzytkownik poda nieprawidłowe dane


```

1 package projektFilmy.Kontroler;
2
3 import javafx.fxml.FXML;
4 import javafx.scene.control.*;
5 import javafx.fxml.FXMLLoader;
6 import javafx.scene.Parent;
7 import javafx.scene.Scene;
8 import javafx.stage.Stage;
9 import projektFilmy.baza.UzytkownikDAO;
10
11 public class LogowanieKontroler { 1 usage
12
13     </> @FXML private TextField loginField;
14     </> @FXML private PasswordField hasloField;
15     </> @FXML private Label komunikatLabel;
16
17     @FXML
18     private void zaloguj() {
19         String login = loginField.getText().trim();
20         String haslo = hasloField.getText().trim();
21
22         if (!czyPoprawneDane(login, haslo)) {
23             return;
24         }
25
26         if (UzytkownikDAO.zaloguj(login, haslo)) {
27             otworzWidokGlownegoOkna();
28         } else {
29             komunikatLabel.setText("Niepoprawny login lub haslo albo uzytkownik nie istnieje.");
30         }
31     }

```

Rysunek 18: Tworzy mechanizm logowania użytkownika przy użyciu danych wprowadzonych w polach formularza. Po kliknięciu przycisku „Zaloguj”, metoda `zaloguj()` weryfikuje poprawność loginu i hasła oraz przekazuje je do metody `zaloguj()` z `UzytkownikDAO`. Jeśli dane są poprawne, użytkownik zostaje przeniesiony do głównego widoku aplikacji. W przeciwnym razie wyświetlany jest komunikat o błędzie.



Rysunek 19: Wygląd widoku logowania, gdy użytkownik poda nieprawidłowe dane, które nawet nie mogą istnieć

```

44
45 @ private boolean czyPoprawneDane(String login, String haslo) { 1 usage
46     if (login.isEmpty() || haslo.isEmpty()) {
47         komunikatLabel.setText("Login i haslo sa wymagane.");
48         return false;
49     }
50
51     if (!login.matches( regex: "[A-Za-z0-9._-]{3,20}")) {
52         komunikatLabel.setText("Login może zawierać litery, cyfry, myślniki, podkreślniki (3-20 znaków).");
53         return false;
54     }
55
56     if (haslo.length() < 6 || !haslo.matches( regex: ".*\\d.*")) {
57         komunikatLabel.setText("Hasło musi mieć min. 6 znaków i zawierać co najmniej jedną cyfrę.");
58         return false;
59     }
60
61     return true;
62 }

```

Rysunek 20: Metoda sprawdza poprawność formatu loginu i hasła przed próbą logowania. Weryfikuje, czy pola nie są puste, czy login spełnia wymagania (3–20 znaków: litery, cyfry, myślniki, podkreślniki) oraz czy hasło ma minimum 6 znaków i zawiera przynajmniej jedną cyfrę. W przypadku błędów użytkownik otrzymuje odpowiedni komunikat.

3.2 Spis filmów

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia
The Lion King	Rob Minkoff	1994	99	8.4	Animacja
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja
Cars 3	Brian Fee	2017	102	6.6	Animacja
tytuł	Wojciech Koziol	2000	323	2.0	Komedia
tytuł	Wojciech Koziol	28282	300	5.0	Animacja
tytuł	Dean Fleischer-Camp	28282	300	5.0	Animacja

Tytuł: Reżyser:

Rok: Czas trwania:

Ocena: Kategoria:

Dodaj

Edytuj

Usun

Reżyser

Kategoria

Rysunek 21: Spis filmów, który służy użytkownikowi do zapisywania filmów, które już oglądał i może ocenić je sobie na przyszłość, aby o nich nie zapomnieć lub wiedzieć do którego chętnie wrócić

```
private void otwórzWidokGłównegoOkna() { 1 usage
    try {
        Parent root = FXMLLoader.load(getClass().getResource( name: "/widok.fxml"));
        Stage stage = (Stage) loginField.getScene().getWindow();
        stage.setScene(new Scene(root));
    } catch (Exception e) {
        e.printStackTrace();
        komunikatLabel.setText("Błąd ładowania widoku.");
    }
}
```

Rysunek 22: Po poprawnym logowaniu użytkownik zostaje przeniesiony do głównego widoku aplikacji. Metoda ładuje plik widok.fxml i ustawia go jako nową scenę w bieżącym oknie (Stage). W przypadku błędu podczas ładowania, użytkownik otrzymuje komunikat o niepowodzeniu.

Logowanie

Filmy

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria	
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia	
The Lion King	Rob Minkoff	1994	99	8.4	Animacja	
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja	
Cars 3	Brian Fee	2017	102	6.6	Animacja	
tytuł	Wojciech Koziol	2000	323	2.0	Komedia	
tytuł	Wojciech Koziol	28282	300	5.0	Animacja	
tytuł	Dean Fleischer-Camp	28282	300	5.0	Animacja	

Tytuł:
Reżyser:
Rok:
Ocena:

Czas trwania:
Kategoria:

Pole tytuł nie może być puste.

Dodaj
Edytuj
Usun
Reżyser
Kategoria

Rysunek 23: Walidacja pola tytuł

Logowanie

Filmy

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria	
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia	
The Lion King	Rob Minkoff	1994	99	8.4	Animacja	
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja	
Cars 3	Brian Fee	2017	102	6.6	Animacja	
tytuł	Wojciech Koziol	2000	323	2.0	Komedia	
tytuł	Wojciech Koziol	28282	300	5.0	Animacja	
tytuł	Dean Fleischer-Camp	28282	300	5.0	Animacja	

Tytuł:
Reżyser:
Rok:
Ocena:

Czas trwania:
Kategoria:

Wypełnij wszystkie pola liczbowe.

Dodaj
Edytuj
Usun
Reżyser
Kategoria

Rysunek 24: Walidacja pól liczbowych

Logowanie

Filmy

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia
The Lion King	Rob Minkoff	1994	99	8.4	Animacja
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja
Cars 3	Brian Fee	2017	102	6.6	Animacja
tytuł	Wojciech Koziol	2000	323	2.0	Komedia
tytuł	Wojciech Koziol	28282	300	5.0	Animacja
tytuł	Dean Fleischer-Camp	28282	300	5.0	Animacja

Tytuł:

Reżyser:

Rok:

Czas trwania:

Ocena:

Kategoria:

Reżyser może zawierać tylko litery i myślniki.

Dodaj

Edytuj

Usun

Reżyser

Kategoria

Rysunek 25: Walidacja pola reżyser

Logowanie

Filmy

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia
The Lion King	Rob Minkoff	1994	99	8.4	Animacja
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja
Cars 3	Brian Fee	2017	102	6.6	Animacja
tytuł	Wojciech Koziol	2000	323	2.0	Komedia
tytuł	Wojciech Koziol	28282	300	5.0	Animacja
tytuł	Dean Fleischer-Camp	28282	300	5.0	Animacja

Tytuł:

Reżyser:

Rok:

Czas trwania:

Ocena:

Kategoria:

Kategoria może zawierać tylko litery i myślniki.

Dodaj

Edytuj

Usun

Reżyser

Kategoria

Rysunek 26: Walidacja pola kategoria

```

69 // Walidacje pól tekstowych
70 if (tytuł.isEmpty()) {
71     komunikatLabel.setText("Pole tytuł nie może być puste.");
72     return;
73 }
74 if (!tytuł.matches(regex: "[A-Za-z0-9 .!?-]+")) {
75     komunikatLabel.setText("Tytuł zawiera niedozwolone znaki.");
76     return;
77 }
78 if (rokText.isEmpty() || czasText.isEmpty() || ocenaText.isEmpty()) {
79     komunikatLabel.setText("Wypełnij wszystkie pola liczbowe.");
80     return;
81 }
82 if (!rezTekst.matches(regex: "[A-Za-z -]+")) {
83     komunikatLabel.setText("Reżyser może zawierać tylko litery i myślniki.");
84     return;
85 }
86 if (!katTekst.matches(regex: "[A-Za-z -]+")) {
87     komunikatLabel.setText("Kategoria może zawierać tylko litery i myślniki.");
88     return;
89 }
90

```

Rysunek 27: Fragment kodu sprawdza poprawność danych wejściowych w formularzu dodawania filmu. Waliduje obecność tytułu oraz formaty tekstowe pól: tytuł (dozwolone znaki), reżyser i kategoria (tylko litery i myślniki). W przypadku błędu użytkownik otrzymuje precyzyjny komunikat w interfejsie.

```

95
96 if (rok <= 0) { komunikatLabel.setText("Rok musi być większy od 0."); return; }
97 if (czas <= 0) { komunikatLabel.setText("Czas trwania musi być większy od 0."); return; }
98 if (ocena < 0 || ocena > 10) { komunikatLabel.setText("Ocena musi być w zakresie 0-10."); return; }
99

```

Rysunek 28: Ten fragment kodu sprawdza poprawność wartości liczbowych wpisanych przez użytkownika. Rok i czas trwania muszą być większe od zera, a ocena mieścić się w zakresie od 0 do 10. W razie nieprawidłowych danych, użytkownik otrzymuje komunikat błędu.

Filmy

Tytuł	Reżyser	Rok	Czas trwania	Ocena	Kategoria
Lilo & Stitch	Dean Fleischer-Camp	2025	108	7.6	Komedia
The Lion King	Rob Minkoff	1994	99	8.4	Animacja
Shrek	Andrew Adamson	2001	90	7.5	Animacja
Shrek 2	Andrew Adamson	2004	92	7.5	Animacja
Cars 3	Brian Fee	2017	102	6.6	Animacja
tytuł	reżyser	2025	10	10.0	Animacja

Tytuł: Reżyser:

Rok: Czas trwania:

Ocena: Kategoria:

Film dodany!

Dodaj

Edytuj

Usun

Usuń reżysera

Usuń kategorię

Rysunek 29: Po poprawnym dodaniu filmu mamy komunikat, że został on dodany

3.3 Usuniecie reżysera

Reżyser

Dodaj nowego reżysera:

Imię

Nazwisko

Dodaj Reżysera

Wybierz reżysera do usunięcia:

Usuń Reżysera

Wróć do spisu filmów

Rysunek 30: Widok umożliwia dodawanie i usuwanie reżysera w aplikacji. Z lewej strony użytkownik może wprowadzić imię i nazwisko nowego reżysera, a następnie kliknąć „Dodaj Reżysera”, co zapisuje dane do bazy i odświeża listę. Z prawej strony możliwe jest wybranie reżysera z listy i usunięcie go z bazy, o ile nie ma przypisanych filmów. Przycisk „Wróć do spisu filmów” przenosi użytkownika z powrotem do głównego widoku aplikacji.

```

40 @FXML new *
41 private void dodajReżysera() {
42     String imie = imieField.getText().trim();
43     String nazwisko = nazwiskoField.getText().trim();
44
45     if (!imie.isEmpty() && !nazwisko.isEmpty()) {
46         String pelne = imie + " " + nazwisko;
47         boolean dodano = RezyserDAO.dodaj(pelne);
48
49         if (!imie.matches( regex: "[A-Za-z \\-]{1,50}" ) ||
50             !nazwisko.matches( regex: "[A-Za-z \\-]{1,50}" )) {
51             pokazAlert( tekst: "Imię i nazwisko mogą zawierać tylko litery oraz myślniki. max 50 znaków.");
52             return;
53         }
54
55         if (dodano) {
56             rezyserComboBox.getItems().clear();
57             rezyserComboBox.getItems().addAll(RezyserDAO.pobierzWszystkich());
58             imieField.clear();
59             nazwiskoField.clear();
60             pokazAlert( tekst: "Dodano reżysera: " + pelne);
61         } else {
62             pokazAlert( tekst: "Reżyser " + pelne + " już istnieje.");
63         }
64     } else {
65         pokazAlert( tekst: "Uzupełnij oba pola, aby dodać reżysera.");
66     }
67 }

```

Rysunek 31: Na rysunku przedstawiono metodę `dodajReżysera()`, która umożliwia dodanie nowego reżysera do bazy danych. Najpierw pobierane są dane z pól tekstowych, a następnie następuje walidacja — imię i nazwisko muszą zawierać tylko litery oraz myślnik i mieć maksymalnie 50 znaków. Weryfikacja odbywa się przy pomocy wyrażenia regularnego (`.matches(...)`). Jeśli dane są poprawne i reżysera nie ma jeszcze w bazie, zostaje on dodany, pola są czyszczone, a lista w `ComboBoxie` odświeżana. W przeciwnym wypadku wyświetlany jest komunikat o błędzie.

```

24 @FXML new *
25 private void usunReżysera() {
26     Rezyser rezyser = rezyserComboBox.getValue();
27
28     if (rezyser != null) {
29         String pelneImieNazwisko = rezyser.getImie() + " " + rezyser.getNazwisko();
30         boolean usunieto = RezyserDAO.usun(pelneImieNazwisko);
31
32         if (usunieto) {
33             rezyserComboBox.getItems().remove(rezyser);
34             pokazAlert( tekst: "Usunięto reżysera: " + pelneImieNazwisko);
35         } else {
36             pokazAlert( tekst: "Nie można usunąć reżysera; Ma przypisane filmy.");
37         }
38     }
39 }

```

Rysunek 32: Na rysunku przedstawiono metodę `usunReżysera()`, która usuwa wybranego reżysera z listy, jeśli nie ma on przypisanych filmów. Po udanym usunięciu pojawia się komunikat potwierdzający, a reżyser znika z listy rozwijanej; w przeciwnym razie wyświetlany jest komunikat o braku możliwości usunięcia.

Logowanie

Reżyser

Dodaj nowego reżysera:

hashahs(

Nazwisko

Dodaj Reżysera

Wybierz reżysera do usunięcia:

Usuń Reżysera

Uzupełnij oba pola, aby dodać reżysera.

Wróć do spisu filmow

Rysunek 33: Nie wprowadzenie danych uniemożliwia dodanie reżysera i pojawia się komunikat

Logowanie

Reżyser

Dodaj nowego reżysera:

hashahs(

tythg

Dodaj Reżysera

Wybierz reżysera do usunięcia:

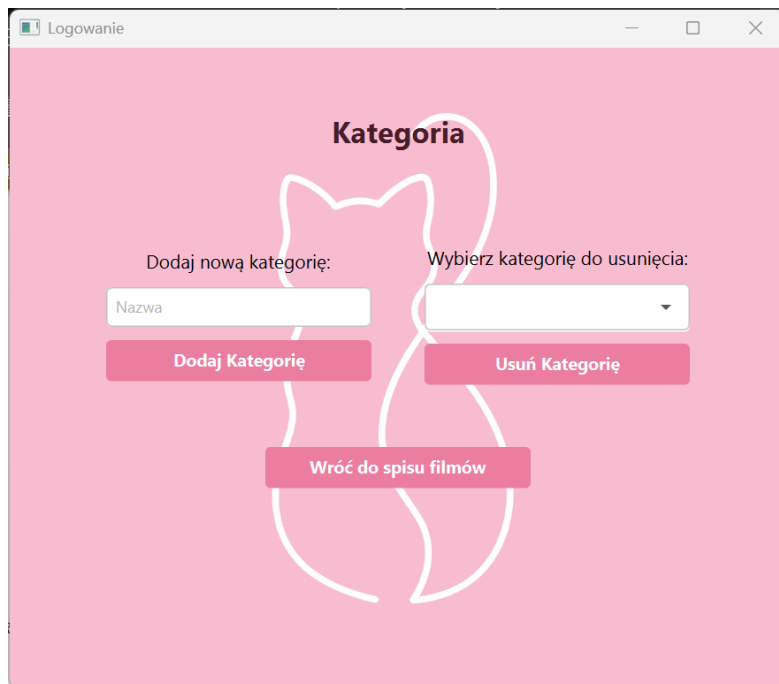
Usuń Reżysera

Imię i nazwisko mogą zawierać tylko litery oraz myślniki. max 50 znakow.

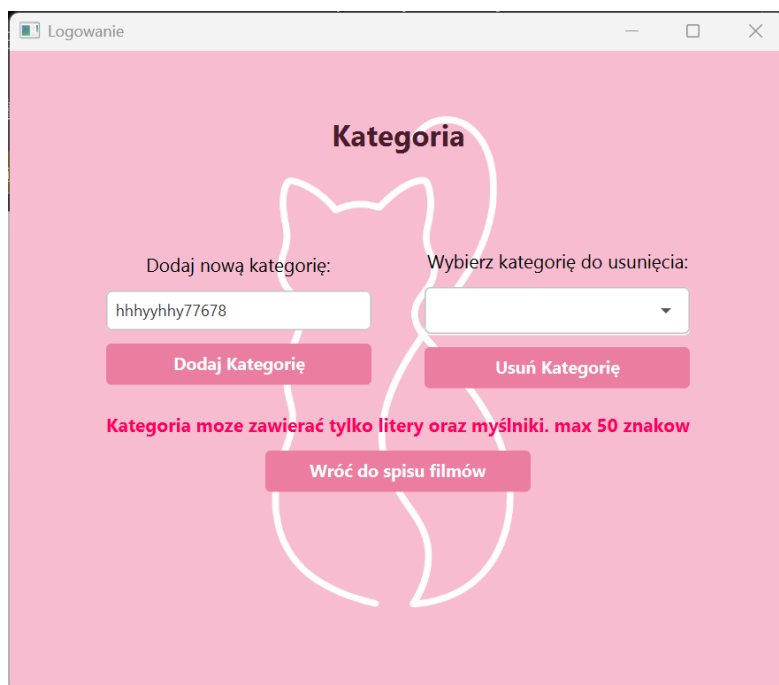
Wróć do spisu filmow

Rysunek 34: Wprowadzenie błędnych danych uniemożliwia dodanie reżysera i pojawia się komunikat.

3.4 Usuniecie kategorii



Rysunek 35: Z lewej strony użytkownik może wpisać nazwę nowej kategorii i kliknąć przycisk „Dodaj Kategorię”, aby zapisać ją do bazy danych i odświeżyć listę. Po prawej stronie dostępna jest lista istniejących kategorii, z której można wybrać jedną i usunąć ją z bazy. Przycisk „Wróć do spisu filmów” umożliwia powrót do głównego widoku aplikacji.



Rysunek 36: Wprowadzenie błędnych danych uniemożliwia dodanie kategorii i pojawia się komunikat.

```

23 @FXML new *
24 private void dodajKategorie() {
25     String nazwa = nazwaField.getText().trim();
26
27     if (!nazwa.isEmpty()) {
28         boolean dodano = KategoriaDAO.dodaj(nazwa);
29         if (!nazwa.matches( regex: "[A-Za-z \\-]{1,50}" ) ) {
30             komunikatLabel.setText("Kategoria może zawierać tylko litery oraz myślniki. max 50 znaków");
31             return;
32         }
33         if (dodano) {
34             kategoriaComboBox.getItems().clear();
35             kategoriaComboBox.getItems().addAll(KategoriaDAO.pobierzWszystkie());
36             nazwaField.clear();
37             komunikatLabel.setText("Dodano kategorię: " + nazwa);
38         } else {
39             komunikatLabel.setText("Kategoria \"" + nazwa + "\" już istnieje.");
40         }
41     } else {
42         komunikatLabel.setText("Wpisz nazwę kategorii.");
43     }
44 }

```

Rysunek 37: Na rysunku przedstawiono metodę `dodajKategorie()`, która umożliwia dodanie nowej kategorii do bazy danych. Wprowadzona nazwa jest najpierw sprawdzana — musi być niepusta i zgodna z wyrażeniem dopuszczającym tylko litery i myślniki (do 50 znaków). Jeśli kategoria zostanie poprawnie dodana, lista jest odświeżana i użytkownik otrzymuje komunikat potwierdzający. W przeciwnym razie wyświetlany jest odpowiedni komunikat błędu.

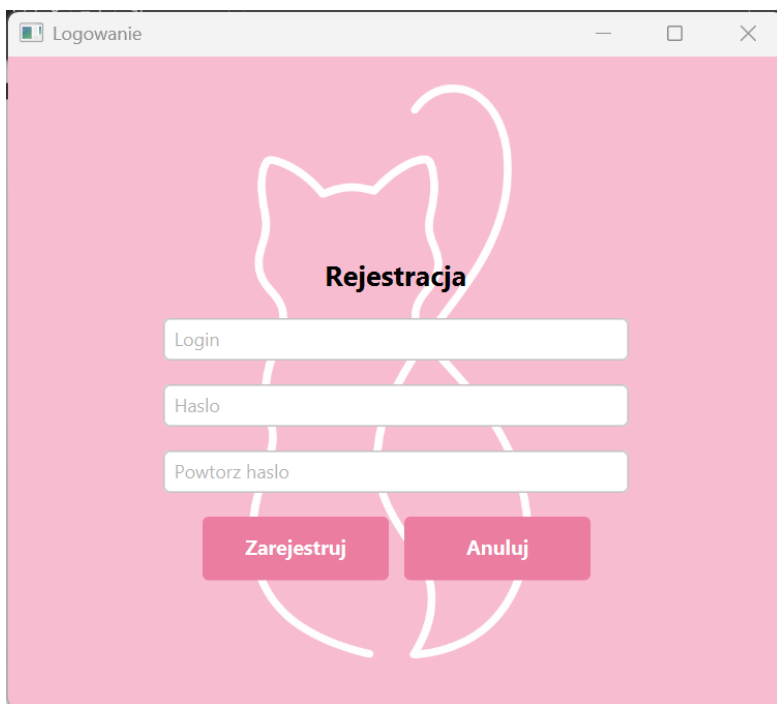
```

46 @FXML new *
47 private void usunKategorie() {
48     Kategoria kategoria = kategoriaComboBox.getValue();
49
50     if (kategoria != null) {
51         boolean usunieto = KategoriaDAO.usun(kategoria.getNazwa());
52
53         if (usunieto) {
54             kategoriaComboBox.getItems().remove(kategoria);
55             komunikatLabel.setText("Usunięto kategorię: " + kategoria.getNazwa());
56         } else {
57             komunikatLabel.setText("Nie można usunąć - kategoria przypisana do filmu.");
58         }
59     }
60 }

```

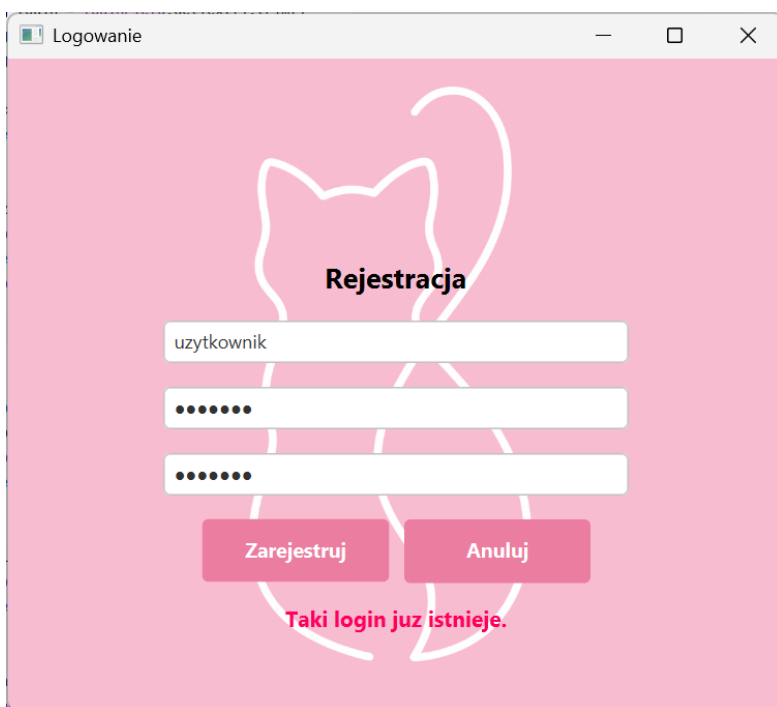
Rysunek 38: Kod przedstawia metodę `usunKategorie()`, która usuwa wybraną kategorię z listy, jeśli nie jest ona przypisana do żadnego filmu. W przypadku powodzenia kategoria znika z ComboBoxa, a użytkownik otrzymuje potwierdzenie; w przeciwnym razie pojawia się komunikat o niemożliwości usunięcia.

3.5 Rejestracja



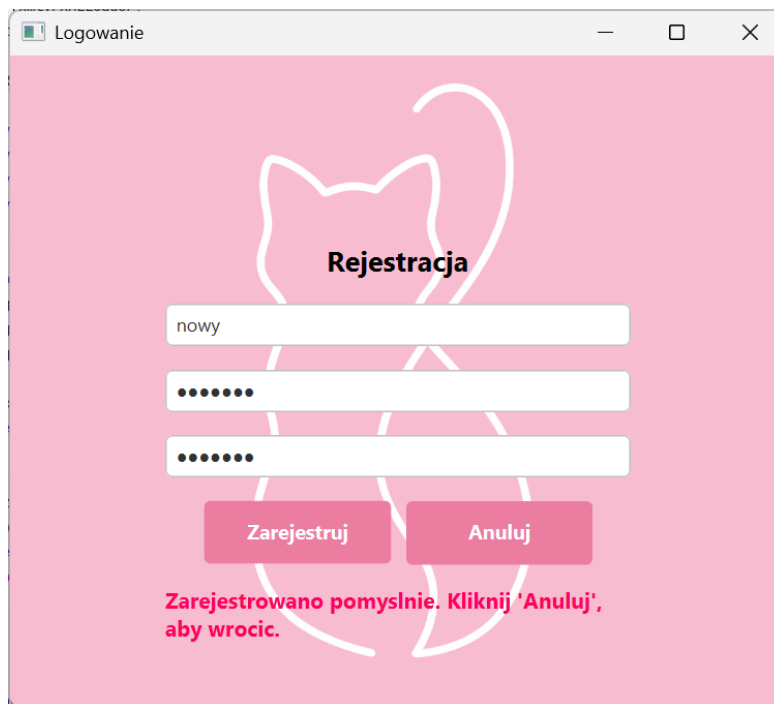
The image shows a web application window titled "Logowanie". The background is pink with a large white outline of a cat. In the center, the word "Rejestracja" is displayed in bold black text. Below it are three white input fields with labels "Login", "Hasło", and "Powtorz hasło". At the bottom, there are two pink buttons: "Zarejestruj" and "Anuluj".

Rysunek 39: Wygląd widoku rejestracji



The image shows the same registration form as in Figure 39, but with an error message. The input fields are now labeled "uzytkownik", and the password fields are filled with dots. Below the buttons, the text "Taki login juz istnieje." is displayed in red. The window title "Logowanie" and the cat background remain the same.

Rysunek 40: Wygląd widoku rejestracji dla istniejącego już takiego użytkownika



Rysunek 41: Wygląd widoku rejestracji dla poprawnie zarejestrowanego użytkownika

```

18  @FXML
19  private void zarejestruj() {
20      String login = loginField.getText().trim();
21      String haslo = hasloField.getText().trim();
22      String powtorz = powtorzHasloField.getText().trim();
23
24      if (!czyPoprawneDane(login, haslo, powtorz)) {
25          return;
26      }
27
28      if (UzytkownikDAO.zarejestruj(login, haslo)) {
29          komunikatLabel.setText("Zarejestrowano pomyslnie. Kliknij 'Anuluj', aby wrocic.");
30      } else {
31          komunikatLabel.setText("Taki login juz istnieje.");
32      }
33
34  }

```

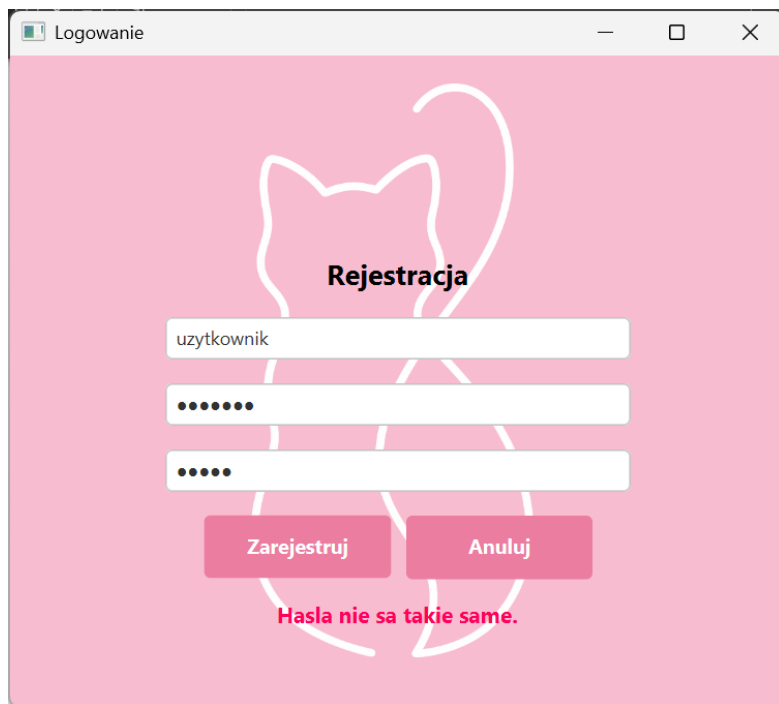
Rysunek 42: Metoda `zarejestruj()` odczytuje dane z formularza rejestracji i przekazuje je do metody walidującej. Jeżeli dane są poprawne i login nie jest jeszcze zajęty, następuje zapis użytkownika do bazy. Po udanej rejestracji wyświetlany jest komunikat potwierdzający, w przeciwnym wypadku – informacja o istnieniu konta z takim loginem.

The image shows a web browser window titled "Logowanie". The background is pink with a white cat silhouette. The title "Rejestracja" is centered. There are three input fields: the first contains "uzytkownik()", the second and third contain seven dots. Below the fields are two buttons: "Zarejestruj" and "Anuluj". A red error message is displayed at the bottom: "Login może zawierać litery, cyfry, kropki, myślniki (3-20 znaków)."

Rysunek 43: Obsługa błędu, gdy login nie spełnia założeń

The image shows a web browser window titled "Logowanie". The background is pink with a white cat silhouette. The title "Rejestracja" is centered. There are three input fields: the first contains "uzytkownik", the second contains five dots, and the third contains six dots. Below the fields are two buttons: "Zarejestruj" and "Anuluj". A red error message is displayed at the bottom: "Hasło musi mieć min. 6 znaków i zawierać cyfry."

Rysunek 44: Obsługa błędu, gdy hasło nie spełnia założeń



Rysunek 45: Obsługa błędu, gdy hasła nie są takie same

```

35
36 @
37 private boolean czyPoprawneDane(String login, String haslo, String powtorzHaslo) { 1 usage
38     if (login.isEmpty() || haslo.isEmpty() || powtorzHaslo.isEmpty()) {
39         komunikatLabel.setText("Wszystkie pola sa wymagane.");
40         return false;
41     }
42     if (!login.matches( regex: "[A-Za-z0-9._-]{3,20}")) {
43         komunikatLabel.setText("Login moze zawierac litery, cyfry, kropki, myslniki (3-20 znakow).");
44         return false;
45     }
46
47     if (haslo.length() < 6 || !haslo.matches( regex: ".*\\d.*")) {
48         komunikatLabel.setText("Haslo musi miec min. 6 znakow i zawierac cyfre.");
49         return false;
50     }
51
52     if (!haslo.equals(powtorzHaslo)) {
53         komunikatLabel.setText("Hasla nie sa takie same.");
54         return false;
55     }
56
57     return true;
58 }

```

Rysunek 46: Metoda sprawdza poprawność danych wpisanych przy rejestracji: czy pola nie są puste, login ma od 3 do 20 znaków (dozwolone litery, cyfry i symbole), hasło ma minimum 6 znaków i zawiera cyfrę oraz czy podane hasła są identyczne. W przypadku błędu użytkownik otrzymuje informację w interfejsie.

4. Interfejs graficzny

4.1 CSS

```
1  .root {
2      -fx-font-family: "Segoe UI", sans-serif;
3      -fx-background-color: #F8BCD0FF;
4      -fx-background-image: url("/images/kot.png");
5      -fx-background-repeat: no-repeat;
6      -fx-background-position: center;
7      -fx-background-size: 450px;
8  }
9
10 .label {
11     -fx-font-size: 14px;
12     -fx-text-fill: #000000;
13 }
14
15 .button {
16     -fx-background-color: rgb(234, 125, 160);
17     -fx-text-fill: rgb(255, 255, 255);
18     -fx-font-size: 13px;
19     -fx-font-weight: bold;
20     -fx-border-radius: 4px;
21     -fx-pref-width: 120px;
22     -fx-pref-height: 40px;
23     -fx-background-radius: 4px;
24     -fx-padding: 6 12;
25 }
26
27 .button:hover {
28     -fx-background-color: #a2c2e7;
29 }
30
31 .table-view {
32     -fx-border-color: #f1dae1;
33     -fx-background-color: #fafafa;
34     -fx-table-cell-border-color: #e0e0e0;
35 }
36
37 .text-field, .password-field, .combo-box {
38     -fx-pref-width: 220px;
39     -fx-padding: 5px;
40     -fx-background-color: rgb(255, 255, 255);
41     -fx-border-color: #bfbfbf;
42     -fx-border-radius: 4px;
43     -fx-background-radius: 4px;
44 }
45
46 .table-column {
47     -fx-alignment: CENTER_LEFT;
48 }
49
50 .label#komunikatLabel {
51     -fx-text-fill: #ff005e;
52     -fx-font-weight: bold;
53 }
```

Rysunek 47: Plik `style.css` definiuje wygląd graficzny komponentów JavaFX w całej aplikacji. Określa kolorystykę, czcionki, rozmiary oraz styl interaktywnych elementów, takich jak przyciski, pola tekstowe czy tabela. Umożliwia zastosowanie spójnej i estetycznej szaty graficznej. Przykładowo, `.root` ustawia tło i obraz, `.button:hover` zmienia kolor przycisku po najechaniu, a `.label#komunikatLabel` wyróżnia komunikaty błędów na czerwono.

4.2 Logowanie.fxml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.geometry.Insets?>
4 <?import javafx.scene.control.*?>
5 <?import javafx.scene.layout.*?>
6
7 <VBox xmlns="http://javafx.com/javafx"
8       xmlns:fx="http://javafx.com/fxml"
9       fx:controller="projektFilmy.Kontroler.LogowanieKontroler"
10      spacing="15"
11      alignment="CENTER"
12      stylesheets="@style.css"
13      prefWidth="500" prefHeight="420">
14
15   <padding>
16       <Insets top="100" bottom="20" left="100" right="100"/>
17   </padding>
18
19   <Label text="Logowanie" style="-fx-font-size: 18px; -fx-font-weight: bold;"/>
20
21   <TextField fx:id="loginField" promptText="Login"/>
22   <PasswordField fx:id="hasloField" promptText="Hasło"/>
23
24   <HBox spacing="10" alignment="CENTER">
25       <Button text="Zaloguj" onAction="#zaloguj"/>
26       <Button text="Zarejestruj sie" onAction="#zarejestruj"/>
27   </HBox>
28
29   <Label fx:id="komunikatLabel" wrapText="true"/>
30
31 </VBox>
32
```

Rysunek 48: Widok logowania zbudowany w technologii FXML. Zawiera pola do wprowadzania loginu i hasła, dwa przyciski do logowania i rejestracji oraz komunikat informacyjny. Powiązany z klasą `LogowanieKontroler`, a wygląd jest częściowo definiowany przez zewnętrzny plik CSS (`style.css`). Layout oparty na kontenerze `VBox` z wyrównaniem i marginesami.

5. Repozytorium Projekt

GitHub: <https://github.com/a-0130/Filmy.git>