Bazy danych - Projekt

Filmowa baza danych z interaktywną aplikacją

Maja Frankowska, Tomasz Kąkol, Natalia Pospiech

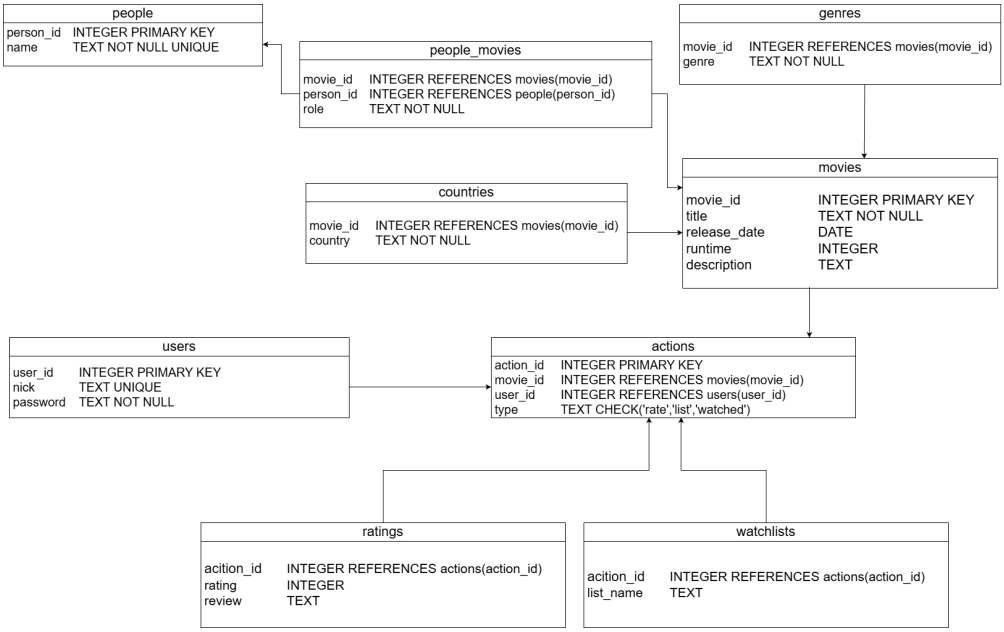
Spis treści

Opis projektu

Celem projektu było utworzenie interaktywnej aplikacji służącej do katalogowania, przeglądania i oceniania filmów kinowych. Zostało to zrealizowane poprzez połączenie relacyjnej bazy danych (napisanej w języku *SQL*) z interfejsem (napisanym w języku *Python*) umożliwiającym w sposób prosty i intuicyjny obsługę oraz edycję bazy. Korzystający z aplikacji użytkownik ma między innymi możliwość: tworzenia własnych list filmów, w których może przechochowywać swoje ulubione produkcje; dokumentowania obejrzanych przez siebie filmów; wystawiania recenzji i ocen; przeszukiwania bazy filmowej w oparciu o konkretne cechy takie jak kraj produkcji czy gatunek; czytania recenzji innych użytkowników oraz przeglądania wystawionych przez nich ocen; dodawania nowych filmów bezpośrednio do bazy wszystkich filmów (jako admin).

Opis bazy danych

Baza danych wykorzystana przy implementacji projektu została napisana w języku *SQL*. Składa się na nią 9 tabel przechowujących dane (o filmach i użytkownikach aplikacji) oraz 22 funkcje i 2 wyzwalacze, które wykorzystywane są przy jej obsłudze. Poniżej przedstawiamy diagram relacji tabel oraz objaśniamy sens i wykorzystanie każdej z nich.



(i) Tabela ‘people’ zawiera dane osób pracujących przy produkcji przynajmniej jednego filmu znajdującego się w bazie.

(ii) Tabela ‘people_movies’ przechowuje informacje o funkcji osoby w danym filmie.

(iii) Tabele ‘genres’ i ‘countries’ przypisują odpowiednio gatunek i kraj produkcji konkretnemu filmowi.

(iv) Tabela ‘movies’ zawiera informacje o danym filmie.

(v) Tabela ‘users’ przechowuje dane użytkowników.

(vi) Tabela ‘actions’ zawiera informacje o aktywności użytkownika wobec danego filmu.

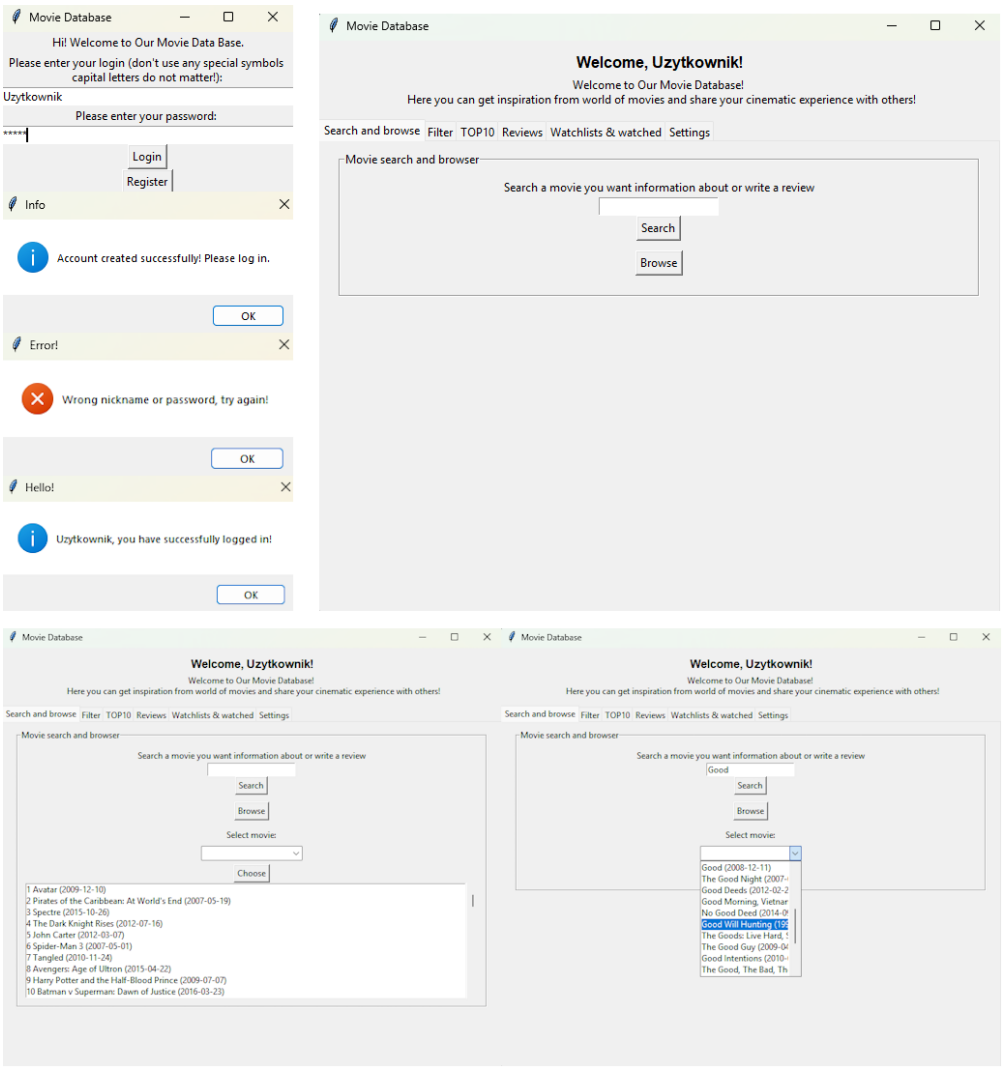
(vii) Tabela ‘ratings’ przechowuje szczegóły akcji dotyczących oceny lub recenzowania filmu.

(viii) Tabela ‘watchlists’ zawiera nazwy list, na których zostały umieszczone filmy w konkretnej akcji.

Opis aplikacji

Baza danych obsługiwana jest przez interaktywną aplikację zarówno w przypadku użytkownika, który korzysta z niej w celu katalogowania obejrzanych lub interesujących go produkcji, jak i w przypadku administratora bazy, który korzystając z aplikacji może dodawać nowe filmy. Jest ona przejrzysta, łatwa w obsłudze i intuicyjna - poniżej przedstawiamy jej główne funkcje wraz z odpowiednimi zrzutami ekranu.

Korzystanie z aplikacji rozpoczyna się od procesu logowania. Jeśli użytkownik nie założył wcześniej konta, musi dokonać rejestracji wpisując w odpowiednie pola login i hasło, które będą przypisane do jego konta. Po wykonaniu tych kroków, użytkownik wpisuje swój login i hasło w te same pola, tym razem wybierając opcję *login*. Jeśli wpisane dane są poprawne, tj. odpowiadają pewnemu zarejestrowanemu kontu, użytkownik uzyskuje dostęp do aplikacji - w przeciwnym wypadku musi spróbować zalogować się ponownie.

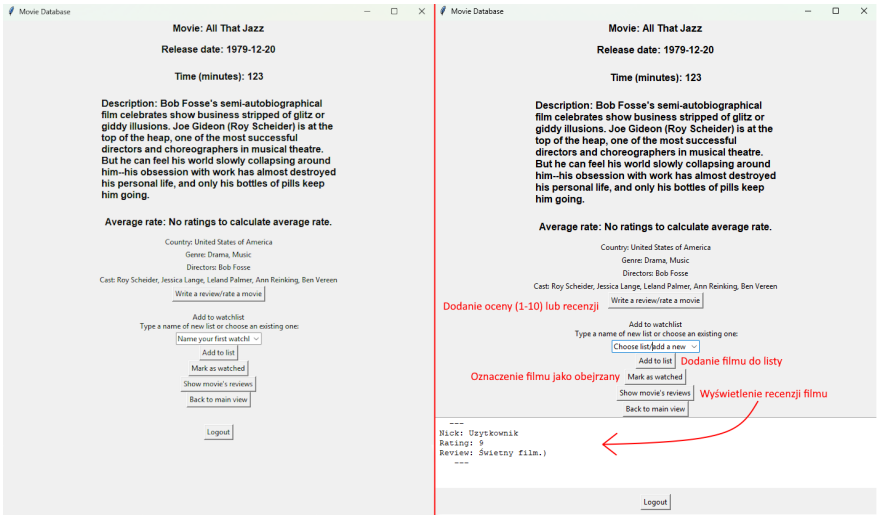


Po pomyślnym zalogowaniu użytkownik witany jest przez stronę główną aplikacji. Klikając na jeden z nagłówków *Filter*, *TOP 10*, *Reviews*, *Watchlists & watched* lub *Settings* użytkownik może nawigować po różnych funkcjach aplikacji - po kolei przedstawimy każdy z nich, rozpoczynając od korzystania ze strony głównej.

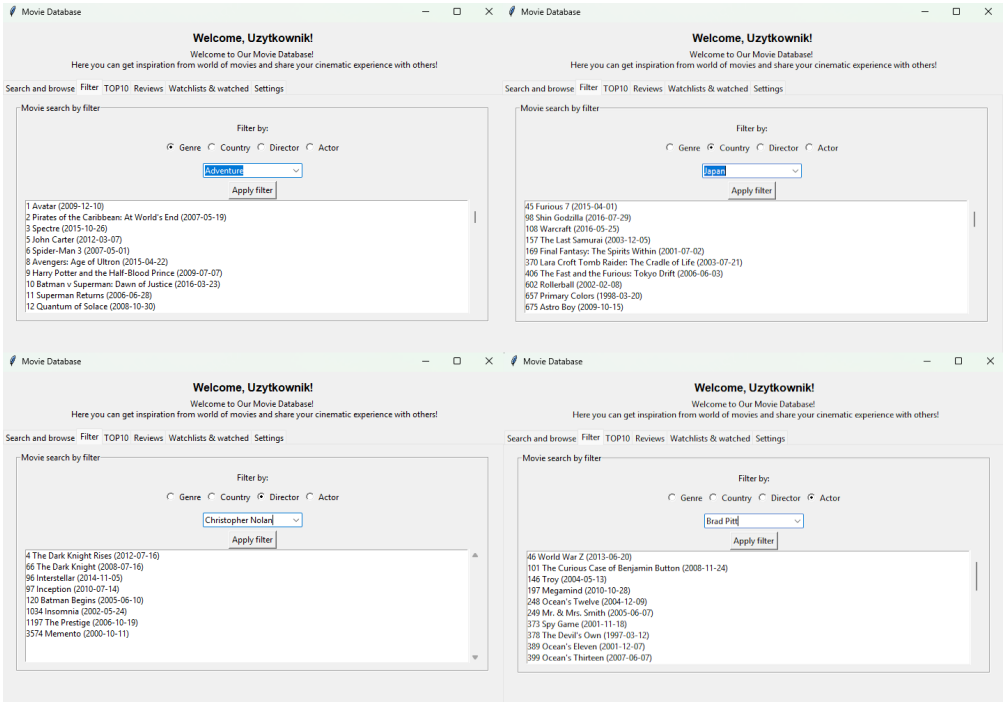
Wpisując tytuł (bądź tylko jego część) interesującego nas filmu w puste okienko tekstowe na stronie głównej i wybierając opcję *Search* wyświetlona zostaje lista filmów znajdujących się w bazie, których tytuły

odpowiadają wpisanej przez użytkownika frazie. Jeśli jeden z filmów znajdujących się na wyświetlonej liście zainteresuje użytkownika, może go kliknąć, aby otworzyć osobne okno z informacjami o filmie.

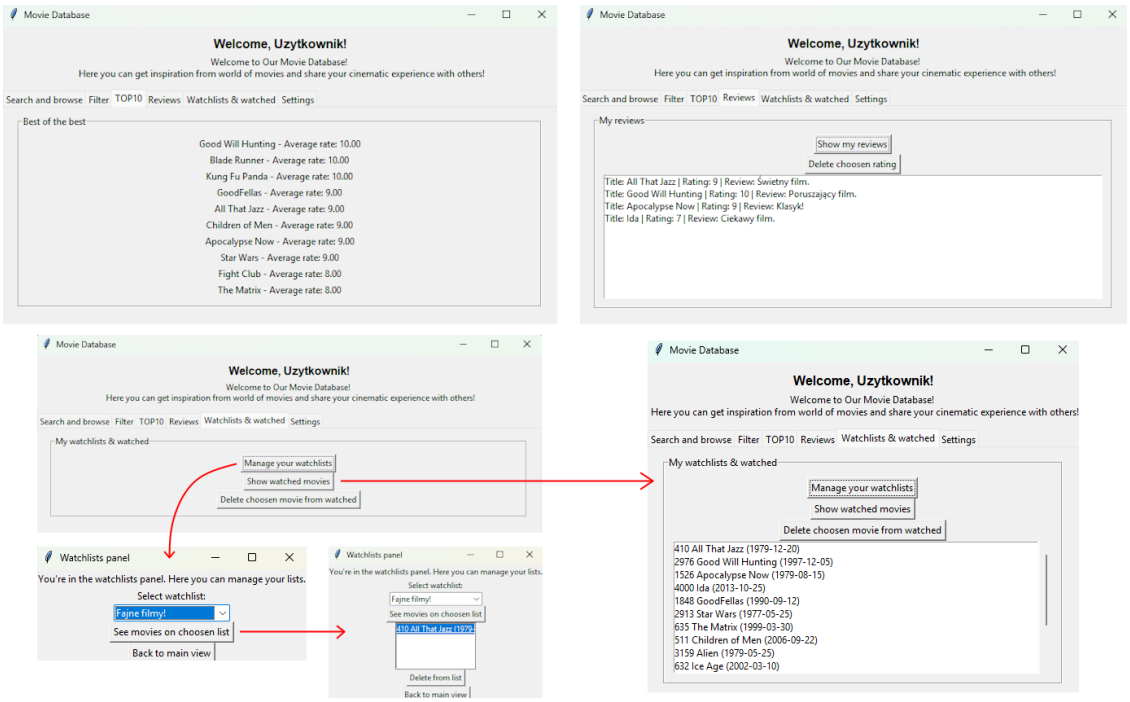
Alternatywną metodą wyszukiwania filmu jest opcja *Browse* - umożliwia ona przeszukiwanie całej bazy filmowej. Po wybraniu tej opcji wyświetlona zostaje lista wszystkich produkcji znajdujących się w bazie danych, posortowanych względem ich popularności (od najbardziej do najmniej popularnych).



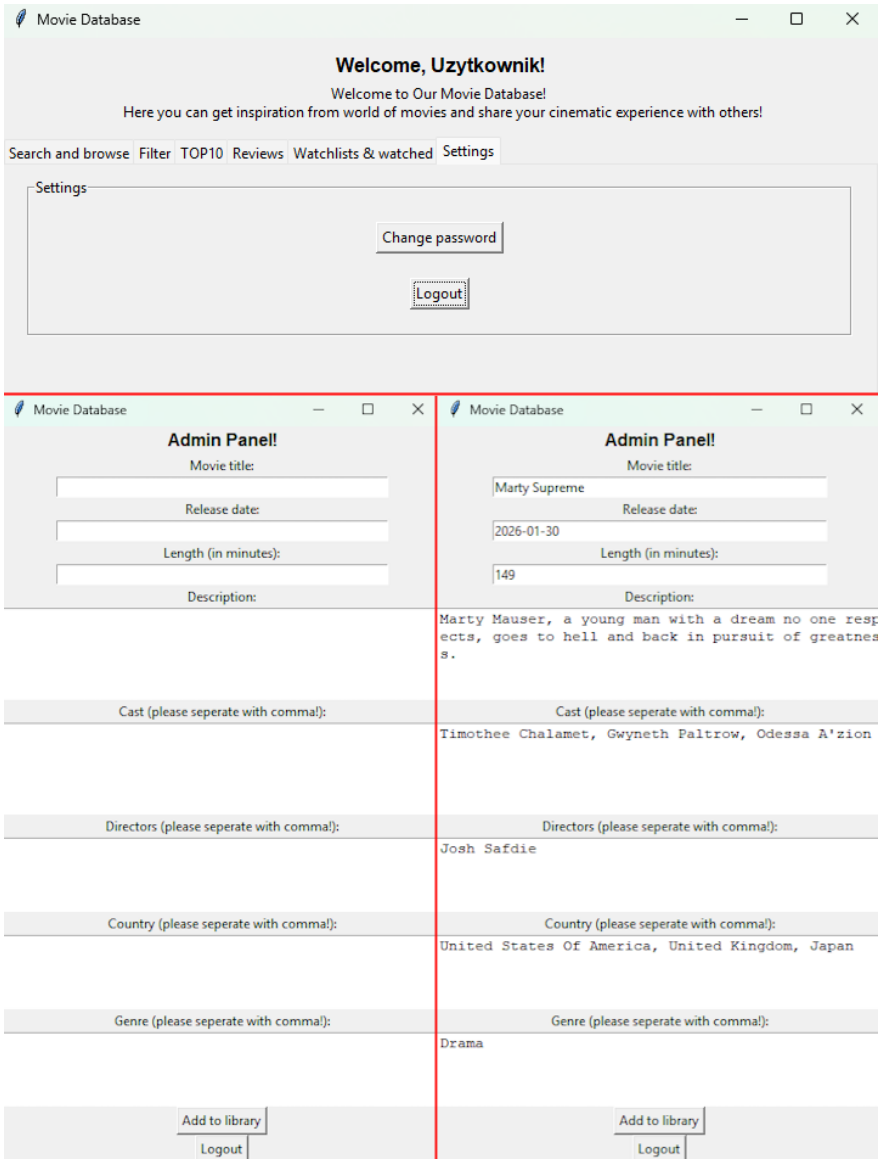
Po wybraniu filmu na stronie głównej otwierane jest osobne okno aplikacji - zawiera ono podstawowe informacje o wybranym filmie: tytuł, datę premiery, czas trwania, krótki opis fabuły, średnią ocenę, kraje produkcji, gatunki, dane reżysera/reżyserów i aktorów. Poza wyświetlaniem tych informacji, użytkownik ma również dostęp do następujących działań: dodanie oceny lub napisanie recenzji (*Write a review/rate a movie*), dodanie filmu do jednej z wybranych przez siebie list (*Add to list*), oznaczenie filmu jako obejrzany (*Mark as watched*) lub przeczytanie recenzji napisanych przez innych użytkowników (*Show movie's reviews*). Jeśli użytkownik chce powrócić do strony głównej wybiera opcję *Back to main view*, co skutkuje zamknięciem dodatkowego okna.



Wybierając nagłówek *Filter* użytkownik zostaje przeniesiony do części aplikacji, w której ma możliwość wyszukiwania filmów spełniających pewne cechy - może wybrać filmy z wybranego gatunku, nakręcone w wybranym kraju, wyreżyserowane przez wybranego reżysera lub takie, w których występuje wybrany aktor. Jeśli użytkownik znalazł na liście film, który go zainteresował, może kliknąć na niego, co w konsekwencji otworzy osobne okno z dodatkowymi informacjami o danym filmie (to opisane wcześniej przy stronie głównej).



Klikając w następny nagłówek, *TOP 10*, zostaje wyświetlona lista 10 filmów o największej średniej ocen (wystawionych przez innych użytkowników) ze wszystkich filmów znajdujących się w bazie danych. Wybór kolejnego nagłówka, *Reviews*, umożliwia użytkownikowi przejrzenie wystawionych przez siebie ocen i napisanych recenzji. Jeśli użytkownik stwierdzi, że jeden z wpisów jest nieaktualny lub chciałby go zmienić, może wybrać opcję *Delete chosen rating*, aby wybraną recenzję czy też ocenę usunąć. Przedostatni nagłówek, *Watchlists & watched* zawiera informacje i działania związane z listami oraz filmami obejrzanymi przez użytkownika. Po wybraniu opcji *Show watched movies* użytkownik może przejrzeć filmy, które uprzednio oznaczył jako obejrzane i ewentualnie je z takiej listy usunąć (opcja *Delete chosen movie from watched*). Jeśli użytkownik chce przejrzeć lub zedytować utworzone przez siebie listy filmów może w tym celu wybrać opcję *Manage your watchlists*. Aplikacja otwiera wówczas osobne okno, w którym użytkownik może wybrać jedną z należących do niego list, aby dokonać w niej ewentualnych zmian (np. usunąć jeden z filmów opcją *Delete from list*) lub po prostu przejrzeć jej zawartość (*See movies on chosen list*). Ostatni nagłówek, tj. *Settings*, umożliwia użytkownikowi zmianę hasła jego konta lub wylogowanie się z aplikacji.



W celu dodania filmu do bazy danych poprzez aplikację wymagane jest odpowiednie konto - administrator bazy. Aby się na nie zalogować, w polach loginu i hasło wpisujemy kolejno ADMIN i admin. Następnie zostanie otwarte okno, które służy do dodawania nowych produkcji - jeśli uzupełnimy w sposób poprawny informacje o filmie, który chcemy dodać do bazy, zostanie on automatycznie do niej wpisany.

Kod

W tej części przedstawiamy kod wykorzystany przy tworzeniu bazy danych oraz aplikacji umożliwiającej jej obsługę. W pierwszej części wypisujemy kod dotyczący bazy danych (*SQL*) wraz z krótkim komentarzem, a w drugiej kod związany z interfejsem (*Python*).

Baza danych - SQL

Relacje

Kod dotyczący tworzenia tabel w bazie danych:

```
CREATE TABLE people(
    person_id   INTEGER GENERATED BY DEFAULT AS identity (start with 2000000) PRIMARY KEY,
    name        TEXT NOT NULL UNIQUE
);

CREATE TABLE movies(
    movie_id            INTEGER GENERATED BY DEFAULT AS
                  identity (start with 5000) PRIMARY KEY,
    title               TEXT NOT NULL,
    release_date        DATE,
    runtime             INTEGER,
    description         TEXT
);

CREATE TABLE genres(
    movie_id        INTEGER references movies(movie_id) on delete cascade,
    genre           TEXT not null
);

create table countries(
    movie_id        INTEGER references movies(movie_id) on delete cascade,
    country         TEXT not null
);

CREATE TABLE people_movies(
    movie_id        INTEGER REFERENCES movies(movie_id) on delete cascade,
    person_id       INTEGER REFERENCES people(person_id) on delete cascade,
    role            TEXT not NULL
);

create table users(
    user_id     INTEGER GENERATED BY DEFAULT AS identity primary key,
    nick        TEXT not null UNIQUE,
    password    TEXT not null
);

create table actions(
    action_id       INTEGER GENERATED BY DEFAULT AS identity primary key,
    movie_id        INTEGER REFERENCES movies(movie_id) on delete set null,
    user_id         INTEGER REFERENCES users(user_id) on delete set null,
    type            TEXT CHECK(type in ('rate','list','watched'))
);

create table ratings(
    action_id       INTEGER references actions(action_id) on delete cascade,
    rating          INTEGER,
    review          TEXT
);
```

```
create table watchlists(
    action_id       INTEGER references actions(action_id) on delete cascade,
    list_name       TEXT
);
```

Funkcje

Funkcje grupujące (odpowiednio: obsada filmowa, filmy z danym aktorem, reżyserzy danego filmu, filmy danego reżysera, filmy z danego państwa, filmy danego gatunku):

```
create or replace function movie_cast(t INTEGER) returns setof TEXT as $$
BEGIN
    RETURN QUERY
    select p.name
    from people p natural join people_movies pm natural join movies m
    where m.movie_id = t AND pm.role LIKE 'Actor';
END;
$$ language 'plpgsql';


create or replace function actor_movies(t TEXT)
  returns TABLE(m_id INTEGER, m_title TEXT, m_year DATE) as $$
BEGIN
    RETURN QUERY
    select m.movie_id, m.title, m.release_date
    from people p natural join people_movies pm natural join movies m
    where p.name ILIKE t AND pm.role LIKE 'Actor';
END;
$$ language 'plpgsql';


create or replace function movie_director(t INTEGER) returns setof TEXT as $$
BEGIN
    RETURN QUERY
    select p.name
    from people p natural join people_movies pm natural join movies m
    where m.movie_id = t AND pm.role LIKE 'Director';
END;
$$ language 'plpgsql';


create or replace function director_movies(t TEXT)
  returns TABLE(m_id INTEGER, m_title TEXT, m_year DATE) as $$
BEGIN
    RETURN QUERY
    select m.movie_id, m.title, m.release_date
    from people p natural join people_movies pm natural join movies m
    where p.name ILIKE t AND pm.role LIKE 'Director';
END;
$$ language 'plpgsql';
```

```
create or replace function movies_from(t TEXT)
  returns TABLE(m_id INTEGER, m_title TEXT, m_year DATE) as $$
BEGIN
    RETURN QUERY
    select m.movie_id, m.title, m.release_date
    from countries c natural join movies m
    where c.country ILIKE t;
END;
$$ language 'plpgsql';


create or replace function genre_movies(t TEXT)
  returns TABLE(m_id INTEGER, m_title TEXT, m_year DATE) as $$
BEGIN
    RETURN QUERY
    select m.movie_id, m.title, m.release_date
    from genres g natural join movies m
    where g.genre ILIKE t;
END;
$$ language 'plpgsql';
```

Funkcja tworząca połączenie między osobą a filmem oraz funkcja dodająca nowy film do bazy (opcja dla administratora):

```
create or replace function add_pm(
    movie_id INTEGER,
    actor TEXT,
    role TEXT
) returns VOID AS $$
DECLARE
    p_id INTEGER;
BEGIN
    INSERT INTO people (name) VALUES (actor) ON CONFLICT (name) DO NOTHING;
    SELECT person_id INTO p_id FROM people WHERE name = actor;
    INSERT INTO people_movies VALUES (movie_id, p_id, role);
END;
$$ language 'plpgsql';

create or replace function add_movie(
    title TEXT, release_date DATE,
    runtime INTEGER, description TEXT,
    actors TEXT[], directors TEXT[],
    country_l TEXT[], genre_l TEXT[]
    ) returns VOID as $$
DECLARE
    m_id INTEGER;
    director TEXT;
    actor TEXT;
    country TEXT;
    genre TEXT;
BEGIN
    INSERT INTO movies (title,release_date,runtime,description)
    VALUES (title,release_date,runtime,description)
```

```
    RETURNING movie_id INTO m_id;
    FOREACH director IN ARRAY directors LOOP
        PERFORM add_pm(m_id, director, 'Director');
    END LOOP;
    FOREACH actor IN ARRAY actors LOOP
        PERFORM add_pm(m_id, actor, 'Actor');
    END LOOP;
    FOREACH country IN ARRAY country_l LOOP
        INSERT INTO countries VALUES (m_id, country);
    END LOOP;
    FOREACH genre IN ARRAY genre_l LOOP
        INSERT INTO genres VALUES (m_id, genre);
    END LOOP;
END;
$$ language 'plpgsql';
```

Funkcje umożliwiające wykonanie akcji zalogowanemu użytkownikowi (odpowiednio: oznaczenie filmu jako obejrzany, ocena filmu, dodanie filmu do listy, zmiana hasła, usunięcie filmu z listy, usunięcie oceny filmu, odznaczenie filmu jako obejrzany).

```
create or replace function mark_as_watched(
    m_id INTEGER, u_id INTEGER
    ) returns VOID AS $$
BEGIN
    INSERT INTO actions (movie_id, user_id, type) VALUES (m_id, u_id,'watched');
END;
$$ language 'plpgsql';


create or replace function rate(
    m_id INTEGER, u_id INTEGER,
    rating_v DECIMAL, review_v TEXT
    ) returns VOID AS $$
DECLARE
    a_id INTEGER;
BEGIN
    INSERT INTO actions (movie_id, user_id, type) VALUES (m_id, u_id, 'rate')
    RETURNING action_id INTO a_id;
    INSERT INTO ratings(action_id,rating,review) VALUES(a_id,rating_v,review_v);
    IF NOT EXISTS
    (SELECT 1 FROM actions WHERE movie_id = m_id AND user_id = u_id AND type = 'watched')
      THEN
        INSERT INTO actions (movie_id, user_id, type) VALUES (m_id, u_id, 'watched');
    END IF;
END;
$$ language 'plpgsql';


create or replace function add_to_list(
    m_id INTEGER, u_id INTEGER,
    l_name TEXT
    ) returns VOID AS $$
DECLARE
```

```
    a_id INTEGER;
BEGIN
    INSERT INTO actions (movie_id, user_id, type) VALUES (m_id, u_id,'list')
    RETURNING action_id INTO a_id;
    INSERT INTO watchlists(action_id,list_name) VALUES(a_id,l_name);
END;
$$ language 'plpgsql';


create or replace function change_password(u_id INTEGER, new_password TEXT)
  returns TEXT as $$
BEGIN
    update users set password = new_password where user_id = u_id;
    RETURN 'Hasło zostało zmienione.';
END;
$$ language 'plpgsql';


create or replace function delete_from_list(u_id INTEGER, m_id INTEGER, l_name TEXT)
  returns TEXT as $$
begin
    delete from actions
    where user_id = u_id and movie_id = m_id
    and action_id in (SELECT action_id FROM watchlists WHERE list_name = l_name);
    return 'Usunięto film z listy.';
end;
$$ language 'plpgsql';

create or replace function delete_rating(u_id INTEGER, m_id INTEGER) returns TEXT as $$
begin
    delete from actions
    where user_id = u_id and movie_id = m_id
    and action_id in (SELECT action_id FROM ratings);
    return 'Usunięto ocenę filmu';
end;
$$ language 'plpgsql';


create or replace function delete_watched(u_id INTEGER, m_id INTEGER) returns VOID as $$
begin
    delete from actions
    where user_id = u_id and movie_id = m_id and type = 'watched';
end;
$$ language 'plpgsql';
```

Funkcje zwracające odpowiednio: filmy obejrzane przez użytkownika, opinie wystawione przez użytkownika, oceny wystawione przez użytkownika, opinie filmu, oceny filmu, średnią ocenę filmu.

```
create or replace function show_watched(u_id INTEGER)
RETURNS TABLE (movie_id INT, title TEXT, release_date DATE)
AS $$
BEGIN
    RETURN QUERY
```

```
    SELECT a.movie_id, m.title, m.release_date
    FROM actions a JOIN movies m USING (movie_id)
    WHERE a.user_id = u_id
      AND a.type = 'watched';
END;
$$ LANGUAGE plpgsql;



create or replace function show_user_reviews(u_id INTEGER)
RETURNS TABLE (movie_id INT, title TEXT, rating INT, review TEXT)
AS $$
BEGIN
    RETURN QUERY
    SELECT a.movie_id, m.title, r.rating, r.review
    FROM actions a JOIN ratings r ON a.action_id = r.action_id
    JOIN movies m ON a.movie_id = m.movie_id
    WHERE a.user_id = u_id;
END;
$$ LANGUAGE plpgsql;



create or replace function show_user_ratings(u_id INTEGER)
RETURNS TABLE (movie_id INT, title TEXT, rating INT)
AS $$
BEGIN
    RETURN QUERY
    SELECT a.movie_id, m.title, r.rating
    FROM actions a JOIN ratings r ON a.action_id = r.action_id
    JOIN movies m ON a.movie_id = m.movie_id
    WHERE a.user_id = u_id;
END;
$$ LANGUAGE plpgsql;


create or replace function show_movie_reviews(m_id INTEGER)
RETURNS TABLE (user_id INT, user_nick TEXT, rating INT, review TEXT)
AS $$
BEGIN
    RETURN QUERY
    SELECT a.user_id, u.nick, r.rating, r.review
    FROM actions a JOIN ratings r ON a.action_id = r.action_id
    JOIN users u ON a.user_id = u.user_id
    WHERE a.movie_id = m_id;
END;
$$ LANGUAGE plpgsql;


create or replace function show_movie_ratings(m_id INTEGER)
RETURNS TABLE (user_id INT, user_nick TEXT, rating INT)
AS $$
BEGIN
```

```
    RETURN QUERY
    SELECT a.user_id, u.nick, r.rating
    FROM actions a JOIN ratings r ON a.action_id = r.action_id
    JOIN users u ON a.user_id = u.user_id
    WHERE a.movie_id = m_id;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION movie_avg(m_id INTEGER) RETURNS DECIMAL AS $$
DECLARE
    avg_score DECIMAL(4, 2);
BEGIN
    SELECT avg(rating)::DECIMAL(4,2) INTO avg_score
    FROM actions JOIN ratings USING (action_id)
    WHERE movie_id = m_id;
    RETURN avg_score;
END;
$$ LANGUAGE 'plpgsql';
```

Wyzwalacze, widoki

Wyzwalacz uniemożliwiający dodanie filmu, który już jest w bazie. Uruchamia się przed dodaniem nowego filmu:

```
CREATE OR REPLACE FUNCTION prevent_duplicate_film_in_database()
RETURNS trigger AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM movies
        WHERE title = NEW.title
          AND release_date = NEW.release_date
    ) THEN
        RAISE EXCEPTION
            'Film "%" już istnieje w bazie',
            NEW.title;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_prevent_duplicate_film_in_database
BEFORE INSERT ON movies
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_film_in_database();
```

Wyzwalacz uniemożliwiający dwukrotne oznaczenie filmu jako obejrzany. Uruchamia się przed dodaniem nowej akcji:

```
CREATE OR REPLACE FUNCTION prevent_duplicate_watched_film()
RETURNS trigger AS $$
```

```
BEGIN
    IF  NEW.type = 'watched'
    AND EXISTS (
            SELECT 1
            FROM actions
        WHERE movie_id = NEW.movie_id
        AND user_id = NEW.user_id
        AND type = 'watched'
    ) THEN
        RAISE EXCEPTION
            'Ten film już jest oznaczony jako obejrzany';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_prevent_duplicate_watched_film
BEFORE INSERT ON actions
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_watched_film();
```

Widok przedstawiający 10 najlepiej ocenionych filmów zamieszczonych w bazie:

```
create view top_10 as
    select m.title, m.release_date, m.runtime, r.average_rating
    from movies m
    join  (select movie_id, avg(rating)::DECIMAL(4,2) as average_rating
            from actions natural join ratings
            group by movie_id) r using (movie_id)
    order by r.average_rating desc
    limit 10;
```

Aplikacja - Python

Ze względu na długość skryptu odpowiedzialnego za działanie interfejsu, poniżej zamieszczamy jedynie opis zawartości plików składających się na całość programu oraz listę najważniejszych wykorzystanych modułów.

Wykorzystane moduły: *psycopg2* - możliwość połączenia i obsługi bazy danych przy użyciu *Pythona*, *tkinter* - stworzenie interaktywnej aplikacji i jej oprawa graficzna.

Pliki:

(i) main.py - główny skrypt umożliwiający uruchomienie aplikacji;

(ii) server.py - łączenie z bazą danych w *PostgreSQL*;

(iii) controllers.py - funkcje oraz klasy wykorzystywane w aplikacji;

(iv) views.py - wyświetlanie i wygląd interfejsu;

(v) sql_functions.py - wykonywanie poleceń związanych bezpośrednio z bazą danych przez *Pythona*;

(vi) import_danych.py - skrypt umożliwiający wstawienie zbioru danych filmowych do istniejącej bazy w *PostgreSQL*.