



**WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA**
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Programowanie

Oleksandr Bubnov
Nr albumu studenta w68135

*Aplikacja desktopowa do przeglądania
multimedów*

Prowadzący: dr Marek Giebułtowski

Praca projektowa Szkolenie Techniczne 1

Rzeszów 2024

Spis treści

Wstęp	5
1 Opis założeń projektu	6
1.1 Cele projektu	6
1.2 Wymagania funkcjonalne i niefunkcjonalne	7
1.2.1 Opis wymagań funkcjonalnych	7
1.2.2 Opis wymagań niefunkcjonalnych	8
2 Opis struktury projektu	9
2.1 Wykorzystywany język i narzędzia	9
2.2 Hierarchia klas i opis metod	10
2.2.1 Klasa Banner	10
2.2.2 Klasa DatabaseDriver	12
2.2.3 Klasa User	13
2.2.4 Klasa UserProperties	14
2.2.5 Klasa Model	15
2.2.6 Klasa MediaPage	17
2.2.7 Klasa ViewFactory	18
2.2.8 Wyliczenie LeftPanelOptions	19
2.2.9 Wyliczenie DatabaseTables	20
2.2.10 Klasa CenterPanelController	21
2.2.11 Klasa LeftPanelController	21
2.2.12 Klasa TopPanelController	23
2.2.13 Klasa BrowseSectionController	23
2.2.14 Klasa YourLibrarySectionController	24
2.2.15 Klasa MediaPageController	25
2.2.16 Klasa LogInWindowController	27
2.2.17 Klasa MainWindowController	28
2.2.18 Klasa App	29
2.3 Baza danych i zarządzanie danymi	30
2.4 Minimalne wymagania sprzętowe	32
3 Prezentacja warstwy użytkowej projektu	33
3.1 Okno logowania/rejestracji	33
3.1.1 Logowanie	33
3.1.2 Rejestracja	34
3.2 Główne okno aplikacji	36
3.3 Górnny panel aplikacji	36
3.3.1 Przycisk panelu centralnego	36
3.3.2 Pole wyszukiwania multimedialów	37
3.3.3 Przycisk profilu użytkownika	37
3.4 Lewy panel aplikacji	38

3.4.1	Sekcja "Browse"	38
3.4.2	Sekcja "Your library"	40
3.4.3	Sekcja "Other"	41
3.5	Strona multimedialna	42
3.5.1	Przyciski do dodawania multimediiów do biblioteki użytkownika	43
3.5.2	Odtwarzacz multimedialny	44
4	Podsumowanie	47
Bibliografia		48
Spis rysunków		49

Wstęp

Współczesny styl życia często stawia przed nami wiele codziennych zadań, które napotykamy w różnych okolicznościach, niezależnie od tego, czy jesteśmy na spacerze, w podróży, czy nawet w pracy. Często te zadania, choć pozornie nieistotne, mogą być ważnym czynnikiem wpływającym na ogólną satysfakcję i komfort osoby. Nawet taki aspekt jak oglądanie ulubionego filmu, serialu czy anime może mieć znaczący wpływ na nasz nastrój i ogólne samopoczucie. Źle zorganizowana biblioteka multimediiów, niewygodny interfejs lub błędy w odtwarzaniu mogą zniechęcić użytkownika i sprawić, że oglądanie będzie nieprzyjemne.

Niezawodnym rozwiązaniem tego problemu jest aplikacja desktopowa do oglądania multimediiów. Została ona zaprojektowana w celu wyeliminowania tych codziennych kłopotów, zapewniając wygodną i wydajną obsługę, a także ułatwiając znajdowanie ulubionych filmów, seriali lub anime i zarządzanie nimi w celu zapewnienia jak najprzyjemniejszego oglądania.

Rozdział 1

Opis założeń projektu

1.1 Cele projektu

- **Jaki jest cel projektu?** Celem projektu jest stworzenie aplikacji desktopowej do przeglądania multimedialnych przy użyciu JavaFX - platformy programistycznej Java dla GUI. Aplikacja powinna zapewniać prostą i wygodną usługę przeglądania, wyszukiwania multimedialnych i zarządzania biblioteką użytkownika.
- **Jaki jest problem?** Problemem jest brak przyjaznych dla użytkownika i prostych aplikacji na rynku IT. Istniejące aplikacje mogą być niewygodne w użyciu, zawodne lub nieuczciwe wobec użytkowników. Przyczyną tego zjawiska może być niekompetencja programistów, całkowity brak zainteresowania ich tworzeniem lub interes własny producentów takich usług.
- **Dlaczego ten problem jest ważny?** Problem ten jest istotny, ponieważ w dzisiejszym świecie rośnie zapotrzebowanie użytkowników na prostotę, wygodę i bezpieczeństwo. Problem ten obejmuje trudności w zarządzaniu interfejsem, brak wygodnego przeglądania multimedialnych oraz luki/naruszenia danych/praw użytkownika.
- **Co jest niezbędne, aby problem został rozwiązany?** Aby rozwiązać ten problem, konieczne jest stworzenie aplikacji, która zapewni użytkownikowi przyjazny i prosty interfejs do przeglądania, a także niezawodność danych i zgodność z zasadami prywatności.
Wykorzystanie platformy JavaFX, bazy danych SQLite, zestawów narzędzi do stylizacji i innych bibliotek jest niezbędne do stworzenia takiej aplikacji.
- **W jaki sposób problem zostanie rozwiązany?** Problem zostanie rozwiązany poprzez stworzenie aplikacji desktopowej na platformie JavaFX. Aplikacja będzie rozwijana krok po kroku, za każdym razem rozszerzając swoją funkcjonalność:
 1. Przygotowanie wstępnej struktury projektu, planu rozwoju, przygotowanie środowiska programistycznego;
 2. Tworzenie GUI dla okna logowania/rejestracji i głównego okna aplikacji;
 3. Wypełnienie głównego okna elementami graficznymi i różnymi sekcjami, z którymi użytkownik będzie wchodził w interakcje;
 4. Zapewnienie funkcjonalności okna logowania/rejestracji, głównego okna aplikacji i innych elementów pomocniczych;
 5. Utworzenie i połączenie bazy danych z GUI;
 6. Stylizacja aplikacji;
 7. Modyfikacje, poprawki błędów i finalizacja funkcjonalności całej aplikacji.

Rezultatem projektu będzie w pełni funkcjonalna aplikacja desktopowa, spełniająca wszystkie określone wymagania.

1.2 Wymagania funkcjonalne i niefunkcjonalne

W tej sekcji opisane zostaną wymagania funkcjonalne i niefunkcjonalne aplikacji.

1.2.1 Opis wymagań funkcjonalnych

- **Okno logowania/rejestracji.** Po uruchomieniu programu powinno pojawić się okno logowania/rejestracji. Domyślnie użytkownik jest proszony o zalogowanie się do aplikacji poprzez wprowadzenie nazwy użytkownika i hasła. Jeśli konto jeszcze nie istnieje, można je utworzyć, przechodząc do sekcji rejestracji konta. Podczas logowania do głównego okna aplikacji użytkownik może między innymi wybrać opcję "Remember me".
- **Główne okno aplikacji.** Po pomyślnym zalogowaniu lub utworzeniu konta otwiera się główne okno aplikacji, które zawiera centralny, lewy i górny panel sterowania.
- **Centralny panel.** Domyślnie, po zalogowaniu, centralny panel powinien wyświetlać trzy sekcje polecanych banerów. Każda sekcja banerów powinna reprezentować różne rodzaje multimediów: anime, filmy i seriale telewizyjne. Kliknięcie banera powoduje otwarcie strony multimedialnej w centralnym panelu.
- **Lewy panel.** Panel ten powinien zawierać główną nawigację aplikacji. Użytkownik ma do dyspozycji trzy sekcje z podsekcjami: sekcja "Browse" ("Top Anime", "Top Films", "Top Series"), sekcja "Your library" ("Now Watching", "Favorites", "Watch List"), sekcja "Other" ("Categories", "Novelties"). Każda z podsekcji powinna otworzyć się na centralnym panelu po naciśnięciu przycisku.
- **Panel górny.** Ten panel powinien zawierać elementy pomocnicze:
 1. Przycisk przejścia do panelu centralnego. Po jego naciśnięciu standardowy interfejs jest wyświetlany na panelu centralnym, tak jak przy pierwszym wejściu do głównego okna;
 2. Pole do wyszukiwania multimediów. Użytkownik może wpisać nazwę multimediów w polu, po czym zostanie mu wyświetlona lista z odpowiednimi wynikami. Każda pozycja na liście prowadzi do strony multimedialnej, która jest wyświetlana w centralnym panelu;
 3. Przycisk menu. Menu powinno wyświetlać nazwę użytkownika, a także zapewniać opcję wyjścia z głównego okna do okna logowania/rejestracji.
- **Strona multimedialna.** Ta strona powinna zawierać podstawowe informacje o wybranym anime, filmie lub serialu. Między innymi, strona powinna zawierać przyciski do dodawania/usuwania multimediów z biblioteki użytkownika. Ponadto strona powinna zawierać odtwarzacz multimedialny do przeglądania multimediów.
- **Odtwarzacz multimedialny.** Odtwarzacz powinien zapewniać przyjazną dla użytkownika i prostą funkcjonalność. Należy wdrożyć następujące funkcje:
 1. Odtwarzanie/pauza wideo;
 2. Regulacja głośności wideo;
 3. Wyświetlanie czasu trwania wideo oraz czasu, który już upłynął;
 4. Możliwość przewijania wideo do tyłu lub do przodu;
 5. Regulacja prędkości odtwarzania;
 6. Możliwość wyświetlania wideo w trybie pełnoekranowym, a także wyjścia z niego;
 7. W zależności od rodzaju multimediów, udostępnia użytkownikowi menu z możliwością wyboru określonej serii.

- **Sekcja "Browse".** Ta sekcja powinna zapewniać użytkownikowi możliwość przeglądania wszystkich dostępnych multimedów. Po kliknięciu przycisku każdej podsekcji, wszystkie dostępne banery powinny zostać wyświetlane na panelu centralnym.
- **Sekcja "Your library".** Ta sekcja powinna umożliwiać użytkownikowi przeglądanie wszystkich multimedów, które dodał do odpowiedniej podsekcji. Każda z podsekcji powinna być wyświetlana na panelu środkowym i zawierać banery oraz trzy przyciski umożliwiające przełączanie między typami multimedów.
- **Sekcja "Other".** Ta sekcja powinna być tymczasowo niedostępna i dodana w przyszłych wersjach aplikacji. Po naciśnięciu odpowiednich przycisków podsekcji na ekranie powinny pojawiać się komunikaty ostrzegające użytkownika o niedostępności.
- **Baza danych.** Baza danych powinna być bezpośrednio połączona z aplikacją i przechowywać informacje o multimedach, użytkownikach. Powinna zawierać oddzielne tabele dla każdego typu multimedów, oddzielne tabele do przechowywania informacji o seriach odpowiednich multimedów, oddzielną tabelę dla użytkowników, oddzielną tabelę do przechowywania informacji o bibliotece użytkownika.

1.2.2 Opis wymagań niefunkcjonalnych

- **Bezpieczeństwo.** Aby zapobiec niepożądanemu dostępowi i zapewnić bezpieczeństwo użytkownika, przed wejściem do okna głównego on musi wprowadzić nazwę użytkownika i hasło. Jeśli wprowadzone dane nie są zgodne z rzeczywistymi danymi przechowywanymi w bazie danych, dostęp do okna głównego nie będzie możliwy. Podczas tworzenia konta dane wprowadzone przez użytkownika muszą zostać poddane ścisłej walidacji. Ponadto system musi być odporny na iniekcje SQL.
- **Efektywność i wydajność.** Aplikacja powinna działać wydajnie i zapewniać użytkownikowi wszystkie funkcje bez opóźnień i przerw.
- **Użyteczność interfejsu.** Interfejs aplikacji powinien być intuicyjny i łatwy w użyciu. Stylizacja powinna być zgodna z nowoczesnymi standardami.
- **Niezawodność.** Aplikacja musi być stabilna i niezawodna, całkowicie eliminując możliwość wystąpienia awarii i błędów.
- **Kompatybilność.** Oprogramowanie musi być kompatybilne z bazą danych SQLite i zapewniać poprawną interakcję z nią.
- **Skalowalność pionowa.** Jeżeli liczba działań użytkownika wzrasta, system musi utrzymywać stałe działanie.
- **Skalowalność pozioma.** Jeżeli funkcjonalność aplikacji musi zostać rozszerzona poprzez dodanie nowych typów multimedów lub funkcji, oprogramowanie musi być elastyczne i łatwe do rozszerzenia. Nowe klasy i funkcje powinny być zintegrowane z istniejącym kodem bez naruszania jego integralności.
- **Aktualizacje i udoskonalenia.** System aplikacji musi być w stanie szybko wdrażać aktualizacje i ulepszenia bez większych problemów. Obejmuje to łatwe wprowadzanie zmian w kodzie źródłowym i zapewnienie kompatybilności z poprzednimi wersjami programu.
- **Elastyczność zmian interfejsu.** Interfejs aplikacji powinien być elastyczny, umożliwiając łatwe wprowadzanie zmian w strukturze i projekcie bez narażania funkcjonalności systemu. Jest to ważne w przypadku dodawania nowych funkcji lub ulepszania wyglądu aplikacji.

Rozdział 2

Opis struktury projektu

2.1 Wykorzystywany język i narzędzia

Do implementacji programu wykorzystano JavaFX opartą na pakiecie SDK corretto-21 w wersji Java 21.0.2. JavaFX to platforma aplikacji klienckich nowej generacji o otwartym kodzie źródłowym, przeznaczona dla komputerów stacjonarnych, urządzeń mobilnych i systemów wbudowanych, oparta na języku Java.

Do pisania wysokiej jakości oprogramowania i jego niezawodnej obsługi wykorzystywane będzie środowisko programistyczne IntelliJ IDEA 2024.1 (Ultimate Edition).

Do wygodnego i szybkiego tworzenia interfejsu graficznego wykorzystany zostanie SceneBuilder - to darmowe i otwarte narzędzie do tworzenia aplikacji JavaFX.

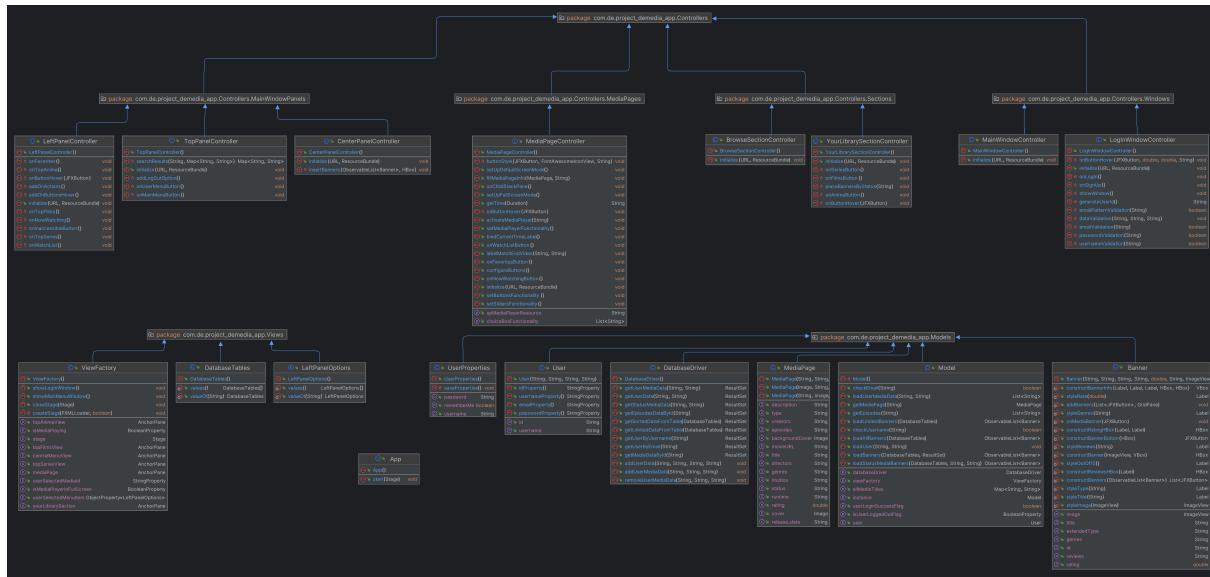
Do przechowywania danych i zarządzania nimi wykorzystywana będzie SQLite - biblioteka w języku C, która implementuje mały, szybki, samodzielny, niezawodny i w pełni funkcjonalny mechanizm bazy danych SQL.

W celu spełnienia wymagań funkcjonalnych programu, nowoczesnego interfejsu, szybkiej i niezawodnej pracy z danymi, wykorzystane zostaną biblioteki i frameworki, takie jak:

- **SQLite JDBC Driver** w wersji 3.44.1.0 – jest biblioteką umożliwiającą dostęp i tworzenie plików bazy danych SQLite w Java za pośrednictwem interfejsu API JDBC;
- **FontAwesomeFX** w wersji 4.7.0-9.1.2 – to biblioteka JavaFX udostępniająca ikony i czcionki Font Awesome;
- **JFoenix** w wersji 9.0.10 – jest biblioteką Java o otwartym kodzie źródłowym, która implementuje Google Material Design przy użyciu komponentów Java;
- **Apache Log4j** w wersji 2.20.0 – to wszechstronny, przemysłowy framework logowania Java składający się z interfejsu API, jego implementacji i komponentów wspomagających wdrażanie w różnych przypadkach użycia;
- **javafx-media** w wersji 17.0.2 – zapewnia zestaw klas do integracji audio i wideo z aplikacjami JavaFX. Głównym zastosowaniem tego pakietu jest odtwarzanie multimedialnych;
- **jetbrains.annotations** w wersji RELEASE – zestaw adnotacji Java, które mogą być używane w językach opartych na JVM. Służą one jako dodatkowa dokumentacja i mogą być interpretowane przez IDE i narzędzia do analizy statycznej w celu usprawnienia analizy kodu.

2.2 Hierarchia klas i opis metod

Rysunek 2.1 przedstawia diagram opisujący strukturę programu, klasy, ich zawartość i hierarchię.

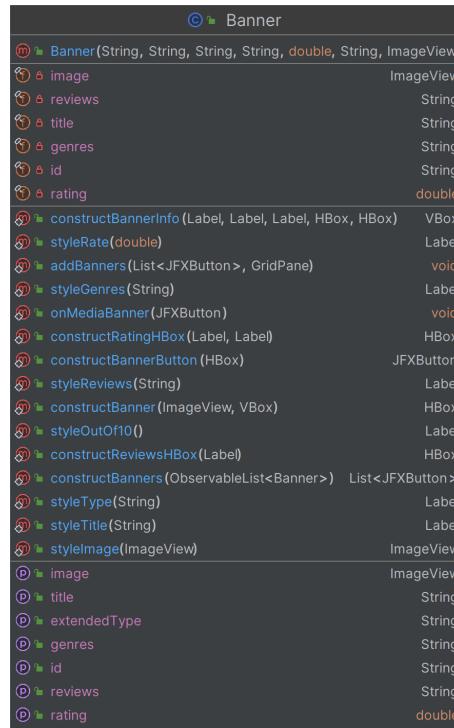


Rysunek 2.1: Diagram klas programu.

W sumie struktura aplikacji składa się z 16 klas i 2 wyliczeń.

2.2.1 Klasa Banner

Jest to klasa reprezentująca obiekt banerowy z informacjami o mediach, takimi jak tytuł mediów, id, typ, gatunek, ocena, liczba ocen, okładka.



Rysunek 2.2: Struktura klasy Banner.

Informacje o media są przekazywane do tej klasy za pośrednictwem klasy Model. Następnie, przy użyciu różnych metod, otrzymane informacje mogą być przekazywane do obiektów JavaFX i tworzyć/stylizować banery, które będą następnie wyświetlane w aplikacji.

W sumie klasa zawiera 23 metody (w tym konstruktor, gettery i settery):

- **public Banner(String, String, String, String, double, String, ImageView).** Ta metoda inicjuje obiekt Banner z podanymi parametrami, takimi jak id, title, type, genres, rating, reviews i image. Umożliwia ona tworzenie obiektów Banner o określonej charakterystyce do wykorzystania w dalszych krokach.
- **public static @NotNull ImageView styleImage(@NotNull ImageView).** Ta metoda stylizuje obraz banera, dodając do niego zaokrąglone kąty i cień. Dostosowuje rozmiar obrazu i dodaje klip, aby stworzyć atrakcyjny wizualnie wygląd.
- **public static @NotNull Label styleTitle(@NotNull String).** Stylizuje tytuł banera, ustawiając czcionkę, rozmiar i kolor tekstu. Ta metoda obsługuje również podziały wierszy w celu poprawnego wyświetlanego.
- **public static @NotNull Label styleType(String).** Stylizuje typ banera poprzez ustawienie wyrównania i koloru tekstu.
- **public static @NotNull Label styleGenres(String).** Stylizuje gatunki banera, ustawiając wyrównanie, rozmiary i wcięcia. Ta metoda pomaga wyróżnić informacje o gatunku banera.
- **public static @NotNull Label styleRate(double).** Stylizuje ocenę banera poprzez ustawienie czcionki, rozmiaru i koloru tekstu.
- **public static @NotNull Label styleOutOf10().** Stylizuje napis "/10" poprzez ustawienie koloru tekstu i wcięcia. Ta metoda uzupełnia wyświetlanie oceny.
- **public static @NotNull Label styleReviews(String).** Stylizuje tekst recenzji poprzez ustawienie czcionki, rozmiaru i koloru tekstu.
- **public static @NotNull HBox constructRatingHBox(Label, Label).** Tworzy poziomy kontener (HBox) do wyświetlania oceny i napisu "/10". Ta metoda dodaje ikonę gwiazdki i dostosowuje wyrównanie i rozmiar kontenera.
- **public static @NotNull HBox constructReviewsHBox(Label).** Tworzy poziomy kontener (HBox) do wyświetlania liczby ocen, biorąc pod uwagę długość tekstu. Ustawia wcięcie i wymiary kontenera, aby wyświetlał się poprawnie.
- **public static @NotNull VBox constructBannerInfo(Label, Label, Label, HBox, HBox).** Tworzy pionowy kontener (VBox) do przechowywania informacji o banerze, w tym nazwy, typu, gatunków, oceny i liczby ocen. Ta metoda konfiguruje wyrównanie i wcięcie kontenera.
- **public static @NotNull HBox constructBanner(ImageView, VBox).** Tworzy poziomy kontener (HBox) do przechowywania obrazu banera i informacji o banerze. Ta metoda konfiguruje rozmiar, wyrównanie i styl kontenera.
- **public static @NotNull JFXButton constructBannerButton(HBox).** Tworzy przycisk (JFXButton) dla banera, dodając animację hover zoom i dostosowanie wyglądu. Ta metoda sprawia, że baner jest interaktywny i atrakcyjny wizualnie.
- **public static @NotNull List<JFXButton> constructBanners(@NotNull ObservableList<Banner>).** Tworzy listę przycisków banerów na podstawie dostarczonej listy obiektów Banner. Ta metoda stylizuje i łączy komponenty każdego banera w gotowe przyciski.
- **public static void addBanners(@NotNull List<JFXButton>, GridPane).** Dodaje przyciski banerów do danej siatki (GridPane), umieszczając je w wierszach i kolumnach. Ta metoda zapewnia uporządkowane wyświetlanie banerów.

- **public static void onMediaBanner(@NotNull JFXButton).** Konfiguruje działanie dla przycisku banera, które jest wykonywane po kliknięciu. Ta metoda ustawia przejście do strony ze szczegółowymi informacjami o zawartości multimedialnej.

2.2.2 Klasa DatabaseDriver

Ta klasa zapewnia interfejs do interakcji z bazą danych SQLite, wykonując operacje odczytu i zapisu na danych użytkownika i zawartości multimedialnej. Zawiera metody dodawania, usuwania i pobierania danych, a także wykonywania różnych zapytań do bazy danych.

DatabaseDriver		
(m) <code>DatabaseDriver()</code>		
(f) <code>connection</code>	Connection	
(m) <code>getUserMediaData(String, String)</code>	ResultSet	
(m) <code>getUserByEmail(String)</code>	ResultSet	
(m) <code>getEpisodesDataById(String)</code>	ResultSet	
(m) <code>getUserData(String, String)</code>	ResultSet	
(m) <code>getSortedDataFromTable(DatabaseTables)</code>	ResultSet	
(m) <code>getStatusMediaData(String, String)</code>	ResultSet	
(m) <code>removeUserMediaData(String, String, String)</code>	void	
(m) <code>getLimitedDataFromTable(DatabaseTables)</code>	ResultSet	
(m) <code>getUserByUsername(String)</code>	ResultSet	
(m) <code>getMediaDataById(String)</code>	ResultSet	
(m) <code>addUserData(String, String, String, String)</code>	void	
(m) <code>addUserMediaData(String, String, String)</code>	void	

Rysunek 2.3: Struktura klasy DatabaseDriver.

W sumie klasa zawiera 13 metod (w tym konstruktor):

- **public DatabaseDriver().** Inicjalizuje obiekt DatabaseDriver i nawiązuje połączenie z bazą danych SQLite przy użyciu pliku demedia.db. W przypadku błędu połączenia, wyjątek SQLException jest ignorowany.
- **public void addUserData(String, String, String, String).** Dodaje nowy rekord do tabeli Users z określonym identyfikatorem użytkownika, nazwą użytkownika, adresem e-mail i hasłem. Wyjątki SQL zgłasiane przez zapytanie są ignorowane.
- **public void addUserMediaData(String, String, String).** Dodaje nowy rekord do tabeli User_Media, kojarząc użytkownika z zawartością multimedialną i statusem. Jeśli wystąpi wyjątek SQL, zostanie on zignorowany.
- **public ResultSet getUserMediaData(String, String).** Zwraca wynik wykonania zapytania, które wybiera rekord z tabeli User_Media na podstawie identyfikatorów użytkownika i treści multimedialnej. Jeśli wystąpi błąd SQL, zwracana jest wartość null.
- **public void removeUserMediaData(String, String, String).** Usuwa rekord z tabeli User_Media na podstawie określonego user_id, media_id i status. Wyjątki SQL zgłasiane przez zapytanie są ignorowane.

- **public ResultSet getUserData(String, String).** Wykonuje zapytanie w celu pobrania danych użytkownika według nazwy użytkownika i hasła. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getUserByUsername(String).** Wykonuje zapytanie w celu pobrania danych użytkownika według nazwy użytkownika. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getUserByEmail(String).** Wykonuje zapytanie w celu pobrania danych użytkownika za pomocą e-maila. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getSortedDataFromTable(DatabaseTables).** Wykonuje zapytanie w celu pobrania wszystkich rekordów z określonej tabeli, posortowanych malejąco według oceny. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getLimitedDataFromTable(@NotNull DatabaseTables).** Wykonuje zapytanie w celu pobrania określonych rekordów z określonej tabeli na podstawie predefiniowanej listy identyfikatorów. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getStatusMediaData(String, String).** Wykonuje zapytanie w celu pobrania rekordów z tabel zawartości multimedialnych (AnimeList, FilmsList, SeriesList) pasujących do podanego identyfikatora media i statusu. Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getMediaDataById(@NotNull String).** Wykonuje zapytanie w celu pobrania danych zawartości multimedialnej według identyfikatora multimedialnych z odpowiedniej tabeli (AnimeList, FilmsList, SeriesList). Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.
- **public ResultSet getEpisodesDataById(@NotNull String).** Wykonuje zapytanie w celu uzyskania danych epizodów anime lub serialu według identyfikatora media z odpowiedniej tabeli (AnimeEpisodes, SeriesEpisodes). Zwraca wynik zapytania lub wartość null w przypadku błędu SQL.

2.2.3 Klasa User

Klasa User to model użytkownika wykorzystujący właściwości StringProperty, który pozwala łatwo powiązać dane użytkownika z elementami interfejsu użytkownika i automatycznie aktualizować wartości w interfejsie użytkownika, gdy dane ulegną zmianie. Zawiera cztery właściwości: identyfikator użytkownika, nazwę użytkownika, adres e-mail i hasło oraz zapewnia metody dostępu i modyfikacji tych właściwości.

W sumie klasa zawiera 7 metod (w tym konstruktor, gettery i settery):

- **User(String, String, String, String).** Inicjuje obiekt User z określonymi wartościami dla identyfikatora użytkownika, nazwy użytkownika, adresu e-mail i hasła. Każda wartość jest przechowywana w odpowiedniej właściwości.

Gettery i settery są pomijane.

© User		
(m) User(String, String, String, String)		
© f	password	StringProperty
© f	id	StringProperty
© f	email	StringProperty
© f	username	StringProperty
(m)	idProperty()	StringProperty
(m)	usernameProperty()	StringProperty
(m)	emailProperty()	StringProperty
(m)	passwordProperty()	StringProperty
(m)	getId()	String
(m)	getUsername()	String

Rysunek 2.4: Struktura klasy User.

2.2.4 Klasa UserProperties

Klasa UserProperties jest przeznaczona do zarządzania ustawieniami użytkownika, takimi jak nazwa użytkownika, hasło i stan pola wyboru "Remember Me", przechowywanymi w pliku konfiguracyjnym user.properties. Klasa zapewnia wygodny interfejs do odczytu i zapisu tych właściwości, automatycznie zapisując zmiany w pliku konfiguracyjnym.

© UserProperties		
(m) UserProperties()		
© f	properties	Properties
© f	CONFIG_FILE	String
(m)	setUsername(String)	void
(m)	getUsername()	String
(m)	saveProperties()	void
(m)	setRememberMe(boolean)	void
(m)	getPassword()	String
(m)	setPassword(String)	void
(m)	isRememberMe()	boolean

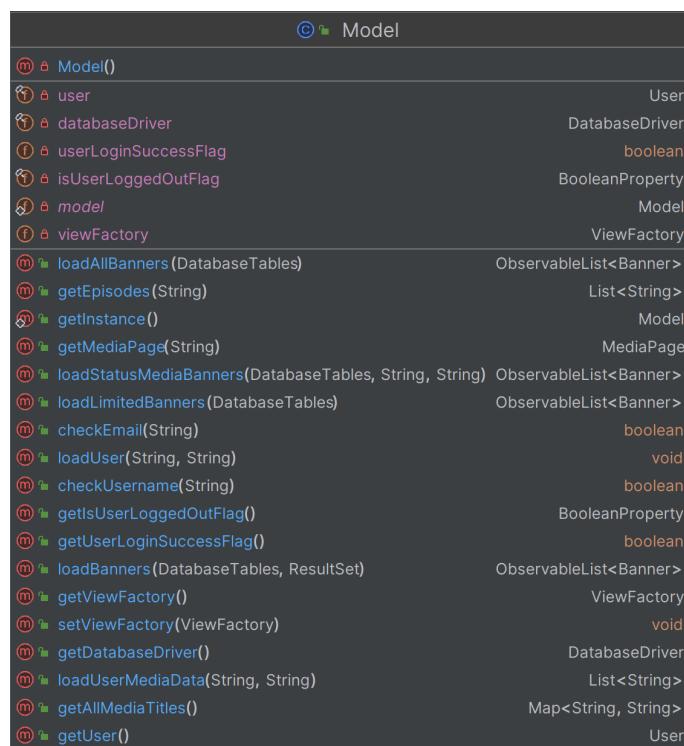
Rysunek 2.5: Struktura klasy UserProperties.

W sumie klasa zawiera 8 metod (w tym konstruktor, gettery i settery):

- **public UserProperties().** Konstruktor klasy, który inicjuje obiekt Properties i ładuje właściwości z pliku konfiguracyjnego. Jeśli plik nie może zostać odczytany, wyjątek IOException jest ignorowany.
- **private void saveProperties().** Zapisuje wszystkie właściwości do pliku konfiguracyjnego. Jeśli wystąpi błąd IO, wyjątek IOException zostanie zignorowany.
- **public void setRememberMe(boolean).** Ustawia wartość właściwości rememberMe. Zapisuje zmienioną właściwość do pliku konfiguracyjnego.
- **public boolean isRememberMe().** Zwraca wartość właściwości rememberMe. Jeśli właściwość nie jest ustawiona, zwracana jest wartość domyślna false.
- **public void setUsername(String).** Ustawia wartość właściwości username. Zapisuje zmienioną właściwość do pliku konfiguracyjnego.
- **public String getUsername().** Zwraca wartość właściwości username. Jeśli właściwość nie jest ustawiona, zwracany jest pusty ciąg znaków.
- **public void setPassword(String).** Ustawia wartość właściwości password. Zapisuje zmienioną właściwość w pliku konfiguracyjnym.
- **public String getPassword().** Zwraca wartość właściwości password. Jeśli właściwość nie jest ustawiona, zwracany jest pusty ciąg znaków.

2.2.5 Klasa Model

Klasa Model implementuje wzorzec Singleton i jest przeznaczona do zarządzania danymi i logiką interakcji między różnymi komponentami aplikacji. Klasa ta hermetyzuje dostęp do bazy danych, zarządzanie stanem użytkownika i tworzenie widoków.



Rysunek 2.6: Struktura klasy Model.

W sumie klasa zawiera 19 metod (w tym konstruktor, gettery i settery):

- **private Model().** Prywatny konstruktor inicjalizuje obiekty viewFactory, databaseDriver, user i isUserLoggedOutFlag. Zapobiega to tworzeniu instancji klasy z zewnętrznego kodu.
- **public static synchronized Model getInstance().** Zwraca jedyną instancję klasy Model. Jeśli instancja nie została jeszcze utworzona, tworzy ją.
- **public ViewFactory getViewFactory().** Zwraca bieżący obiekt ViewFactory.
- **public void setViewFactory(ViewFactory).** Ustawia nowy obiekt ViewFactory.
- **public DatabaseDriver getDatabaseDriver().** Zwraca bieżący obiekt DatabaseDriver.
- **public User getUser().** Zwraca bieżący obiekt użytkownika.
- **public boolean getUserLoginSuccessFlag().** Zwraca wartość flagi udanego logowania użytkownika.
- **public BooleanProperty getIsUserLoggedOutFlag().** Zwraca wartość flagi wylogowania użytkownika.
- **public void loadUser(String, String).** Ładuje dane użytkownika z bazy danych według podanej nazwy użytkownika i hasła. Jeśli użytkownik zostanie znaleziony, ustawia wartości pól użytkownika i aktualizuje flagę pomyślnego logowania.
- **public boolean checkUsername(String).** Sprawdza, czy istnieje użytkownik o podanej nazwie użytkownika. Zwraca wartość true, jeśli nazwa użytkownika nie istnieje, w przeciwnym razie false.
- **public boolean checkEmail(String).** Sprawdza, czy istnieje użytkownik o podanym adresie e-mail. Zwraca wartość true, jeśli adres e-mail nie istnieje, w przeciwnym razie false.
- **public List<String> loadUserMediaData(String, String).** Ładuje dane multimedialne użytkownika z bazy danych według określonego user_id i media_id. Zwraca listę stanów mediów.
- **public ObservableList<Banner> loadLimitedBanners(DatabaseTables).** Ładuje ograniczony zestaw banerów dla określonej tabeli. Zwraca listę banerów.
- **public ObservableList<Banner> loadAllBanners(DatabaseTables).** Ładuje wszystkie banery dla określonej tabeli. Zwraca listę banerów.
- **public ObservableList<Banner> loadStatusMediaBanners(DatabaseTables, String, String).** Ładuje banery multimedialne o określonym statusie. Zwraca listę banerów.
- **public ObservableList<Banner> loadBanners(DatabaseTables, ResultSet).** Konwertuje dane z ResultSet na listę banerów. Zwraca listę banerów.
- **public Map<String, String> getAllMediaTitles().** Ładuje wszystkie tytuły mediów z tabel bazy danych. Zwraca mapę, w której kluczami są tytuły mediów, a wartościami ich identyfikatory.
- **public MediaPage getMediaPage(String).** Ładuje stronę multimedialną według określonego media_id. Zwraca obiekt MediaPage z danymi mediów.
- **public List<String> getEpisodes(String).** Ładuje adresy URL epizodów dla określonego media_id. Zwraca listę adresów URL epizodów.

2.2.6 Klasa MediaPage

Klasa MediaPage to model danych dla strony z zawartością multimedialną. Służy do przechowywania informacji o różnych mediach, takich jak anime, filmy i seriale telewizyjne. Klasa zawiera pola do przechowywania informacji o tytule, statusie, dacie premiery, typie, epizodach, czasie odtwarzania, gatunkach, studiach, reżyserach, twórcach, ocenie, opisie, adresie URL filmu, okładce i obrazie tła.

MediaPage	
⑩	MediaPage(Image, String, String, String, String, String, double, String, String, Image)
⑩	MediaPage(String, Image, String, String, String, String, String, double, String, Image)
⑩	MediaPage(String, String, String, String, String, String, String, double, String, Image)
①	episodes String
①	release_date String
①	rating double
①	movie_URL String
①	description String
①	type String
①	background_cover Image
①	status String
①	genres String
①	cover Image
①	runtime String
①	directors String
①	title String
①	studios String
①	creators String
⑩	getCreators() String
⑩	getMovieURL() String
⑩	getDirectors() String
⑩	getGenres() String
⑩	getStudios() String
⑩	getRating() double
⑩	getTitle() String
⑩	getStatus() String
⑩	getEpisodes() String
⑩	getDescription() String
⑩	getRelease_date() String
⑩	getCover() Image
⑩	getType() String
⑩	getBackgroundCover() Image
⑩	getRuntime() String

Rysunek 2.7: Struktura klasy MediaPage.

W sumie klasa zawiera 18 metod (w tym konstruktory, gettery i settery):

- **public MediaPage(String, String, String, String, String, String, double, String, Image, Image).** Konstruktor inicjujący stronę multimedialną z informacjami o anime.
- **public MediaPage(Image, String, String, String, String, String, String, double, String, String, Image).** Konstruktor inicjujący stronę multimedialną z informacjami o filmach.
- **public MediaPage(String, Image, String, String, String, String, String, String, double, String, Image).** Konstruktor inicjujący stronę multimedialną z informacjami o serialach telewizyjnych.
- **public String getTitle().** Zwraca tytuł multimediiów.
- **public String getStatus().** Zwraca status multimediiów.
- **public String getRelease_date().** Zwraca datę wydania mediów.
- **public String getType().** Zwraca typ mediów.
- **public String getEpisodes().** Zwraca liczbę epizodów multimediiów.

- **public String getRuntime()**. Zwraca czas trwania multimedów.
- **public String getGenres()**. Zwraca gatunki mediów.
- **public String getStudios()**. Zwraca studia zaangażowane w produkcję mediów.
- **public String getDirectors()**. Zwraca reżyserów mediów.
- **public String getCreators()**. Zwraca twórców mediów.
- **public double getRating()**. Zwraca ocenę mediów.
- **public String getDescription()**. Zwraca opis multimedów.
- **public String getMovieURL()**. Zwraca adres URL filmu.
- **public Image getCover()**. Zwraca okładkę multimedów.
- **public Image getBackgroundCover()**. Zwraca obraz tła mediów.

2.2.7 Klasa ViewFactory

Ta klasa jest odpowiedzialna za zarządzanie widokami (Views) i oknami (Stages) w aplikacji. Ładuje różne części interfejsu użytkownika z plików FXML, udostępnia metody kontrolujące stan odtwarzacza multimedialnego i aktualnie wybranej pozycji menu, a także otwiera i zamyka okna.

ViewFactory	
(m) ViewFactory()	
(f) topAnimeView	AnchorPane
(f) topFilmsView	AnchorPane
(t) isMediaPlaying	BooleanProperty
(f) centralMenuView	AnchorPane
(t) isMediaPlayerInFullScreen	BooleanProperty
(f) topSeriesView	AnchorPane
(f) stage	Stage
(t) userSelectedMenuItem	ObjectProperty<LeftPanelOptions>
(f) mediaPage	AnchorPane
(t) userSelectedMediaId	StringProperty
(m) getTopFilmsView()	AnchorPane
(m) getStage()	Stage
(m) setStage(Stage)	void
(m) getUserSelectedMediaId()	StringProperty
(m) getMediaPage()	AnchorPane
(m) getTopAnimeView()	AnchorPane
(m) getYourLibrarySection()	AnchorPane
(m) showMainMenuWindow()	void
(m) showLoginWindow()	void
(m) createStage(FXMLLoader, boolean)	void
(m) getIsMediaPlayerInFullScreen()	BooleanProperty
(m) getIsMediaPlaying()	BooleanProperty
(m) getTopSeriesView()	AnchorPane
(m) getUserSelectedMenuItem()	ObjectProperty<LeftPanelOptions>
(m) getCentralMenuView()	AnchorPane
(m) closeStage(Stage)	void

Rysunek 2.8: Struktura klasy ViewFactory.

W sumie klasa zawiera 17 metod (w tym konstruktor, gettery i settery):

- **public ViewFactory()**. Inicjalizuje właściwości userSelectedItem, userSelectedMediaId, isMediaPlaying i isMediaPlayerInFullScreen oraz tworzy nowe okno (Stage). public ObjectProperty<LeftPanelOptions> getUserSelectedItem(). Zwraca aktualnie wybrany element menu.
- **public StringProperty getUserSelectedMediaId()**. Zwraca identyfikator aktualnie wybranego media.
- **public BooleanProperty getIsMediaPlaying()**. Zwraca flagę wskazującą, czy multimedialne jest odtwarzane.
- **public BooleanProperty getIsMediaPlayerInFullScreen()**. Zwraca flagę wskazującą, czy odtwarzacz multimedialny jest w trybie pełnoekranowym.
- **public void setStage(Stage)**. Ustawia bieżącą scenę aplikacji.
- **public Stage getStage()**. Zwraca bieżącą scenę aplikacji.
- **public AnchorPane getCentralMenuView()**. Zwraca menu centralne, ładowając je z pliku FXML, jeśli nie zostało jeszcze załadowane.
- **public AnchorPane getTopAnimeView()**. Zwraca widok dla najlepszych anime, ładowając go z pliku FXML, jeśli nie został jeszcze załadowany.
- **public AnchorPane getTopFilmsView()**. Zwraca widok dla najlepszych filmów, ładowając go z pliku FXML, jeśli nie został jeszcze załadowany.
- **public AnchorPane getTopSeriesView()**. Zwraca widok dla najlepszych seriali, ładowając go z pliku FXML, jeśli nie został jeszcze załadowany.
- **public AnchorPane getMediaPage()**. Zwraca stronę zawartości multimedialnej, ładowając ją z pliku FXML.
- **public AnchorPane getYourLibrarySection()**. Zwraca widok dla sekcji "Your library", ładowając go z pliku FXML.
- **public void showLogInWindow()**. Otwiera okno logowania, ładowając je z pliku FXML.
- **public void showMainWindow()**. Otwiera menu główne aplikacji, ładowając je z pliku FXML i instalując kontroler.
- **private void createStage(@NotNull FXMLLoader loader, boolean)**. Tworzy nowe okno i ustawia scenę z widokiem załadowanym z pliku FXML.
- **public void closeStage(@NotNull Stage)**. Zamknie określone okno.

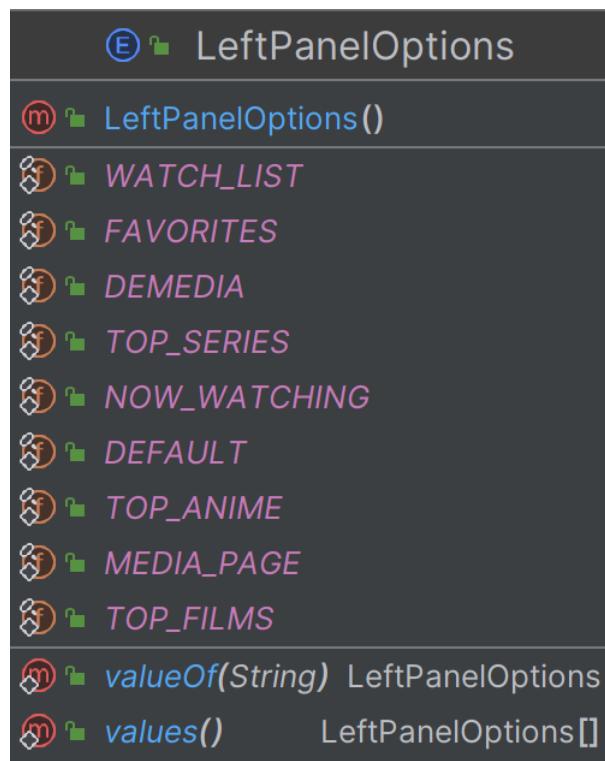
2.2.8 Wyliczenie LeftPanelOptions

Reprezentuje zestaw stałych, które są używane do oznaczania różnych sekcji lewego panelu interfejsu użytkownika aplikacji. Każda stała odpowiada określonej sekcji, do której użytkownik może nawigować.

W sumie wyliczenie zawiera 9 stałych:

- **DEMEDIA**. Główna sekcja aplikacji, która otwiera się po uruchomieniu aplikacji.
- **TOP_ANIME**. Sekcja z najlepszymi anime.

- **TOP_FILMS.** Sekcja z najlepszymi filmami.
- **TOP_SERIES.** Sekcja z najlepszymi serialami.
- **NOW_WATCHING.** Sekcja z multimediami, które użytkownik aktualnie ogląda.
- **FAVORITES.** Sekcja z ulubionymi mediami użytkownika.
- **WATCH_LIST.** Sekcja z listą mediów, które użytkownik planuje obejrzeć.
- **MEDIA_PAGE.** Strona z zawartością multimedialną.
- **DEFAULT.** Służy do resetowania wartości.



Rysunek 2.9: Struktura wyliczenia LeftPanelOptions.

2.2.9 Wyliczenie DatabaseTables

Reprezentuje zestaw stałych, które oznaczają różne tabele bazy danych używane w aplikacji. Każda stała odpowiada konkretnej tabeli, co pomaga uprościć i uczynić bardziej czytelnym kod związany z dostępem do danych.

W sumie wyliczenie zawiera 5 stałych:

- **Users.** Tabela zawierająca informacje o użytkownikach.
- **AnimeList.** Tabela zawierająca listę anime.
- **FilmsList.** Tabela zawierająca listę filmów.
- **SeriesList.** Tabela zawierająca listę seriali telewizyjnych.
- **User_Media.** Tabela zawierająca informacje o mediach powiązanych z użytkownikami.

④ DatabaseTables	
⑥ DatabaseTables()	
⑦ AnimeList	
⑦ User_Media	
⑦ Users	
⑦ FilmsList	
⑦ SeriesList	
⑧ values() DatabaseTables[]	
⑧ valueOf(String) DatabaseTables	

Rysunek 2.10: Struktura wyliczenia DatabaseTables.

2.2.10 Klasa CenterPanelController

Odpowiada za kontrolowanie centralnego panelu głównego okna aplikacji, który wyświetla banery anime, filmów i seriali telewizyjnych po uruchomieniu aplikacji. Implementuje interfejs Initializable, który umożliwia inicjalizację danych podczas tworzenia kontrolera.

© CenterPanelController		
⑥ CenterPanelController()		
⑦ anime_layout	HBox	
⑦ films_layout	HBox	
⑦ series_layout	HBox	
⑧ insertBanners(ObservableList<Banner>, HBox) void		
⑧ initialize(URL, ResourceBundle) void		

Rysunek 2.11: Struktura klasy CenterPanelController.

W sumie klasa zawiera 2 metody:

- **public void initialize(URL, ResourceBundle).** Ta metoda jest wywoływana podczas inicjalizacji kontrolera. Ładuje ona ograniczoną liczbę banerów dla anime, filmów i seriali telewizyjnych, używając klasy Model, aby uzyskać dostęp do wymaganych metod i dodaje te banery do odpowiednich kontenerów (HBox).
- **private void insertBanners(@NotNull ObservableList<Banner>, HBox).** Metoda pobiera listę banerów i kontener HBox, do którego dodawane są banery. Konstruuje przyciski dla każdego banera przy użyciu metody klasy Banner i dodaje je do kontenera.

2.2.11 Klasa LeftPanelController

Odpowiada za kontrolowanie lewego panelu głównego okna aplikacji, który zawiera przyciski do nawigacji po różnych sekcjach aplikacji. Implementuje interfejs Initializable, który umożliwia inicjalizację danych i ustawień podczas tworzenia kontrolera.

© LeftPanelController		
(m)	LeftPanelController()	
(f)	top_series_button	JFXButton
(f)	categories_button	JFXButton
(f)	top_films_button	JFXButton
(f)	watch_list_button	JFXButton
(f)	favorites_button	JFXButton
(f)	now_watching_button	JFXButton
(f)	novelties_button	JFXButton
(f)	top_anime_button	JFXButton
(m)	onTopFilms()	void
(m)	onInaccessibleButton()	void
(m)	onButtonHover(JFXButton)	void
(m)	onFavorites()	void
(m)	initialize(URL, ResourceBundle)	void
(m)	onWatchList()	void
(m)	addOnButtonsHover()	void
(m)	onNowWatching()	void
(m)	addOnActions()	void
(m)	onTopAnime()	void
(m)	onTopSeries()	void

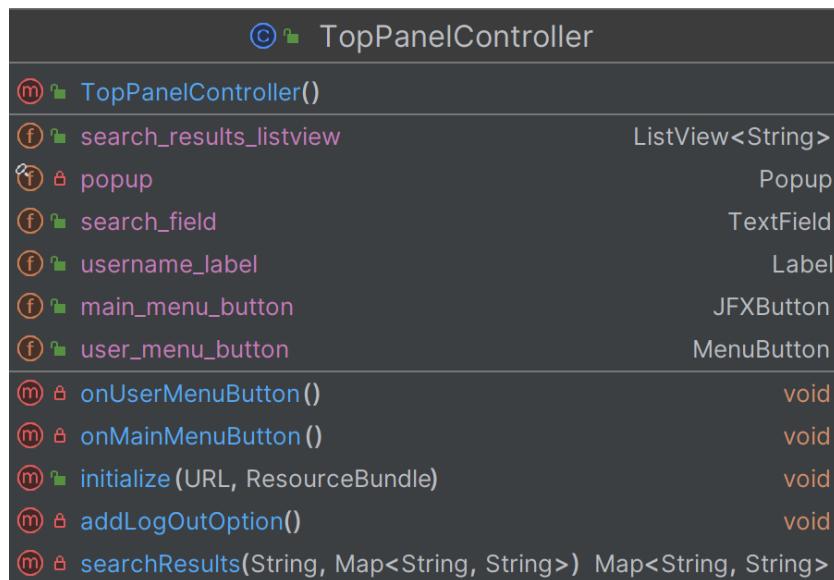
Rysunek 2.12: Struktura klasy LeftPanelController.

W sumie klasa zawiera 11 metod:

- **public void initialize(URL, ResourceBundle).** Jest wywoływana podczas inicjalizacji kontrolera. Dodaje ona akcje i efekty do przycisków oraz ustawia obserwatora dla wybranej opcji menu.
- **private void addOnActions().** Dodaje akcje dla przycisków. Każdy przycisk otrzymuje odpowiednią akcję, która aktualizuje wybraną opcję menu i kontroluje stan przycisków.
- **private void addOnButtonsHover().** Dodaje efekt najechania kursorem do przycisków. Efekt ten zwiększa rozmiar przycisku po najechaniu na niego kursem.
- **private void onButtonHover(@NotNull JFXButton).** Dodaje animację zwiększania i zmniejszania rozmiaru przycisku po najechaniu i oddaleniu kursem.
- **private void onTopAnime().** Aktualizuje wybraną opcję menu do TOP_ANIME i kontroluje stan przycisków.
Podobne metody **onTopFilms**, **onTopSeries**, **onNowWatching**, **onFavorites**, **onWatchList** wykonują podobne akcje dla innych przycisków.
- **private void onInaccessibleButton().** Dodaje obsługę niedostępnych przycisków (categories_button i novelties_button). Po kliknięciu tych przycisków wyświetlana jest wskazówka z komunikatem o niedostępności sekcji.

2.2.12 Klasa TopPanelController

Jest odpowiedzialna za zarządzanie górnym panelem głównego okna aplikacji, który zawiera kontrolki takie jak przyciski menu, pole wyszukiwania, menu użytkownika i inne elementy. Implementuje interfejs Initializable, który umożliwia inicjalizację danych i ustawień podczas tworzenia kontrolera.



Rysunek 2.13: Struktura klasy TopPanelController.

W sumie klasa zawiera 5 metod:

- **public void initialize(URL, ResourceBundle).** Wywoływana podczas inicjalizacji kontrolera. Wykonuje ona początkową konfigurację danych, taką jak ładowanie wszystkich tytułów multi-mediów, ustawianie nazwy użytkownika i konfigurowanie wyskakującego okienka dla wyników wyszukiwania.
- **private void searchResults(String, @NotNull Map<String, String>).** Przeszukuje wszystkie tytuły mediów i zwraca wyniki zawierające wprowadzony tekst.
- **private void onUserMenuButton().** Dodaje efekt najechania kursorem do przycisku menu użytkownika, zmieniając jego styl po najechaniu kursorem i jego oddaleniu.
- **private void addLogOutOption().** Dodaje opcję "Log out" do menu użytkownika z odpowiednią ikoną i stylem.
- **private void onMainMenuButton().** Dodaje akcję dla przycisku menu głównego, który ustawia wartość LeftPanelOptions.DEMEDIA na element wybrany przez użytkownika.

2.2.13 Klasa BrowseSectionController

Odpowiada za sterowanie sekcją "Browse" w aplikacji. W zależności od wybranego elementu lewego panelu otwiera podsekcję ze wszystkimi banerami określonego typu mediów. Implementuje interfejs Initializable, który umożliwia inicjalizację danych i ustawień podczas tworzenia kontrolera.

W sumie klasa zawiera jedną metodę:

- **public void initialize(URL, ResourceBundle).** Wywoywana podczas inicjalizacji kontrolera. Wykonuje ona ładowanie banerów w zależności od wybranej opcji w lewym panelu.

©	BrowseSectionController
(m)	BrowseSectionController()
(f)	banners_layout GridPane
(m)	initialize (URL, ResourceBundle) void

Rysunek 2.14: Struktura klasy BrowseSectionController.

2.2.14 Klasa YourLibrarySectionController

Jest odpowiedzialna za zarządzanie sekcją "Your Library" w aplikacji. Implementuje interfejs Initializable, który umożliwia inicjalizację danych i ustawień podczas tworzenia kontrolera.

©	YourLibrarySectionController
(m)	YourLibrarySectionController()
(f)	series_gridpane_button JFXButton
(f)	anime_gridpane GridPane
(f)	series_gridpane GridPane
(f)	films_gridpane GridPane
(f)	anime_anchorpane AnchorPane
(f)	anime_gridpane_button JFXButton
(f)	films_gridpane_button JFXButton
(f)	films_anchorpane AnchorPane
(f)	series_anchorpane AnchorPane
(m)	onFilmsButton () void
(m)	onButtonHover (JFXButton) void
(m)	onAnimeButton () void
(m)	onSeriesButton() void
(m)	placeBannersByStatus(String) void
(m)	initialize (URL, ResourceBundle) void

Rysunek 2.15: Struktura klasy YourLibrarySectionController.

W sumie klasa zawiera 6 metod:

- **public void initialize(URL, ResourceBundle).** Wywoływana podczas inicjalizacji kontrolera. W zależności od wybranej opcji menu, ładuje odpowiednie banery, ustawia akcje dla przycisków, ustawia efekty po najechaniu na przyciski i wyłącza domyślny przycisk anime.
- **private void placeBannersByStatus(String).** Ładuje banery z bazy danych na podstawie statusu ("Now watching", "Favorites", "Watch list") i dodaje je do odpowiednich siatek (GridPane).
- **private void onButtonHover(@NotNull JFXButton).** Dodaje efekty po najechaniu na przyciski, w tym zmianę koloru tła i dodanie animacji powiększenia.

- **private void onAnimeButton()**. Odpowiada za wyświetlenie odpowiednich paneli i wyłączenie bieżącego przycisku. Podobne metody **onFilmsButton** i **onSeriesButton** wykonują podobne akcje dla innych przycisków i paneli.

2.2.15 Klasa MediaPageController

Jest kontrolerem strony multimedialnej aplikacji. Klasa ta implementuje interfejs Initializable, który umożliwiainicjalizację komponentów interfejsu użytkownika podczas ładowania. Klasa jest odpowiedzialna za kontrolowanie odtwarzania treści multimedialnych, wyświetlanie informacji o mediach i obsługę interakcji użytkownika.

MediaPageController	
MediaPageController()	
episodes_separator	Separator
media_production_label	Label
favorites_icon	FontAwesomeIconView
time_slider	JFXSlider
media_page_scrollpane	ScrollPane
parent_anchorpance	AnchorPane
media_episodes_label	Label
video_mediaview	MediaView
controls_pane	Pane
video_media	Media
is_playing	boolean
background_media_image	ImageView
media_genres_label	Label
media_image	ImageView
controls_hbox	HBox
media_title_label	Label
watch_list_icon	FontAwesomeIconView
status_label	Label
parent_stackpane	StackPane
media_id	String
PPR_button	Button
volume_slider	JFXSlider
media_info_anchorpance	AnchorPane
is_muted	boolean
media_status_label	Label
media_runtime_label	Label
parent_vbox	VBox
speed_button	Button
episodes_hbox	HBox
user_id	String
total_time_label	Label
media_type_label	Label
media_description_label	Label
PPR_icon	FontAwesomeIconView
loading_arc	Arc
at_end_of_video	boolean
now_watching_button	JFXButton
media_info_vbox	VBox
now_watching_icon	FontAwesomeIconView
status	String
fullscreen_button	Button
episodes_choicebox	JFXComboBox<String>
volume_button	Button
rating_hbox	HBox
media_rating_label	Label
current_time_label	Label
production_label	Label
fullscreen_icon	FontAwesomeIconView
child_stackpane	StackPane
parent_borderpane	BorderPane
volume_icon	FontAwesomeIconView
favorites_button	JFXButton
video_mediaplayer	MediaPlayer
watch_list_button	JFXButton
volume_hbox	HBox
bindCurrentTimeLabel()	void
setMediaPlayerFunctionality()	void
setChoiceBoxFunctionality (List<String>)	void
labelMatchEndVideo(String, String)	void
getTime(Duration)	String
setUpFullScreenMode()	void
onButtonHover (JFXButton)	void
buttonStyle(JFXButton, FontAwesomeIconView, String)	void
activateMediaPlayer(String)	void
configureButtons()	void
onFavoritesButton ()	void
setSlidersFunctionality ()	void
onChildStackPane()	void
initialize (URL, ResourceBundle)	void
onWatchListButton ()	void
setUpDefaultScreenMode()	void
fillMediaPageInfo(MediaPage, String)	void
onNowWatchingButton ()	void
setUpMediaPlayerResource(String)	void
setButtonsFunctionality ()	void

Rysunek 2.16: Struktura klasy MediaPageController.

W sumie klasa zawiera 21 metod:

- **public void initialize(URL, ResourceBundle)**. Metoda ta jest wywoływana podczasinicjalizacji kontrolera. Konfiguruje odtwarzacz multimedialny, ładuje dane multimedialne i konfiguruje przyciski. Ustawia funkcję wyboru epizodu i obsługę zdarzeń dla różnych przycisków.
- **private void startLoadingAnimation()**. Uruchamia animację obrotu dla elementu graficznego loading_arc w odtwarzaczu multimedialnym.

- **private void fillMediaPageInfo(MediaPage, String).** Wypełnia informacje o mediach, takie jak okładka, tytuł, status, gatunki, epizody, producenci, ocena i opis. Konfiguruje widoczność i zawartość elementów interfejsu użytkownika w zależności od typu mediów (serial lub film).
- **private void activateMediaPlayer(String).** Aktywuje i konfiguruje odtwarzacz multimediiów do odtwarzania multimediiów pod podanym adresem URL. Konfiguruje przyciski, slider'y i funkcjonalność odtwarzacza multimediiów, a także procedury obsługi zdarzeń do sterowania odtwarzaniem i głośnością.
- **private void setUpMediaPlayerResource(String).** Tworzy nowy obiekt Media i MediaPlayer z podanym adresem URL i przypisuje go do MediaView.
- **private void setChoiceBoxFunctionality(List<String>).** Ustawia funkcjonalność listy rozwijanej do wyboru epizodów. Umożliwia przełączanie między epizodami i aktualizowanie odtwarzacza multimediiów zgodnie z wybranym epizodem.
- **private void setButtonsFunctionality().** Konfiguruje obsługę zdarzeń dla przycisków sterowania odtwarzaniem, szybkości, głośności i trybu pełnoekranowego. Zawiera logikę do przełączania ikon i tekstów przycisków w zależności od bieżącego stanu odtwarzacza multimediiów.
- **private void setSlidersFunctionality().** Konfiguruje slider'y głośności i czasu odtwarzania, kojarząc ich właściwości z odtwarzaczem multimedialnym i dodaje słuchaczy do obsługi zmian wartości slider'ów.
- **private void setMediaPlayerFunctionality().** Konfiguruje podstawowe funkcje odtwarzacza multimediiów, takie jak aktualizacja bieżącego czasu, całkowity czas odtwarzania multimediiów, obsługa zakończenia odtwarzania multimediiów i synchronizacja slider'ów z odtwarzaczem multimediiów.
- **private void setUpDefaultScreenMode().** Ustawia interfejs użytkownika na domyślny tryb ekranu, dostosowując widoczność i rozmiar różnych elementów.
- **private void setUpFullScreenMode().** Ustawia interfejs użytkownika na tryb pełnoekranowy, dostosowując widoczność i rozmiary elementów w celu optymalnego wyświetlania w trybie pełnoekranowym.
- **private void onChildStackPane().** Konfiguruje układy czasowe tak, aby ukrywały panel sterowania i kursor myszy po określonym czasie bezczynności.
- **private void bindCurrentTimeLabel().** Wiąże tekst etykiety bieżącego czasu z bieżącym czasem odtwarzacza multimedialnego, aktualizując go podczas odtwarzania.
- **private String getTime(Duration).** Konwertuje obiekt Duration na ciąg znaków w formacie czasu (godziny, minuty, sekundy).
- **private void labelMatchEndVideo(String, String).** Sprawdza, czy bieżący czas odtwarzania odpowiada całkowitemu czasowi multimediiów i aktualizuje stan przycisku odtwarzania/pauzy.
- **private void configureButtons().** Konfiguruje przyciski służące do dodawania multimediiów do różnych list użytkownika, takich jak „Teraz oglądane”, „Ulubione” i „Lista obserwowanych”, w oparciu o bieżący stan multimediiów w bazie danych użytkownika.
- **private void buttonStyle(JFXButton button, FontAwesomeIconView icon, String).** Dostosowuje styl przycisku na podstawie jego stanu (dodany do listy lub nie).
- **private void addHoverEffect(JFXButton).** Konfiguruje animację po najechaniu na przycisk, zwiększając i zmniejszając jego rozmiar.

- **private void onNowWatchingButton()**. Obsługa zdarzenia dla przycisku "Now Watching", dodaje lub usuwa multimedia z listy "Now Watching" użytkownika.
- **private void onFavouritesButton()**. Obsługa zdarzenia dla przycisku "Favorites", dodaje lub usuwa multimedia z listy "Favorites" użytkownika.
- **private void onWatchListButton()**. Obsługa zdarzenia dla przycisku "Watch List", dodaje lub usuwa multimedia z listy "Watch List" użytkownika.

2.2.16 Klasa LogInWindowController

Jest kontrolerem dla okna autoryzacji. Główne zadania klasy to autoryzacja użytkownika, rejestracja nowego użytkownika, validacja wprowadzonych danych oraz zarządzanie oknami autoryzacji i rejestracji. Implementuje interfejs Initializable, który umożliwia inicjalizację komponentów podczas ładowania okna.

© LogInWindowController	
Ⓜ️ ↳ LogInWindowController()	
Ⓕ️ ↳ apple_button	JFXButton
Ⓕ️ ↳ login_window	AnchorPane
Ⓕ️ ↳ google_button	JFXButton
Ⓕ️ ↳ username_field	TextField
Ⓕ️ ○ valid_email_flag	boolean
Ⓕ️ ↳ phone_button	JFXButton
Ⓕ️ ↳ error_label	Label
Ⓕ️ ↳ login_button	JFXButton
Ⓕ️ ↳ password_error_label	Label
Ⓕ️ ↳ facebook_button	JFXButton
Ⓕ️ ↳ sign_up_window	AnchorPane
Ⓕ️ ↳ remember_me_choicebox	JFXCheckBox
Ⓣ️ ↳ user_properties	UserProperties
Ⓕ️ ↳ signup_error_label	Label
Ⓕ️ ↳ login_hyperlink	Hyperlink
Ⓕ️ ○ valid_password_flag	boolean
Ⓕ️ ○ valid_username_flag	boolean
Ⓕ️ ↳ signup_username_field	TextField
Ⓕ️ ↳ sign_up_button	JFXButton
Ⓕ️ ↳ password_field	PasswordField
Ⓕ️ ↳ signup_email_field	TextField
Ⓕ️ ↳ create_account_hyperlink	Hyperlink
Ⓕ️ ↳ signup_password_field	PasswordField
Ⓜ️ ↳ usernameValidation(String)	boolean
Ⓜ️ ↳ emailPatternValidation(String)	boolean
Ⓜ️ ↳ onButtonHover(JFXButton, double, double, String)	void
Ⓜ️ ↳ dataValidation(String, String, String)	void
Ⓜ️ ↳ showWindow()	void
Ⓜ️ ↳ generateUserId()	String
Ⓜ️ ↳ emailValidation(String)	boolean
Ⓜ️ ↳ onSignUp()	void
Ⓜ️ ↳ passwordValidation(String)	boolean
Ⓜ️ ↳ initialize(URL, ResourceBundle)	void
Ⓜ️ ↳ onLogIn()	void

Rysunek 2.17: Struktura klasy LogInWindowController.

W sumie klasa zawiera 11 metod:

- **public void initialize(URL, ResourceBundle).** Metoda inicjalizacji, ustawia obsługę zdarzeń dla przycisków i hiperłączy oraz ustawia efekty po najechaniu kursorem myszy na przyciski.
- **private void onLogIn().** Metoda obsługująca akcję po kliknięciu przycisku "LOGIN". Sprawdza, czy wprowadzone przez użytkownika dane są poprawne i otwiera menu główne, jeśli autoryzacja się udała.
- **private void onSignUp().** Metoda obsługująca akcję po kliknięciu przycisku "SIGN UP". Sprawdza, czy wprowadzone dane są poprawne, aby zarejestrować nowego użytkownika i otwiera menu główne, jeśli się uda.
- **private void onButtonHover(JFXButton, double, double, String).** Metoda do ustawiania efektu powiększenia i zmiany koloru przycisków po najechaniu kursorem myszy.
- **private @NotNull String generateUserId().** Metoda generująca unikalny identyfikator użytkownika na podstawie bieżącego czasu i losowej liczby.
- **private boolean usernameValidation(String).** Metoda weryfikująca unikalność nazwy użytkownika.
- **private boolean emailValidation(String).** Metoda sprawdzająca unikalność adresu e-mail użytkownika.
- **private boolean emailPatternValidation(String).** Metoda sprawdzająca poprawność formatu adresu e-mail.
- **private boolean passwordValidation(String).** Metoda sprawdzająca złożoność hasła względem określonych kryteriów.
- **private void dataValidation(String, String, String).** Metoda sprawdzająca poprawność wprowadzonych danych podczas rejestracji nowego użytkownika.
- **private void showWindow().** Metoda wyświetlająca okno rejestracji podczas przełączania między rejestracją a logowaniem.

2.2.17 Klasa MainWindowController

Jest kontrolerem głównego okna aplikacji. Głównymi zadaniami tej klasy jest wyświetlanie zawartości w głównym oknie, czyli ładowanie różnych widoków do głównego panelu aplikacji w zależności od działań użytkownika, a także zmiana widoczności elementów interfejsu podczas wchodzenia i wychodzenia z trybu pełnoekranowego odtwarzacza multimedialnego. Implementuje interfejs Initializable, który umożliwia inicjalizację komponentów podczas ładowania okna.

W sumie klasa zawiera jedną metodę:

- **public void initialise(URL, ResourceBundle).** Ustawia obsługę zdarzeń dla różnych akcji użytkownika, takich jak wybór opcji menu – w zależności od nowo wybranej opcji menu, odpowiedni widok jest ładowany do głównego panelu aplikacji. Zmienia również stany trybu pełnoekranowego odtwarzacza multimedialnego – w zależności od stanu trybu pełnoekranowego zmienia się widoczność elementów interfejsu.

©	>MainWindowController
(m)	MainWindowController()
(f)	parent_anchorpance AnchorPane
(m)	initialize(URL, ResourceBundle) void

Rysunek 2.18: Struktura klasy MainWindowController.

2.2.18 Klasa App

Jest punktem wejścia do aplikacji i dziedziczy z klasy Application dostarczonej przez JavaFX do tworzenia aplikacji graficznych. Klasa ta definiuje metodę start(Stage), która jest wywoływana po uruchomieniu aplikacji, oraz metodę main(String[]), która jest punktem wejścia do aplikacji Java.

⌚	App
(m)	App()
(f)	user_properties UserProperties
(m)	start(Stage) void
(m)	main(String[]) void

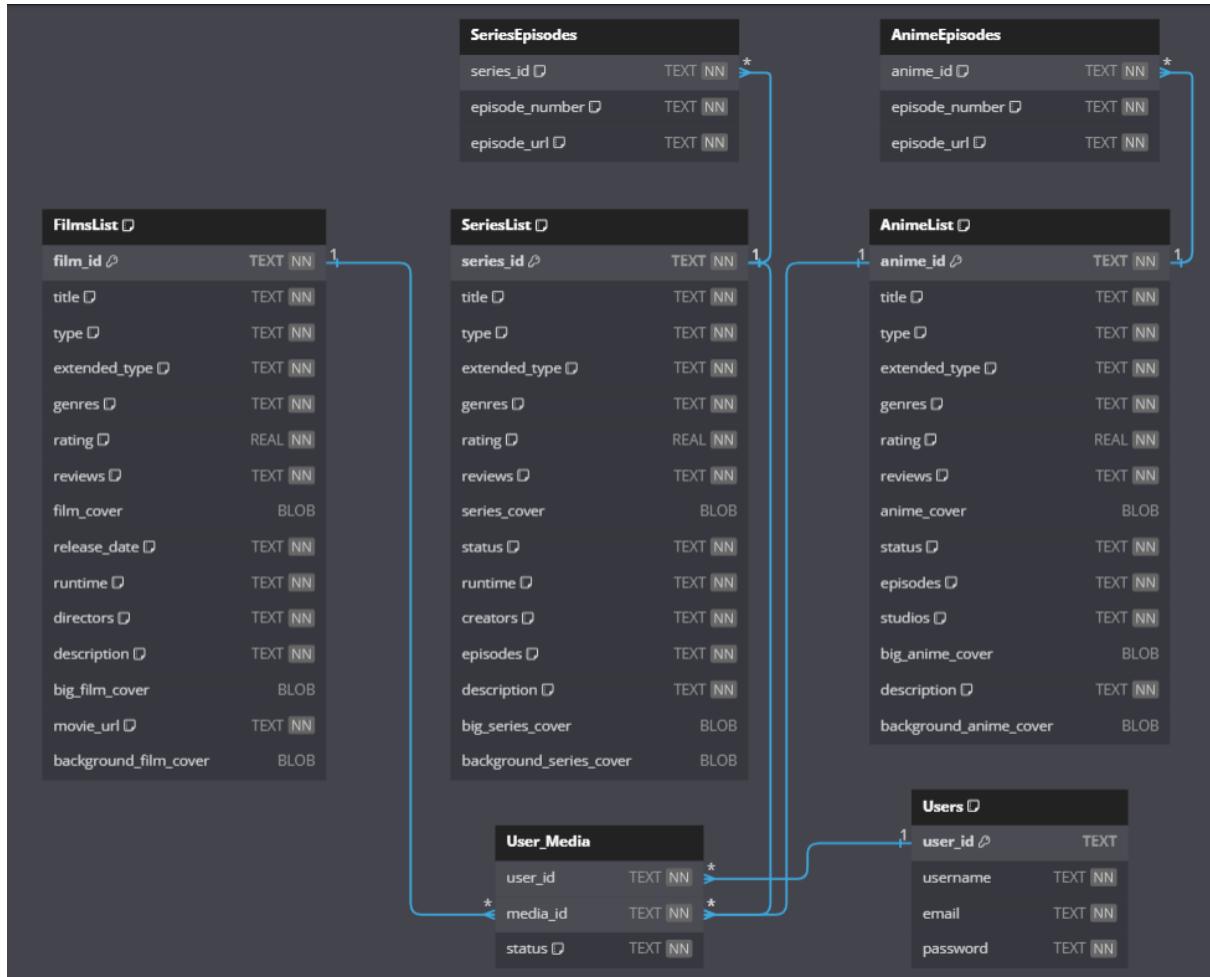
Rysunek 2.19: Struktura klasy App.

W sumie zawiera dwie metody:

- **public void start(Stage).** Metoda wywoływana podczas uruchamiania aplikacji. Metoda ta sprawdza, czy użytkownik jest pamiętany (jeśli wcześniej wybrał opcję "Remember me"). Jeśli użytkownik jest pamiętany, dane użytkownika są ładowane i wyświetlane jest główne okno aplikacji, w przeciwnym razie wyświetlane jest okno logowania. Listener dla flagi logout jest również ustalony, aby śledzić, kiedy użytkownik jest wylogowany i wykonywać odpowiednie akcje.
- **public static void main(String[]).** Punkt wejścia do aplikacji Java. Metoda ta wywołuje metodę launch(), która uruchamia aplikację JavaFX.

2.3 Baza danych i zarządzanie danymi

Jako bazę danych wykorzystano SQLite w wersji 3.35.5. W sumie składa się ona z 7 tabel (**FilmsList**, **SeriesList**, **AnimeList**, **SeriesEpisodes**, **AnimeEpisodes**, **Users** i **User_Media**), które umożliwiają przechowywanie i zarządzanie informacjami o różnych typach mediów, a także śledzenie interakcji użytkowników z tymi mediami.



Rysunek 2.20: Diagram bazy danych.

- **FilmsList.** Ta tabela służy do przechowywania informacji o filmach. Zawiera dane takie jak tytuł, typ, gatunek, ocena, recenzje, okładki, data premiery, czas trwania, reżyserzy, opis i adres URL filmu.
- **SeriesList.** Ta tabela zawiera informacje o serialach. Przechowuje dane o tytule, typie, gatunkach, ocenie, recenzjach, okładce, statusie, czasie trwania, twórcach, opisie i liczbie epizodów serialu.
- **AnimeList.** Służy do przechowywania informacji o anime. Zawiera tytuł, typ, gatunki, ocenę, recenzje, okładkę, status, liczbę epizodów, studia, opis.
- **SeriesEpisodes.** Ta tabela służy do przechowywania informacji o epizodach serialu. Zawiera identyfikator epizodu, numer epizodu i adres URL epizodu. Umożliwia powiązanie każdego serialu telewizyjnego z jego epizodami.
- **AnimeEpisodes.** Tabela przechowuje dane o epizodach anime. Zawiera identyfikator anime, numer epizodu i adres URL epizodu. Umożliwia powiązanie każdego anime z jego epizodami.

- **Users.** Zawiera informacje o użytkownikach bazy danych. Zawiera identyfikator użytkownika, nazwę użytkownika, e-mail i hasło. Ta tabela jest używana do zarządzania użytkownikami i autoryzacji.
- **User_Media.** Ta tabela służy do przechowywania informacji o interakcjach użytkowników z treścią multimedialnymi. Zawiera identyfikator użytkownika, identyfikator mediów i status. Pozwala śledzić, jakie treści użytkownicy oglądają lub zapisują.

2.4 Minimalne wymagania sprzętowe

Aby aplikacja mogła uruchamiać się i działać bez przerw, urządzenie, na którym ona będzie uruchamiana, musi spełniać minimalne wymagania.

- **System operacyjny:**

- Windows: Windows 7 SP1 lub nowsze wersje;
- MacOS: macOS 10.12 Sierra lub nowszy;
- Linux: dowolna nowoczesna dystrybucja z obsługą Java.

- **Wymagania sprzętowe:**

- Procesor: Procesor dwurdzeniowy o częstotliwości taktowania co najmniej 1,5 GHz;
- Pamięć RAM: Minimum 2 GB pamięci RAM, zalecane 4 GB lub więcej;
- Miejsce na dysku: Minimum 500 MB wolnego miejsca na instalację aplikacji i bazy danych.Więcej niż 1 GB jest zalecane do przechowywania dodatkowych danych i kopii zapasowych;
- Grafika: Karta graficzna z obsługą OpenGL 2.0 lub wyższą (zazwyczaj odpowiednie są zintegrowane rozwiązania graficzne);
- Ekran: Rozdzielcość ekranu co najmniej 1024x768 pikseli.

- **Oprogramowanie:**

- Java Runtime Environment (JRE): Java 11 lub nowsza wersja.

- **Biblioteki i struktury:**

- SQLite JDBC Driver w wersji 3.44.1.0;
- FontAwesomeFX w wersji 4.7.0-9.1.2;
- JFoenix w wersji 9.0.10;
- Apache Log4j wersja 2.20.0.

- **Inne wymagania:**

- Połączenie internetowe: do pełnego działania aplikacji i multimediów;
- Prawa dostępu: uprawnienia administratora do zapisu w katalogu instalacyjnym aplikacji i bazy danych oraz możliwość instalacji wymaganych zależności.

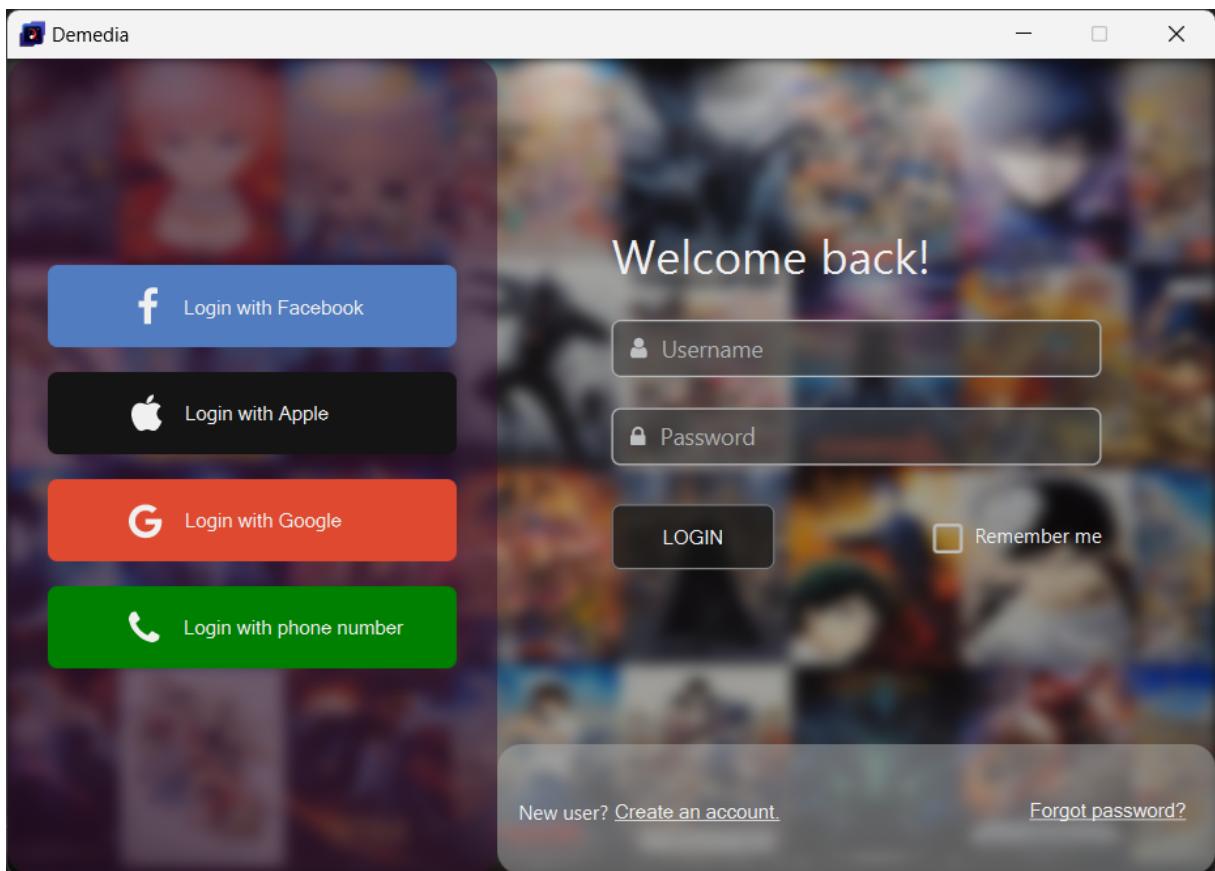
Rozdział 3

Prezentacja warstwy użytkowej projektu

3.1 Okno logowania/rejestracji

3.1.1 Logowanie

Przy pierwszym uruchomieniu programu, na samym początku użytkownik witany jest oknem logowania.



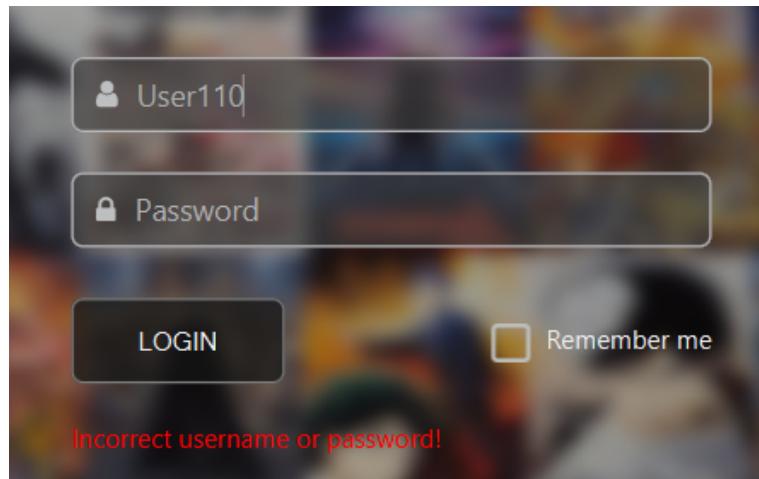
Rysunek 3.1: Okno logowania.

Użytkownik ma kilka możliwości:

- wprowadzić swoją nazwę użytkownika i hasło, jeśli posiada konto i zalogować się klikając na przycisk "LOGIN".
- kliknąć przycisk "Remember me", aby przejść bezpośrednio do głównego okna aplikacji przy następnym uruchomieniu programu.

- utworzyć konto, jeśli go nie posiada lub jeśli potrzebne jest nowe konto. Kliknięcie hiperłącza "Create an account." otworzy okno rejestracji użytkownika.

Jeśli użytkownik wprowadził nieprawidłowe dane, otrzyma odpowiedni komunikat i nie będzie mógł się zalogować.



Rysunek 3.2: Komunikat o nieprawidłowo wprowadzonych danych podczas logowania.

Inne opcje, takie jak "Forgot password?", "Login with Facebook" itp. są tymczasowo niedostępne i mają zostać zaimplementowane w przyszłych wersjach aplikacji.

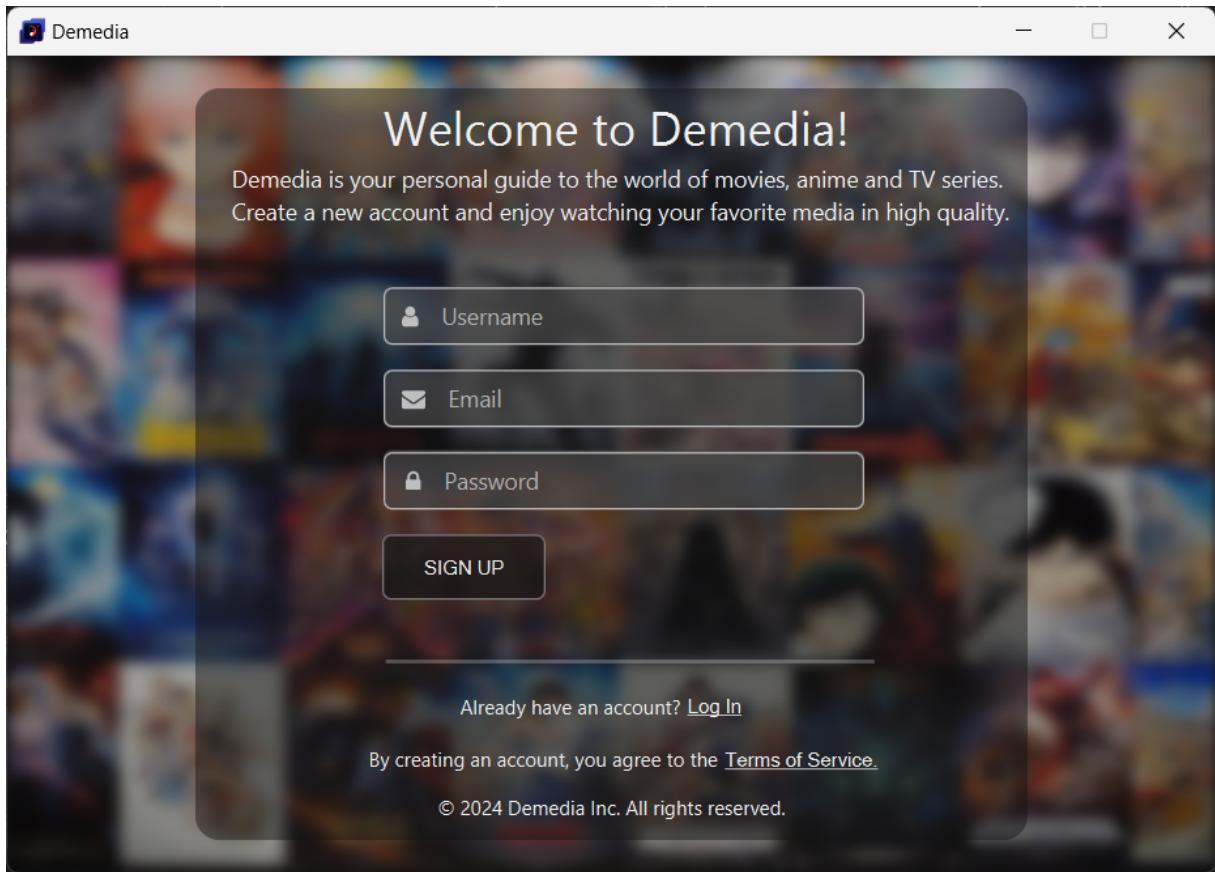
3.1.2 Rejestracja

Kliknięcie hiperłącza "Create an account." otworzy okno, w którym użytkownik może się zarejestrować.

Aby utworzyć nowe konto, użytkownik musi podać nazwę użytkownika, adres e-mail i hasło. Wprowadzone dane muszą spełniać wymagania:

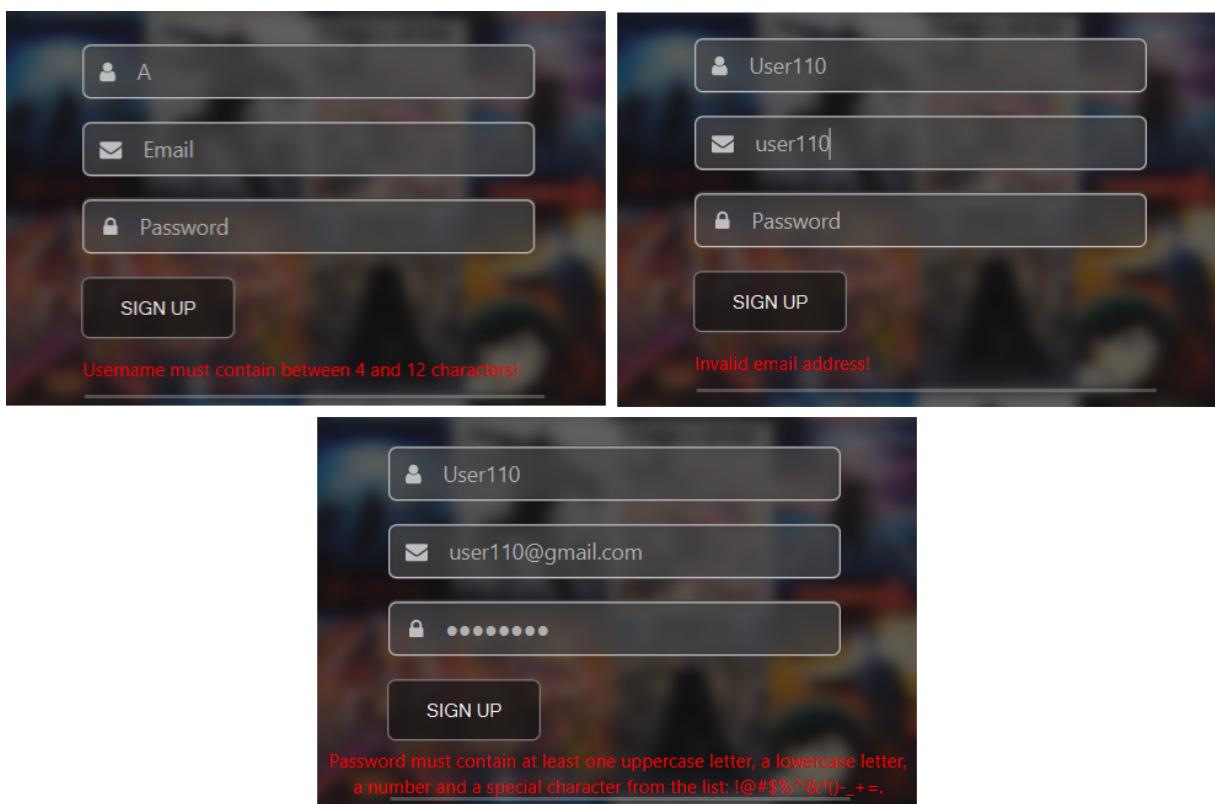
- Nazwa użytkownika o długości od 4 do 12 znaków;
- Prawdziwy adres e-mail (na tym etapie aplikacji wystarczy podać na końcu @gmail.com);
- Silne hasło o długości od 8 do 16 znaków. Ponadto hasło musi zawierać co najmniej jedną wielką literę, małą literę, cyfrę i znak specjalny z listy: !@#\$%&*()-_=^ .

Jeśli użytkownik podał unikalne dane i pomyślnie przeszedł weryfikację, jego konto zostanie pomyślnie utworzone i dodane do bazy danych. Po naciśnięciu przycisku "SIGN UP" użytkownik przejdzie do głównego okna aplikacji.



Rysunek 3.3: Okno rejestracji.

Jeśli użytkownik wprowadził już istniejące dane lub nie przeszedł walidacji, zostanie wyświetlony odpowiedni komunikat.



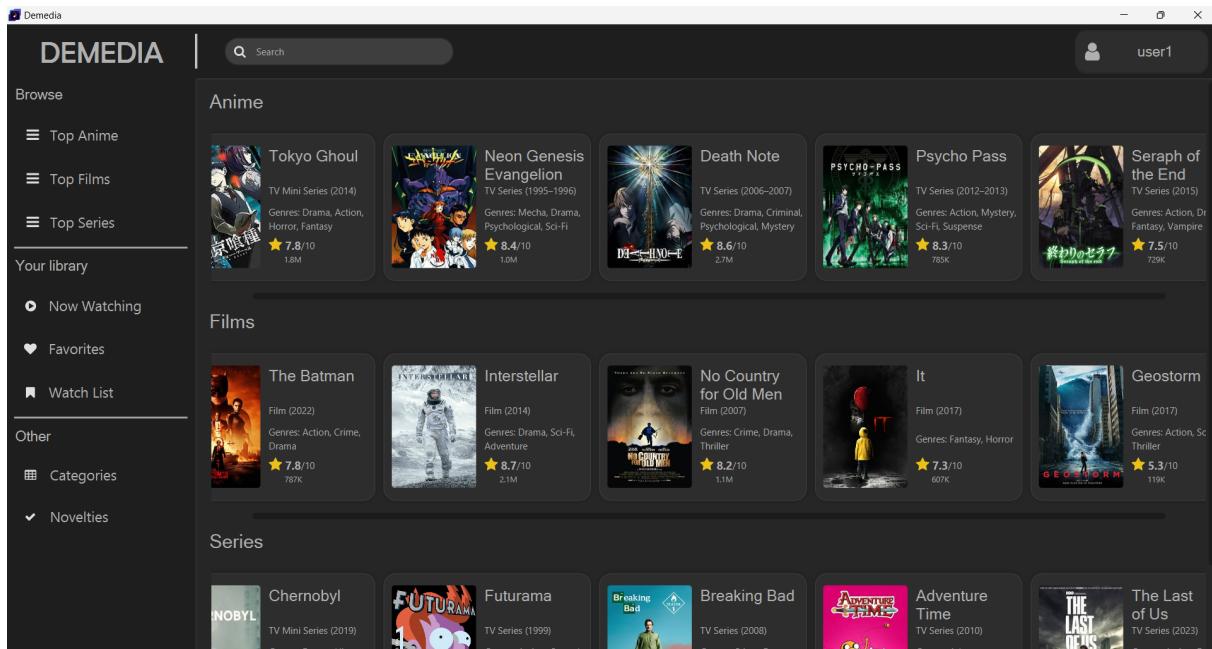
Rysunek 3.4: Komunikaty o nieprawidłowo wprowadzonych danych podczas rejestracji.

Ponadto, jeśli użytkownik zmieni zdanie na temat tworzenia nowego konta lub po prostu chce zalogować się na istniejące konto, może wrócić do okna logowania, klikając hiperłącze "Log In".

Hiperłącze "Terms of Service." jest tymczasowo niedostępne i zostanie zaimplementowane w przyszłych wersjach aplikacji.

3.2 Główne okno aplikacji

Po pomyślnym zalogowaniu lub rejestracji użytkownik zostaje przeniesiony do głównego okna aplikacji.



Rysunek 3.5: Główne okno aplikacji.

Użytkownika wita centralny panel z trzema sekcjami polecanych mediów. Każda sekcja zawiera banery dla określonego typu mediów (anime, filmy i seriale telewizyjne). Każdy baner zawiera krótkie informacje na temat mediów, takie jak okładka, tytuł, typ, gatunek, ogólna ocena i liczba ocen użytkowników.

Centralny panel lub sekcje można przewijać odpowiednio w góre/dół lub w lewo/prawo.

Oprócz centralnego panelu, główne okno zawiera górny i lewy panel, które zapewniają użytkownikowi dodatkowe funkcje i nawigację.

3.3 Górnny panel aplikacji

Górny panel zawiera przyciski i elementy, które zapewniają użytkownikowi dodatkową funkcjonalność.



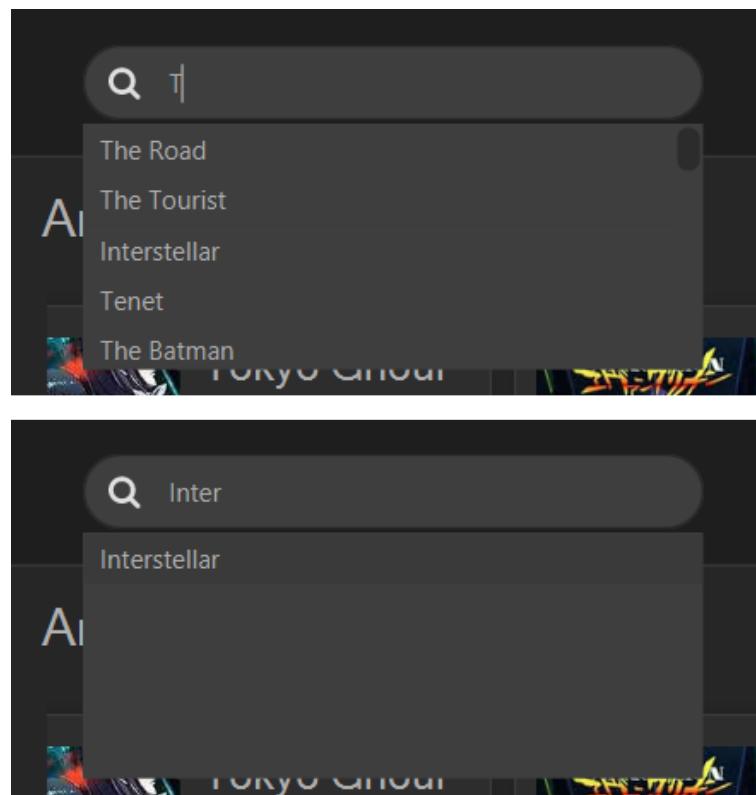
Rysunek 3.6: Górnny panel aplikacji.

3.3.1 Przycisk panelu centralnego

Przycisk z logo "DEMEDIA" po naciśnięciu przekierowuje użytkownika do domyślnego panelu centralnego, niezależnie od tego, w której sekcji się znajduje.

3.3.2 Pole wyszukiwania multimedów

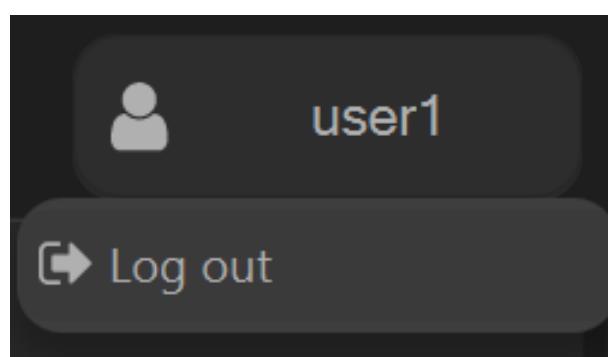
Bardzo przydatnym narzędziem jest pole wyszukiwania. Użytkownik może wyszukać dowolny typ multimedów, po prostu wprowadzając tekst. Program automatycznie wybierze odpowiednie tytuły multimedów na podstawie wprowadzonego tekstu i dynamicznie wyświetli je na liście rozwijanej. Po kliknięciu odpowiedniego tytułu multimedów na liście rozwijanej, użytkownik zostanie przekierowany na stronę z wybranym multimediami.



Rysunek 3.7: Wyszukiwanie multimedów według tytułu.

3.3.3 Przycisk profilu użytkownika

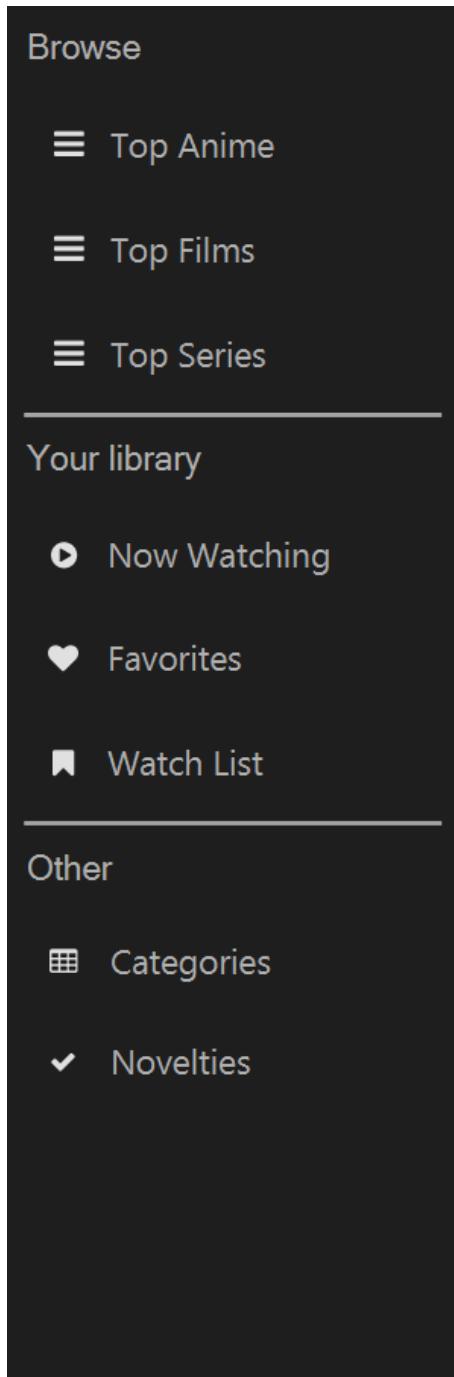
Ostatnim elementem górnego panelu jest przycisk profilu użytkownika. Domyślnie przycisk wyświetla nazwę użytkownika. Po kliknięciu przycisku pojawia się rozwijane menu z jedyną opcją wyjścia z głównego okna. Po kliknięciu oferowanej opcji użytkownik zostanie przekierowany do okna logowania, a wartość "Remember me" zostanie anulowana, jeśli była wcześniej wskazana.



Rysunek 3.8: Wybór opcji "Log out".

3.4 Lewy panel aplikacji

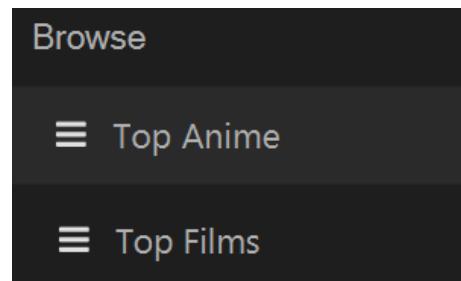
Lewy panel zapewnia użytkownikowi możliwość poruszania się po różnych sekcjach aplikacji. Lewy panel zawiera trzy sekcje: "Browse", "Your library" i "Other", z których każda zawiera podsekcje prowadzące do określonych stron.



Rysunek 3.9: Lewy panel aplikacji.

3.4.1 Sekcja "Browse"

Ta sekcja udostępnia użytkownikowi trzy podsekcje: "Top Anime", "Top Films", "Top Series"). Kliknięcie każdego przycisku spowoduje otwarcie odpowiedniej podsekcji ze wszystkimi dostępnymi banerami.



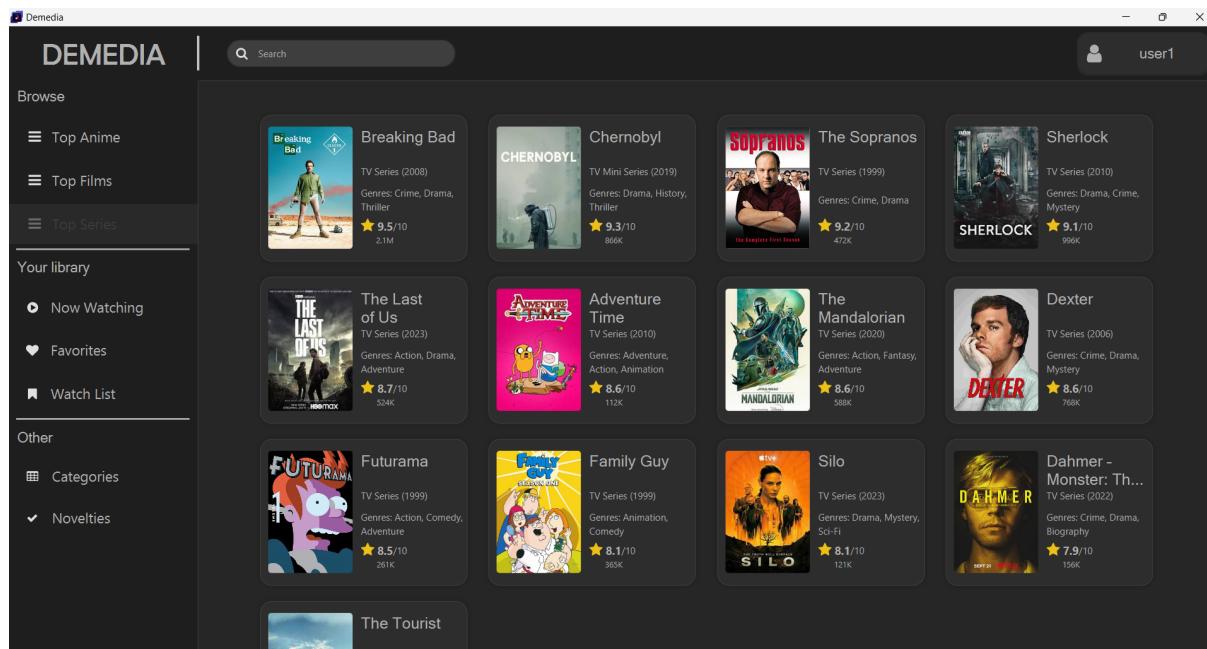
Rysunek 3.10: Naciśnięcie przycisku podsekcji.

Series	Rating	Genre	Details
Steins;Gate	9.0/10	Sci-Fi, Drama, Psychological, Thriller	TV Series (2011)
Death Note	8.6/10	Drama, Criminal, Psychological, Mystery	TV Series (2006–2007)
Neon Genesis Evangelion	8.4/10	Mecha, Drama, Psychological, Sci-Fi	TV Series (1995–1996)
Psycho Pass	8.3/10	Action, Mystery, Sci-Fi, Suspense	TV Series (2012–2013)
The Rising of the Shield Hero	7.9/10	Action, Drama, Fantasy, Adventure	TV Series (2019)
Tokyo Ghoul	7.8/10	Drama, Action, Horror, Fantasy	TV Mini Series (2014)
Memories	7.7/10	Drama, Horror, Sci-Fi, Psychological	TV Series (1995)
Shiki	7.7/10	Horror, Mystery, Supernatural, Suspense	TV Series (2010)
Seraph of the End	7.5/10	Action, Drama, Fantasy, Vampire	TV Series (2015)
Akame ga Kill!	7.5/10	Drama, Action, Fantasy	TV Series (2014)
Is It Wrong to Try to Pick Up Girls in a Dungeon?	7.5/10	Drama, Adventure	TV Series (2015)
Another	7.4/10	Horror, Mystery, Supernatural	TV Mini Series (2012)
Kiznaiver			

Rysunek 3.11: Podsekcja "Top Anime".

Film	Rating	Genre	Details
Interstellar	8.7/10	Drama, Sci-Fi, Adventure	Film (2014)
2001: A Space Odyssey	8.3/10	Adventure, Sci-Fi	Film (1968)
No Country for Old Men	8.2/10	Crime, Drama, Thriller	Film (2007)
Blade Runner	8.1/10	Action, Drama, Sci-Fi	Film (1982)
The Batman	7.8/10	Action, Crime, Drama	Film (2022)
Skyfall	7.8/10	Action, Thriller, Adventure	Film (2012)
The Machinist	7.6/10	Drama, Thriller	Film (2004)
A Quiet Place	7.5/10	Drama, Horror, Sci-Fi	Film (2018)
IT	7.3/10	Fantasy, Horror	Film (2017)
Tenet	7.3/10	Action, Sci-Fi, Thriller	Film (2020)
The Road	7.2/10	Drama, Thriller	Film (2009)
Gold	5.4/10	Action, Thriller, Adventure	Film (2022)
Geostorm			

Rysunek 3.12: Podsekcja "Top Films".

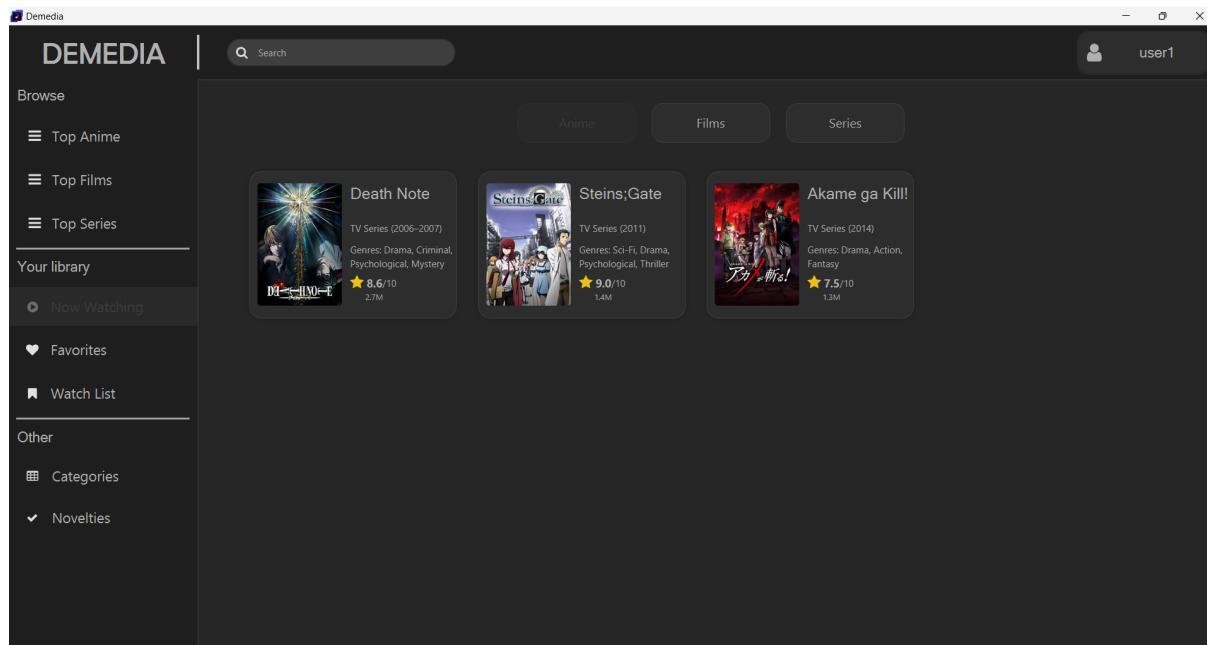


Rysunek 3.13: Podsekcja "Top Series".

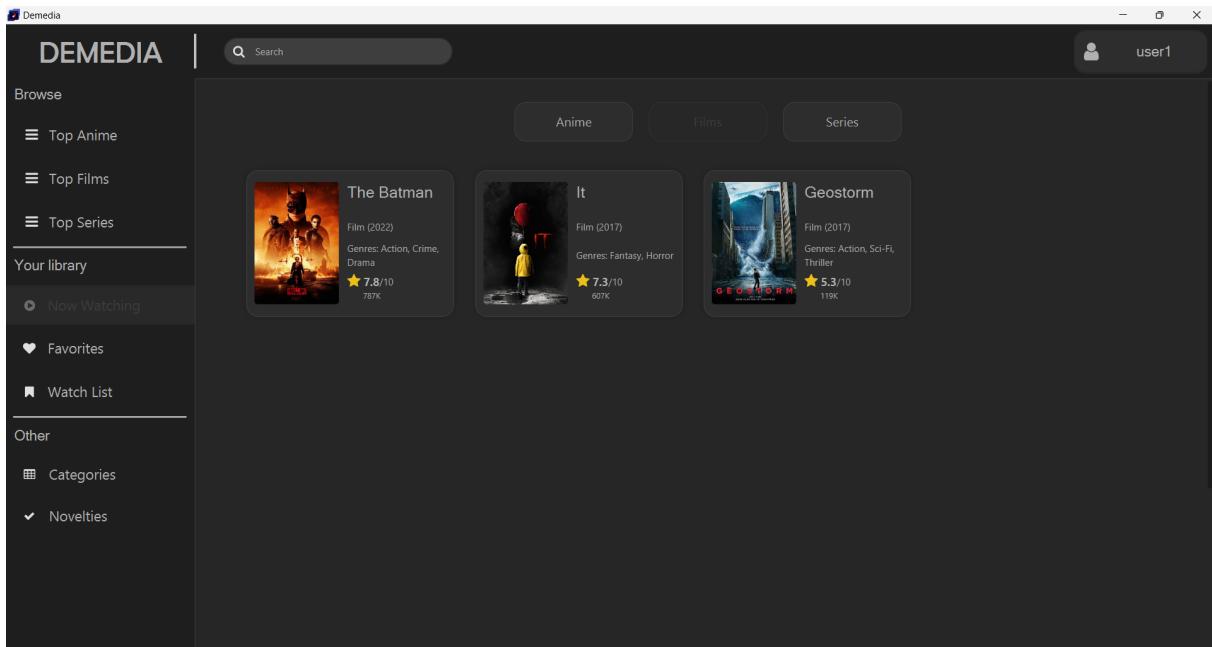
3.4.2 Sekcja "Your library"

Sekcja "Your library" jest osobistą biblioteką użytkownika. Zawiera ona trzy podsekcje: "Now Watching", "Favorites" i "Watch List". Kliknięcie każdego z przycisków spowoduje otwarcie odpowiedniej podsekcji ze wszystkimi materiałami, które użytkownik dodał do swojej biblioteki.

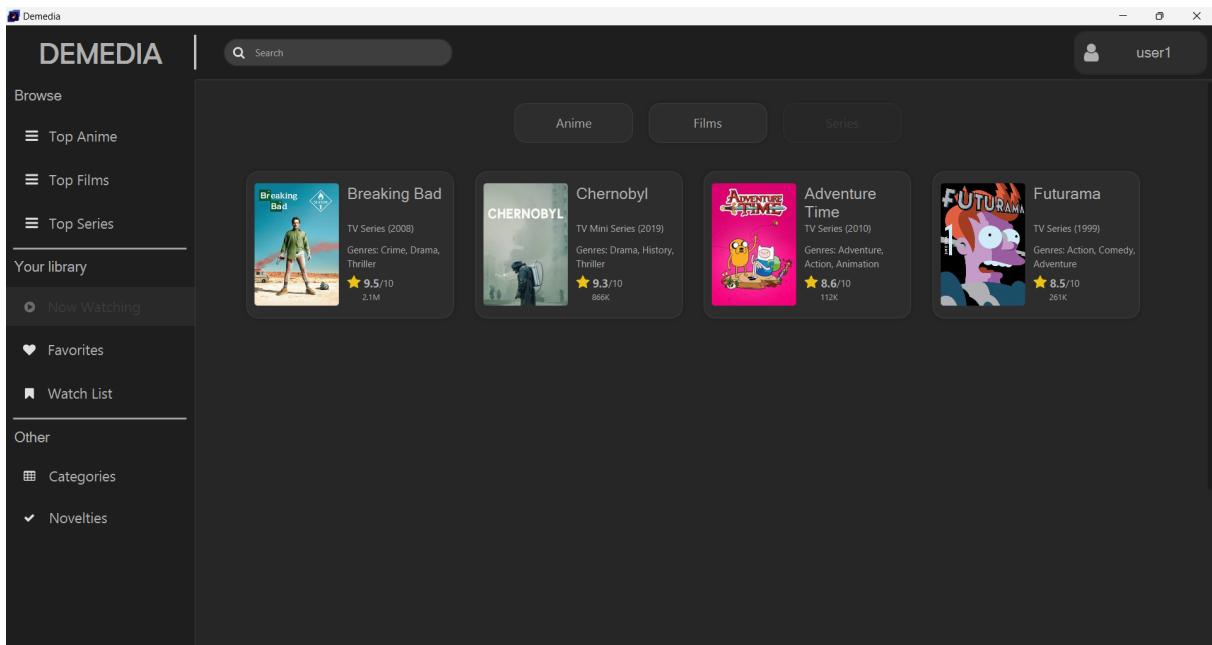
W celu zwięzłego i uporządkowanego wyświetlania banerów w tych podsekcjach, możliwe jest wyświetlanie tylko określonych typów banerów za pomocą specjalnych przycisków nad nimi. Domyślnie wyświetlane są banery anime.



Rysunek 3.14: Wyświetlanie banerów anime w podsekcji "Now Watching".



Rysunek 3.15: Wyświetlanie banerów anime w podsekcji "Now Watching".

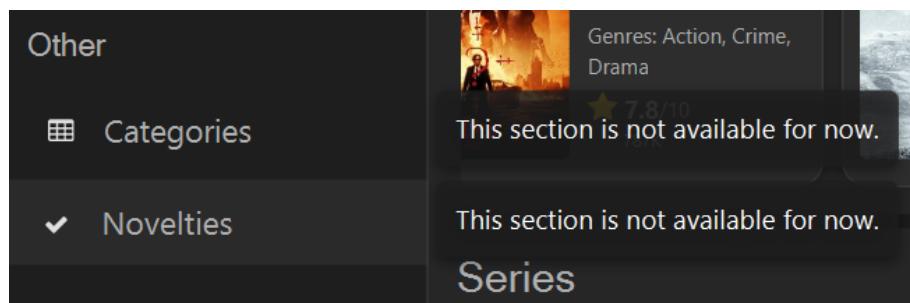


Rysunek 3.16: Wyświetlanie banerów z serialami telewizyjnymi w podsekcji "Now Watching".

3.4.3 Sekcja "Other"

Ta sekcja udostępnia użytkownikowi dwie podsekcje: "Categories" i "Novelties".

W chwili obecnej podsekcje te nigdzie nie prowadzą, ponieważ ich implementacja zostanie dodana w przyszłych wersjach aplikacji. Po kliknięciu przycisków użytkownikowi zostanie wyświetlony odpowiedni komunikat.



Rysunek 3.17: Komunikaty po naciśnięciu przycisków podsekcji "Categories" i "Novelties".

3.5 Strona multimedialna

Po kliknięciu każdego banera, niezależnie od tego, w której podsekcji się znajduje, w centralnym panelu zawsze otwiera się strona wybranego medium.



Rysunek 3.18: Kliknięcie banera.

Strona ta zawiera wszystkie informacje o multimediu, od tytułu po pełny opis. Ponadto strona zawiera takie elementy, jak przyciski do dodawania multimediiów do biblioteki użytkownika i odtwarzacz multimediiów.

Death Note

Status: Finished. Oct 3, 2006 - Jun 26, 2007
Type: TV Series
Episodes: 37
Runtime: 23 min. per ep.
Genres: Drama, Criminal, Psychological, Mystery
Studios: Madhouse
Rating: 8.6/10

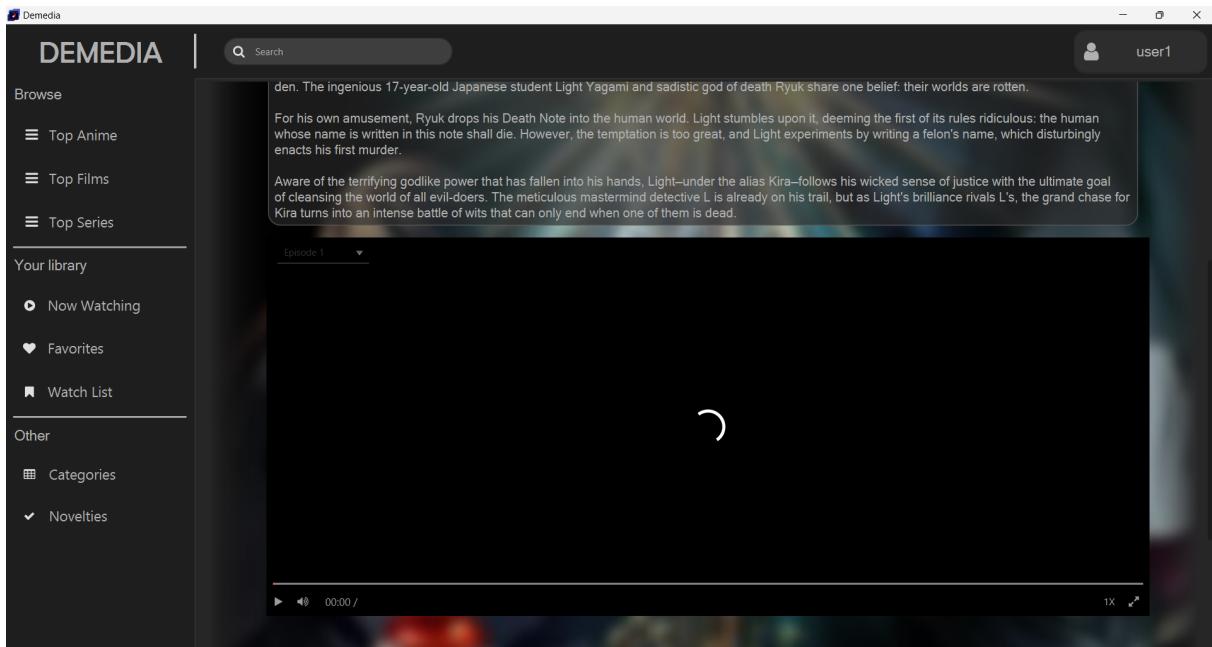
Description

Brutal murders, petty thefts, and senseless violence pollute the human world. In contrast, the realm of death gods is a humdrum, unchanging gambling den. The ingenious 17-year-old Japanese student Light Yagami and sadistic god of death Ryuk share one belief: their worlds are rotten.

For his own amusement, Ryuk drops his Death Note into the human world. Light stumbles upon it, deeming the first of its rules ridiculous: the human whose name is written in this note shall die. However, the temptation is too great, and Light experiments by writing a felon's name, which disturbingly enacts his first murder.

Aware of the terrifying godlike power that has fallen into his hands, Light—under the alias Kira—follows his wicked sense of justice with the ultimate goal of cleansing the world of all evil-doers. The meticulous mastermind detective L is already on his trail, but as Light's brilliance rivals L's, the grand chase for Kira turns into an intense battle of wits that can only end when one of them is dead.

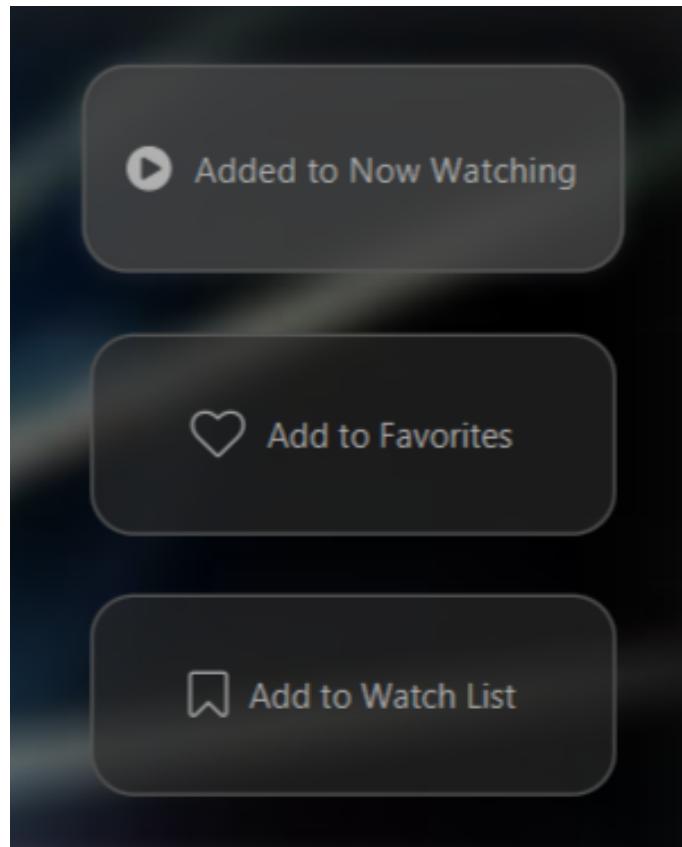
Rysunek 3.19: Strona multimedialna, cz. 1.



Rysunek 3.20: Strona multimedialna, cz. 2.

3.5.1 Przyciski do dodawania multimedów do biblioteki użytkownika

Podczas ładowania strony multimedów ładowane są również informacje o tym, czy użytkownik dodał multimedia do swojej biblioteki. Jeśli użytkownik dodał anime, film lub serial telewizyjny, np. do listy "Now Watching", odpowiedni przycisk będzie miał inny styl niż pozostałe przyciski, a informacje o dodaniu zostaną przesłane do bazy danych.

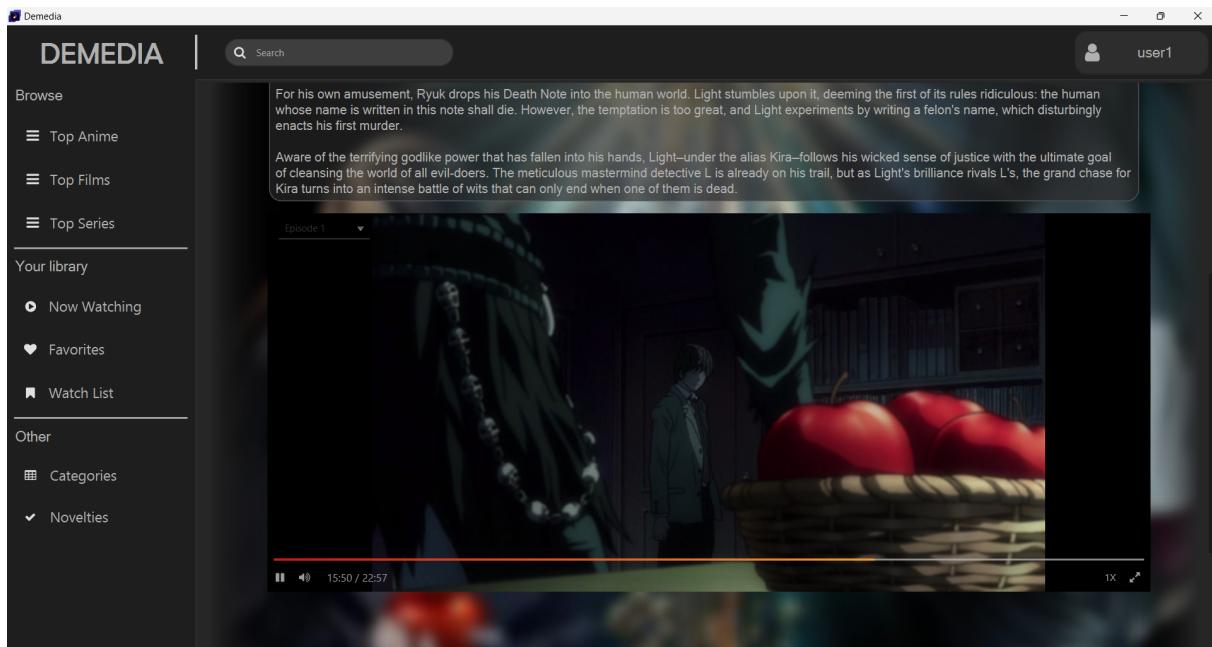


Rysunek 3.21: Użytkownik dodał anime do listy "Now Watching".

Użytkownik może dodać multimedia do wszystkich trzech list jednocześnie lub je usunąć, co spowoduje usunięcie odpowiednich informacji z bazy danych i przywrócenie domyślnego stylu przycisku.

3.5.2 Odtwarzacz multimedialny

Użytkownik może oglądać anime, filmy lub seriale telewizyjne w dedykowanym odtwarzaczu multimedialnym. Odtwarzacz multimedialny automatycznie ładuje zasób po otwarciu strony i rozpoczyna odtwarzanie multimedialnych, gdy są gotowe.



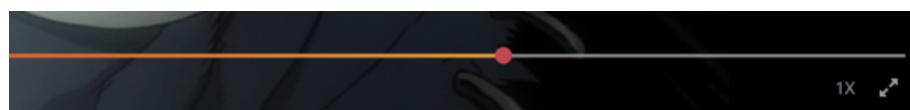
Rysunek 3.22: Wyświetlanie wideo.

W dolnym panelu odtwarzacza multimedialnego użytkownik może wstrzymywać/odtwarzać wideo i zmieniać prędkość odtwarzania. Użytkownik może również zmienić głośność wideo.



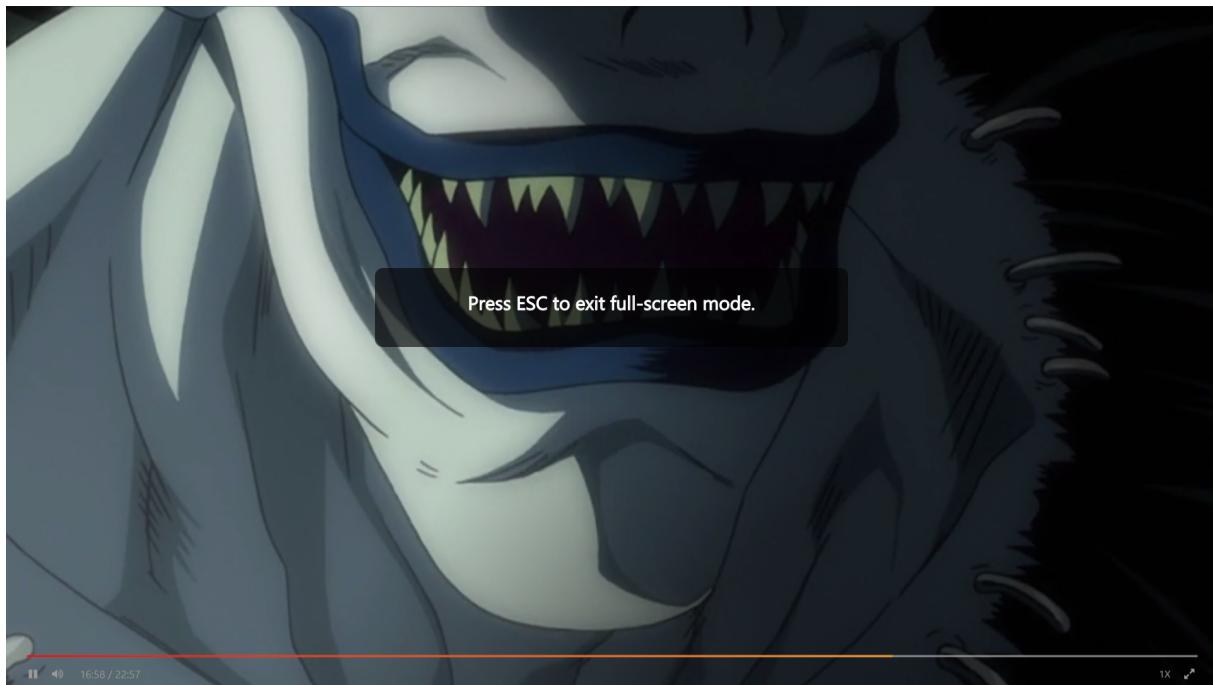
Rysunek 3.23: Regulacja głośności.

W razie potrzeby może przewijać wideo do tyłu lub do przodu za pomocą slider'a czasu.



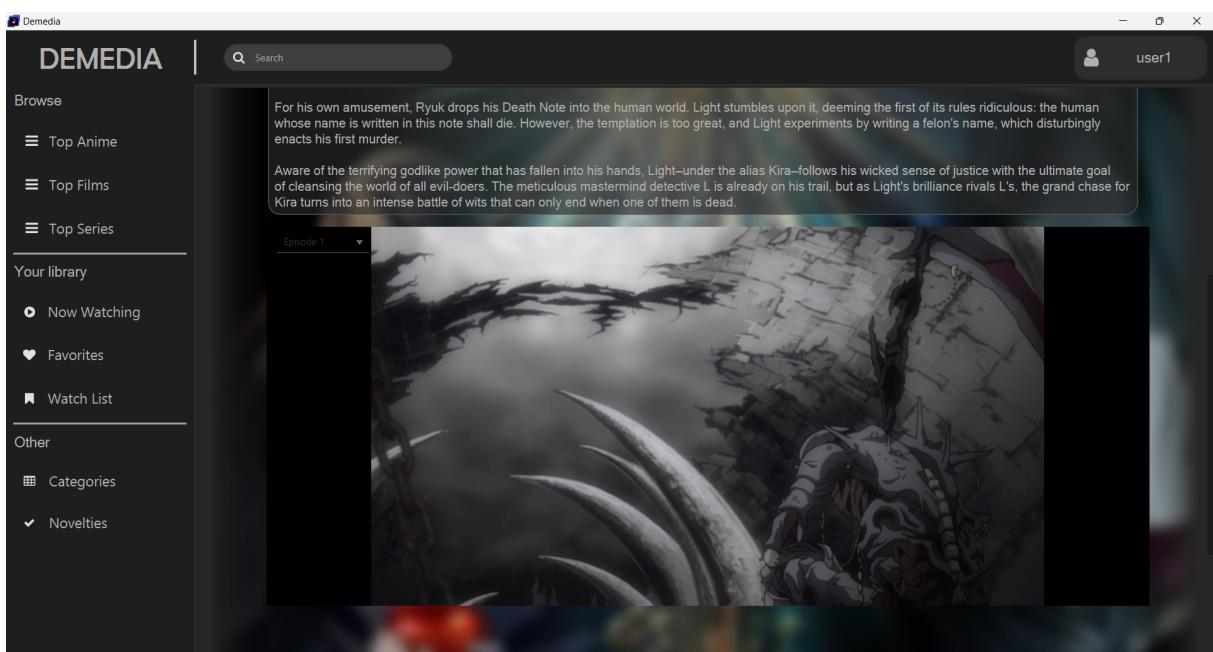
Rysunek 3.24: Bieżąca pozycja slider'a czasu.

Odtwarzacz multimedialny zapewnia również użytkownikowi widok pełnoekranowy, w którym dostępne będą te same funkcje.



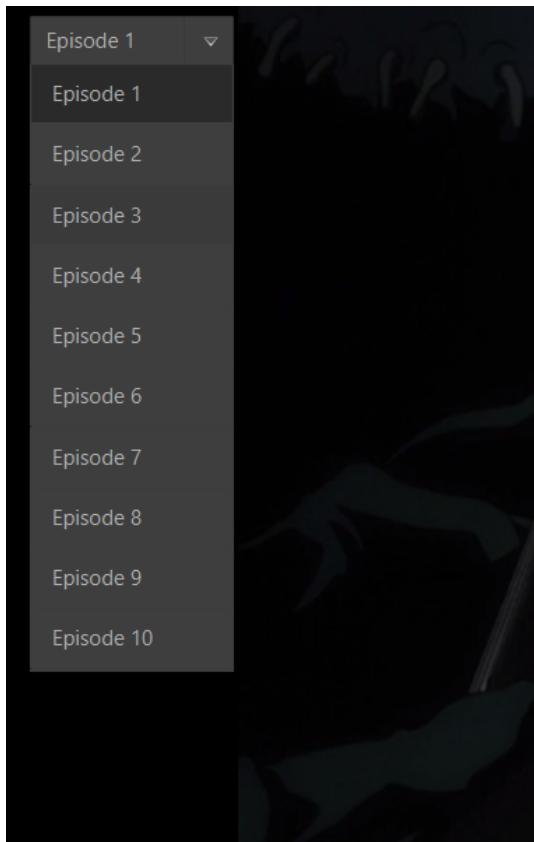
Rysunek 3.25: Wideo w trybie pełnoekranowym.

Jeśli użytkownik nie używa myszy przez 3 sekundy, dolny pasek odtwarzacza multimediiów i kursor są ukrywane, aby umożliwić użytkownikowi swobodne przeglądanie.



Rysunek 3.26: Odtwarzacz multimedialny z ukrytym dolnym panelem i kursorem myszy.

Ponadto, w zależności od typu multimedów, rozwijana lista epizodów jest dostępna w lewym górnym rogu, jeśli użytkownik ogląda anime lub serial telewizyjny. Użytkownik może przewijać listę w górę/dół za pomocą kółka myszy lub klawiszy. W przypadku filmów lista ta nie jest dostępna.



Rysunek 3.27: Lista epizodów.

Rozdział 4

Podsumowanie

W ramach tego projektu stworzono w pełni funkcjonalną aplikację do przeglądania multimediiów z ogromną ilością funkcji dla użytkownika. Pomimo włożonej pracy, aplikacja ta nie jest całkowicie doskonała. W przyszłym rozwoju projektu planowane jest dodanie wielu innych funkcji, takich jak:

- zapamiętywanie ostatniego epizodu, w którym użytkownik przerwał oglądanie;
- możliwość rejestracji za pośrednictwem Facebook, Apple, Google i numeru telefonu;
- sprawdzanie rzeczywistego istnienia adresu e-mail użytkownika;
- możliwość odzyskania hasła.

Planowane jest również dodanie nowych sekcji, takich jak:

- sekcja "Categories";
- sekcja "Novelties";
- sekcja "Your profile", w której użytkownik będzie mógł zarządzać swoimi danymi;
- sekcja "Settings".

Planowane jest również ulepszenie kodu i wiele innych rzeczy.

Bibliografia

- [1] <https://openjfx.io/javadoc/22/> z dnia 29.05.2024
- [2] <https://www.javatpoint.com/javafx-tutorial> z dnia 29.05.2024
- [3] <https://www.geeksforgeeks.org/singleton-class-java/> z dnia 29.05.2024
- [4] <https://www.baeldung.com/java-singleton> z dnia 29.05.2024
- [5] <https://www.jensd.de/wordpress/?p=132> z dnia 29.05.2024
- [6] <https://www.geeksforgeeks.org/javafx-building-a-media-player/> z dnia 29.05.2024
- [7] https://examples.javacodegeeks.com/java-development/desktop-java/javafx/javafx-media-api/media_code z dnia 29.05.2024
- [8] <https://sqlite.org/lang.html> z dnia 29.05.2024
- [9] <https://www.sqlitetutorial.net/sqlite-java/> z dnia 29.05.2024

Spis rysunków

2.1	Diagram klas programu.	10
2.2	Struktura klasy Banner.	10
2.3	Struktura klasy DatabaseDriver.	12
2.4	Struktura klasy User.	14
2.5	Struktura klasy UserProperties.	14
2.6	Struktura klasy Model.	15
2.7	Struktura klasy MediaPage.	17
2.8	Struktura klasy ViewFactory.	18
2.9	Struktura wyliczenia LeftPanelOptions.	20
2.10	Struktura wyliczenia DatabaseTables.	21
2.11	Struktura klasy CenterPanelController.	21
2.12	Struktura klasy LeftPanelController.	22
2.13	Struktura klasy TopPanelController.	23
2.14	Struktura klasy BrowseSectionController.	24
2.15	Struktura klasy YourLibrarySectionController.	24
2.16	Struktura klasy MediaPageController.	25
2.17	Struktura klasy LogInWindowController.	27
2.18	Struktura klasy MainWindowController.	29
2.19	Struktura klasy App.	29
2.20	Diagram bazy danych.	30
3.1	Okno logowania.	33
3.2	Komunikat o nieprawidłowo wprowadzonych danych podczas logowania.	34
3.3	Okno rejestracji.	35
3.4	Komunikaty o nieprawidłowo wprowadzonych danych podczas rejestracji.	35
3.5	Główne okno aplikacji.	36
3.6	Górny panel aplikacji.	36
3.7	Wyszukiwanie multimedii według tytułu.	37
3.8	Wybór opcji "Log out".	37
3.9	Lewy panel aplikacji.	38
3.10	Naciśnięcie przycisku podsekcji.	39
3.11	Podsekcja "Top Anime".	39
3.12	Podsekcja "Top Films".	39
3.13	Podsekcja "Top Series".	40
3.14	Wyświetlanie banerów anime w podsekcji "Now Watching".	40
3.15	Wyświetlanie banerów anime w podsekcji "Now Watching".	41
3.16	Wyświetlanie banerów z serialami telewizyjnymi w podsekcji "Now Watching".	41
3.17	Komunikaty po naciśnięciu przycisków podsekcji "Categories" i "Novelties".	42
3.18	Kliknięcie banera.	42
3.19	Strona multimedialna, cz. 1.	42
3.20	Strona multimedialna, cz. 2.	43
3.21	Użytkownik dodał anime do listy "Now Watching".	43

3.22 Wyświetlanie wideo.	44
3.23 Regulacja głośności.	44
3.24 Bieżąca pozycja slider'a czasu.	44
3.25 Wideo w trybie pełnoekranowym.	45
3.26 Odtwarzacz multimedialny z ukrytym dolnym panelem i kursorem myszy.	45
3.27 Lista epizodów.	46