POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Piotr Zarębski
Nr albumu 22015

Uniwersalny agregator multimediów dla platform strumieniowych

Praca magisterska napisana
pod kierunkiem:

dr. inż. Mariusz Trzaska

Warszawa, styczeń 2022

Streszczenie

Jednym z podstawowych zasobów dostępnych w Internecie są multimedia, dostępne m.in. za pośrednictwem serwisów strumieniowych. Ze względu na mnogość tego typu platform, uzyskanie informacji na temat dostępności poszczególnych albumów muzycznych, filmów czy seriali jest utrudnione. Praca ta podejmuje się próby rozwiązania tego problemu, poprzez opracowanie generycznego systemu, który byłby w stanie pobierać, przetwarzać oraz prezentować dane pochodzące z wielu serwisów strumieniowych w sposób zautomatyzowany. W jej ramach przybliżono problemy techniczne wiążące się z procesem agregacji treści z platform multimedialnych. W tym celu opracowano prototyp aplikacji webowej, której zadaniem jest gromadzenie informacje na temat pozycji dostępnych w serwisach Spotify YouTube Music, Deezer, Netflix oraz Amazon Prime. Na jej potrzeby dokonano analizy tych usług pod kątem dostępu danych oraz zaproponowano system ich klasyfikacji. Praca ta stanowi również przekrój przez wszystkie fazy projektowania i implementacji omawianego systemu: od zbierania wymagań, po szczegółowy opis konkretnych rozwiązań technicznych, ze szczególnym naciskiem na sposoby ekstrakcji danych z sieci web, takie jak interfejsy API oraz web scraping.

Słowa kluczowe: technologie webowe, serwisy strumieniowe, ekstrakcja danych, API, web scraping

Spis treści

1. Wstęp

Multimedia stanowią jeden z głównych rodzajów danych dostępnych za pośrednictwem sieci web. Jak pokazują statystyki, w ostatnich latach zapotrzebowanie na tego typu treści znacząco wzrosło, a ruch w Internecie zmienił swój charakter z opartego na danych tekstowych na zorientowany na multimedia. Potwierdzają to m.in. dane z badań przeprowadzonych przez firmę Cisco [31], według których w 2017 roku aż 75% ruchu w Internecie stanowiło wideo. Jako jedną z przyczyn takiego stanu wskazać można wzrost popularności serwisów strumieniowych. Przytaczając wyniki raportu [1], w 3 kwartale 2015 roku 29,5% wszystkich treści telewizyjnych oglądanych było za pośrednictwem streamingu. W analogicznym okresie 2020 roku wartość ta wyniosła 42,2%, co stanowi znaczący wzrost. Wraz z dużą popularnością serwisów strumieniowych pojawiła się również potrzeba łatwego i zunifikowanego dostępu do informacji znajdujących się na różnych z nich. Niniejsza praca stanowi próbę zaprojektowania systemu rozwiązującego ten problem poprzez ekstrakcję, agregację oraz przetwarzanie danych pochodzących z różnych platform w sposób zautomatyzowany. W jej ramach przenalizowano usługi Spotify, YouTube Music, Deezer, Netflix i Amazon Prime. W celu prezentacji praktycznego zastosowania uzyskanej wiedzy opracowano prototyp aplikacji webowej, za pomocą którego można w wygodny sposób eksplorować dane dostępne na tych serwisach.

1.1. Potrzeba agregacja danych w Internecie

W ciągu ostatniej dekady nawyki osób korzystających z Internetu uległy zmianie. Według raportu opublikowanego przez firmę The Nielsen Company [2], w 2010 średnia liczba unikalnych domen odwiedzanych w miesiącu przez Amerykanów z dostępem do sieci web wyniosła 89. W 2018 roku, na potrzeby pracy [3], przeprowadzane zostały badania na grupie 850 użytkowników smartfonów, których urządzenia zostały podłączone do serwera proxy monitorującego ich ruch w Internecie na przestrzeni roku. Mediana unikalnych domen odwiedzanych w poszczególnych miesiącach mieści się w zakresie 20-30. Ciężko bezpośrednio zestawić ze sobą oba wyniki, ze względu na różnice ilościowe, metodologiczne i geograficzne. Niemniej, skala rozbieżności świadczyć może o trendzie, w ramach którego osoby korzystające z Internetu odwiedzają coraz mniej różnych witryn. W odpowiedzi na tę potrzebę powstała nowa klasa serwisów internetowych i aplikacji mobilnych, których funkcją nie jest dostarczanie nowych treści, tylko agregacja tych dostępnych publicznie w ramach innych portali. Usługi tego typu stanowią interfejs, dzięki któremu w łatwy sposób można uzyskać dostęp do informacji publikowanych przez różnych wydawców. Jako przykład tego typu agregatu podać można czytniki RSS. Są to mobilne aplikacje, dzięki którym użytkownik ma możliwość przeglądania artykułów zamieszczonych na różnych portalach informacyjnych. Praca ta stanowi teoretyczne i praktyczne rozważania nad skonstruowaniem analogicznego systemu dla danych multimedialnych: muzyki, filmów i seriali, jako ich źródło przyjmując serwisy strumieniowe.

1.2. Cel pracy

Celem pracy jest analiza możliwości ekstrakcji danych na temat dostępnych albumów, playlist, filmów i seriali z popularnych serwisów strumieniowych oraz opracowanie systemu pozwalającego na automatyzację tego procesu. W jej ramach przeanalizowano 3 serwisy muzyczne: Spotify, YouTube Music i Deezer oraz 2 platformy z filmami i serialami: Netflix i Amazon Prime.

1.3. Rozwiązanie przyjęte w pracy

Część analizowanych serwisów strumieniowych udostępnia publiczny interfejs programistyczny API. Na potrzeby pracy wykorzystano następujące z nich:

1. Spotify Web API [4]
2. YouTube Data API [5]
3. Deezer API [6]

Ponadto, na potrzeby katalogowania filmów i seriali, skorzystano z publicznej internetowej bazy danych IMDb. Informacji na temat dostępności multimediów w serwisie Netflix oparto o usługę uNogs, dostępną za pośrednictwem interfejsu API. Platforma Amazon Prime nie posiada dedykowanego interfejsu API. Do ekstrakcji danych dostępnych w jej ramach wykorzystano metodę web scraping, korzystając z biblioteki puppeteer dla języka JavaScript.

Prototyp aplikacji powstałej w ramach pracy posiada strukturę modułową, składającą się z następujących komponentów:

1. **Aplikacja *front-end*** – część oprogramowania wykonywana po stronie klienta, w przeglądarce internetowej. Napisana w języku JavaScript z wykorzystaniem biblioteki React.
2. **Aplikacja *back-end*** – komponent działający po stronie serwera stworzony w języku TypeScript oraz przy użyciu frameworku Express.js. Przy projektowaniu tej części zdecydowano się na implementację wzorca architektury heksagonalnej. W jej ramach wydzielono następujące moduły:
    a. **User** – odpowiedzialny za rejestrację i logowanie użytkownika oraz pobieranie i zapisywanie danych z nim związanych
    b. **Music** – zapewniający integrację z muzycznymi serwisami strumieniowymi.
    c. **Movie** – pozwalający na ekstrakcję danych na temat filmów z serwisów strumieniowych.
    d. **Tv** – odpowiedzialny za agregację danych z serwisów strumieniowych dotyczących seriali
3. **Baza danych MongoDB** – nierelacyjna baza danych przechowująca informacje na temat użytkowników oraz część treści pozyskanych z serwisów strumieniowych.

1.4. Rezultat pracy

Rezultatem pracy jest opracowany zestaw różnych strategii, służących do automatyzacji procesu pozyskiwania informacji na temat multimediów dostępnych w serwisach strumieniowych. W celu potwierdzenia skuteczności tych metod przygotowano prototyp aplikacji agregującej dane pochodzące z usług Spotify, YouTube Music, Deezer, Netflix i Amazon Prime.

1.5. Organizacja pracy

Praca podzielona została na 7 rozdziałów. Pierwszy z nich nakreśla najważniejsze problemy związane z agregacją danych z serwisów strumieniowych. Przedstawiono w nim definicje związane z omawianą tematyką, wyzwania oraz przeanalizowano istniejące na rynku rozwiązania w tym zakresie.

Kolejne rozdziały poświęcone zostały opisowi rozwiązania w postaci aplikacji webowej, które powstało jako integralna część pracy. W pierwszym z nich opisano wizję oraz ogólne założenia systemu. Zestawiono w nim również wymagania, jakie stawiane są przed projektowanym system, zarówno funkcjonalne jak i niefunkcjonalne. Następna część w sposób ogólny opisuje kluczowe technologie, języki programowania, biblioteki oraz narzędzia wykorzystane do przygotowania prototypu. Piąty rozdział skupia się na interfejsie użytkownika, analizując dostępne w jego ramach widoki. Kolejna część poświęcona został implementacji platformy. W jej ramach dokonano charakterystyki poszczególnych komponentów wchodzących w skład systemu. Na jej stronach znalazł się również opis różnych strategii zastosowanych do ekstrakcji danych z serwisów strumieniowych: od wykorzystania interfejsów API, po metodę web scraping. Na koniec tej części zestawiono pomysły na rozwój systemu. Ostatnim rozdział stanowi podsumowanie działań wykonanych w ramach pracy oraz próbę wysnucia wniosków na temat pozyskiwania danych z serwisów strumieniowych.

2. Agregacja danych z serwisów strumieniowych

Rozdział ten stanowi wstęp do rozważań nad problematyką agregacji danych z serwisów strumieniowych. W jego początkowej części przybliżone zostały definicje pojęć, których znajomość jest niezbędna do zrozumienia tematyki pracy. Dalsze podrozdziały poświęcone zostały nakreśleniu wyzwań jakie niesie ze sobą tematyka agregacji danych z serwisów strumieniowych. Na koniec przedstawiono kilka istniejących rozwiązań, które mierzą się z tą problematyką wraz z analizą ich mocnych i słabych stron.

2.1. Definicje kluczowych pojęć

Ze względu na dużą niejednoznaczność wykorzystywanych w pracy pojęć zdecydowano się przedstawić definicje najważniejszych z nich.

Definicja 1

Treści mulimedialne to pojęcie, pod którym rozumiane są dane dostępne w Internecie w postaci utworów audio, filmów, seriali oraz wszelkie struktury służące do ich katalogowania, takie jak albumy, playlisty czy sezony.

Definicja 2

W celu wyjaśnienia pojęcia **serwisu strumieniowego** przytoczono definicję pochodzącą z [7]:

Serwis, który przesyła dane takie jak wideo, audio poprzez Internet do osób, które odtwarzają je w czasie rzeczywistym. Treści te nie muszą być wcześniej pobrane. Dostęp do nich uzyskać można w dowolnym momencie, a nie wyłącznie w czasie ich emisji.[1]

Na potrzeby pracy przyjęto podział serwisów strumieniowych ze względu na typ udostępnianych treści na **muzyczne** (Spotify, YouTube Music, Deezer) oraz **wideo** (Netflix, Amazon Prime).

Definicja 3

API (ang. application programming interface) to sposób komunikacji między 2 aplikacjami komputerowymi poprzez sieć (zazwyczaj web). Jej zasady są jasno zdefiniowane i znane przez obie ze stron. Definicję opracowano w oparciu o [8].

Definicja 4

Definicję **web scrapingu** zaczerpnięto z pozycji [32]:

*W teorii, **web scraping** to metoda gromadzenia danych w sposób inny niż przy pomocy interakcji z API (lub, oczywiście, inny niż poprzez bezpośrednie wykorzystanie przeglądarki sieciowej przez człowieka). Zazwyczaj na jego potrzeby wykorzystuje się zautomatyzowane programy odpytujące serwer. W odpowiedzi otrzymują one dane (w formie HTML i innych plików znajdujących się na stronie), które następnie przetwarzają i wydobywają z nich potrzebne informacje.* [2]

[1] tłum. autora
[2] tłum. autora

2.2. Wyzwania

Gromadzenie danych pochodnych z wielu serwisów strumieniowych niesie ze sobą szereg wyzwań. Na wstępie warto zaznaczyć, że nie istnieje żaden standard specyfikujący w jakiej formie powinny być udostępniane tego typu dane. Dotyczy to zarówno muzycznych, jak i wideo. Każda platforma udostępnia swój własny, niezależny interfejs, a co za tym idzie inne dane. W związku z tym wymagane jest indywidualne podejście do każdej z usług, dostosowane do formatu i rodzaju treści jakie udostępnia.

Innym wyzwaniem, występującym w ramach analizowanego procesu jest mapowanie multimediów pochodzących z różnych platform. Rozpatrzmy sytuację, w której pewien album muzyczny dostępny jest w kilku serwisach strumieniowych. System agregujący powinien być zdolny do rozpoznania, iż jest to to samo dzieło i dokonać złączenia danych pochodzących z wielu źródeł. Analogiczny problem występuje również w przypadku filmów i seriali.

2.3. Poziomy dostępu danych w serwisach strumieniowych

Osobnym zagadnieniem wymagającym omówienia są poziomy dostępności danych na różnych platformach strumieniowych. Jest to czynnik o tyle kluczowy, iż warunkuje on zestaw narzędzi jakie wykorzystane będą do ich pozyskania. Na potrzeby niniejszej pracy wydzielono 3 poziomy dostępu danych dla serwisów strumieniowych:

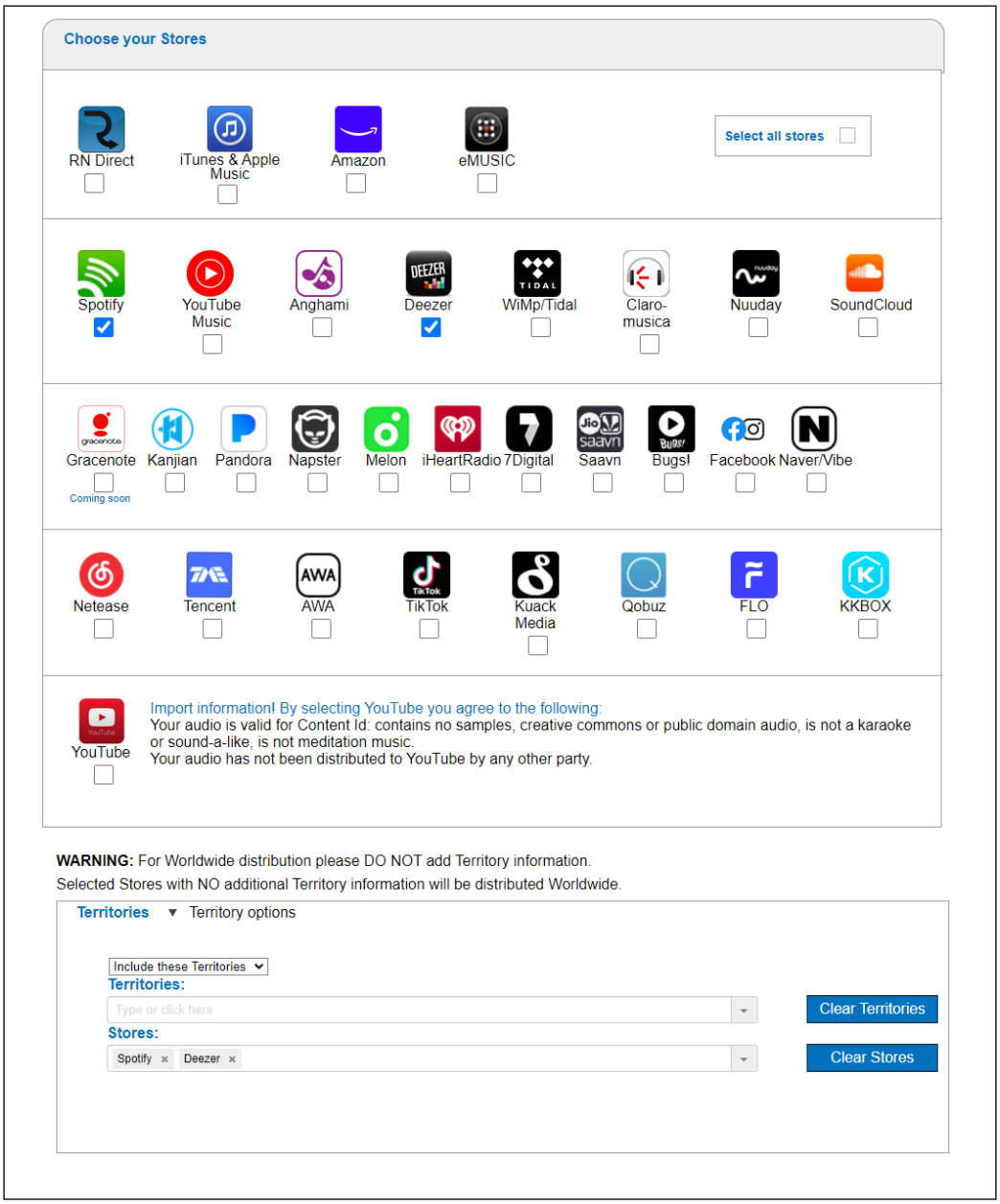
1. **Dane publiczne, dostępne za pomocą API** – w celu pozyskania danych wystarczy wysłać odpowiednie zapytanie HTTP. Dotyczy: Spotify, Deezer.
2. **Dane publiczne, bez dostępu za pomocą API –** w celu pozyskania danych potrzebne jest wykorzystanie prostych metod web srapingu. Dotyczy: YouTube Music.
3. **Dane prywatne –** w celu pozyskania danych wymagane jest uwierzytelnienie użytkownika. Wymagane jest użycie zaawansowanych metod web scrapingu, bądź rozwiązań oferowanych przez usługi pośredniczące. Dotyczy: Netflix, Amazon Prime.

2.4. Przegląd istniejących rozwiązań

Na rynku dostępnych jest wiele rozwiązań, które mierzą się z problematyką agregacji treści z serwisów strumieniowych. Żadne z nich nie posiada jednak na tyle rozbudowanych funkcjonalności, aby określić je mianem kompleksowego. Niniejszy podrozdział stanowi przegląd kilku najbardziej złożonych serwisów oferujących dostęp do danych z platform strumieniowych oraz analizę ich mocnych i słabych stron.

2.4.1 RouteNote

RouteNote to serwis internetowy pozwalający twórcom muzyki udostępniać oraz monitorować statystyki swoich utworów na różnych platformach strumieniowych i sklepach cyfrowych. Jest to rozwiązanie zaawansowane, zapewniające integrację z większością popularnych usług muzycznych, w tym m.in. Spotify, YouTube Music, Deezer, Tidal czy Apple Music. System zorientowany jest na artystę, a co za tym idzie nie umożliwia przeglądania informacji na temat albumów zamieszczanych przez innych twórców. Widok z aplikacji RouteNote zamieszczono na rysunku 1.



Rysunek 1. Okno platformy RouteNote. Źródło: [9]

Zalety:

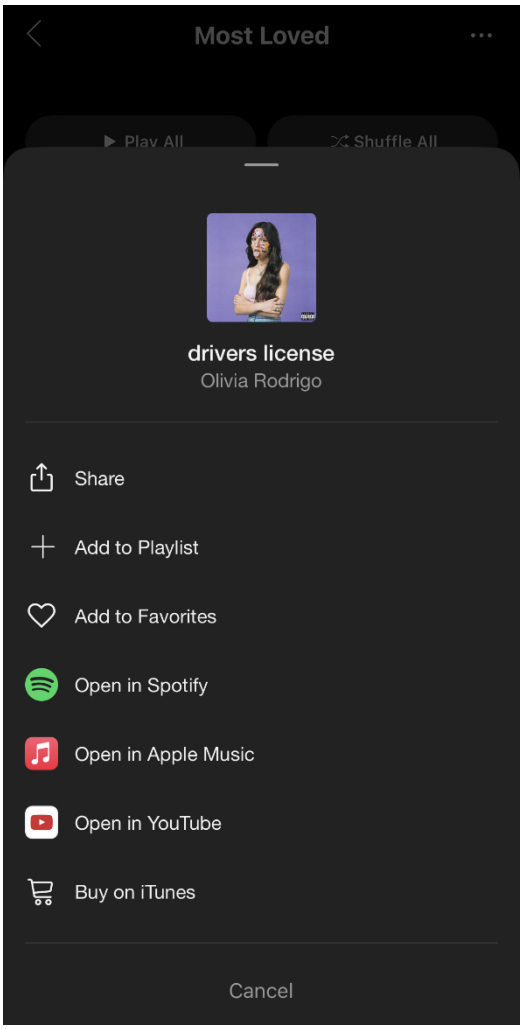
- wsparcie dla wielu serwisów muzycznych.
- możliwość monitorowania statystyk.
- łatwa w użytkowaniu.

Wady:

- brak możliwości przeglądania albumów dodanych przez innych twórców.

2.4.2 SoundHound

Podstawową funkcjonalnością aplikacji mobilnej SoundHound jest wyszukiwanie utworów na podstawie dźwięków rejestrowanych za pomocą mikrofonu smartfonu. Jako funkcję dodatkową, twórcy przewidzieli funkcję odtworzenia znalezionej piosenki w serwisach strumieniowych Spotify, Apple Music lub na platformie YouTube. Opcja ta została jednak zaprojektowana w taki sposób, że uzyskanie do niej dostępu nie jest intuicyjne. SoundHound dostępny jest jedynie w formie aplikacji mobilnej przeznaczonej na system iOS i Android. Przykładowe okno z poglądem dostępności album na serwisach strumieniowych widoczne jest na rysunku 2.



Rysunek 2. Okno aplikacji mobilnej SoundHound. Źródło: opracowanie własne.

Zalety:

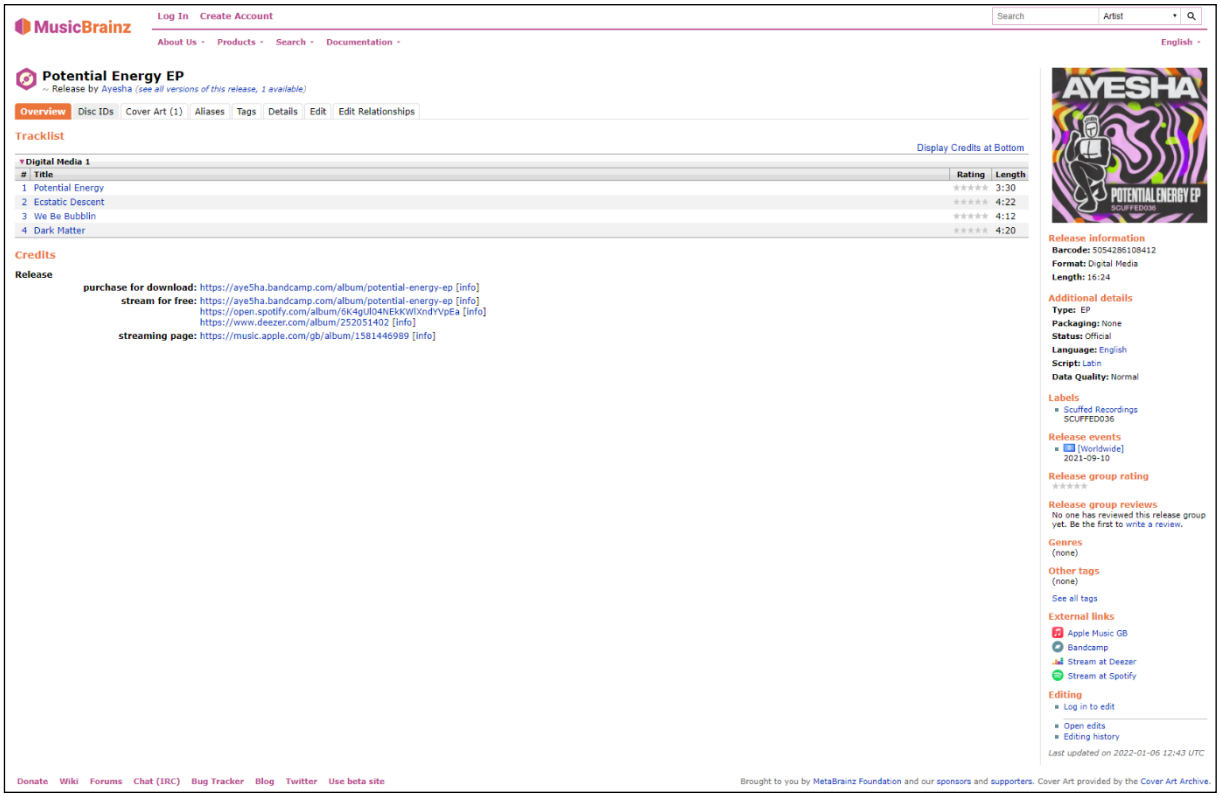
- możliwość wyszukiwania utworów za pomocą dźwięku.

Wady:

- dostęp wyłącznie za pośrednictwem natywnej aplikacji mobilnej.
- integracja z niewielką ilością serwisów strumieniowych.
- znalezienie opcji odtworzenia utworu w zewnętrznych serwisach jest nieintuicyjne.

2.4.3 MusicBrainz

MusicBrainz to internetowa baza danych zbierająca informacje na temat wydanych album muzycznych. Serwis działa na zasadzie wiki – treści znajdujące się na nim są współtworzone i edytowane przez społeczność zaangażowanych użytkowników. Choć nie należy to do podstawowej funkcjonalności serwisu, za jego pomocą można sprawdzić dostępność utworów na różnych serwisach strumieniowych. Przykładowy widok pochodzący z serwisu MusicBrainz zamieszczono na rysunku 3.



Rysunek 3. Serwis internetowy MusicBrainz. Źródło: [10]

Zalety:

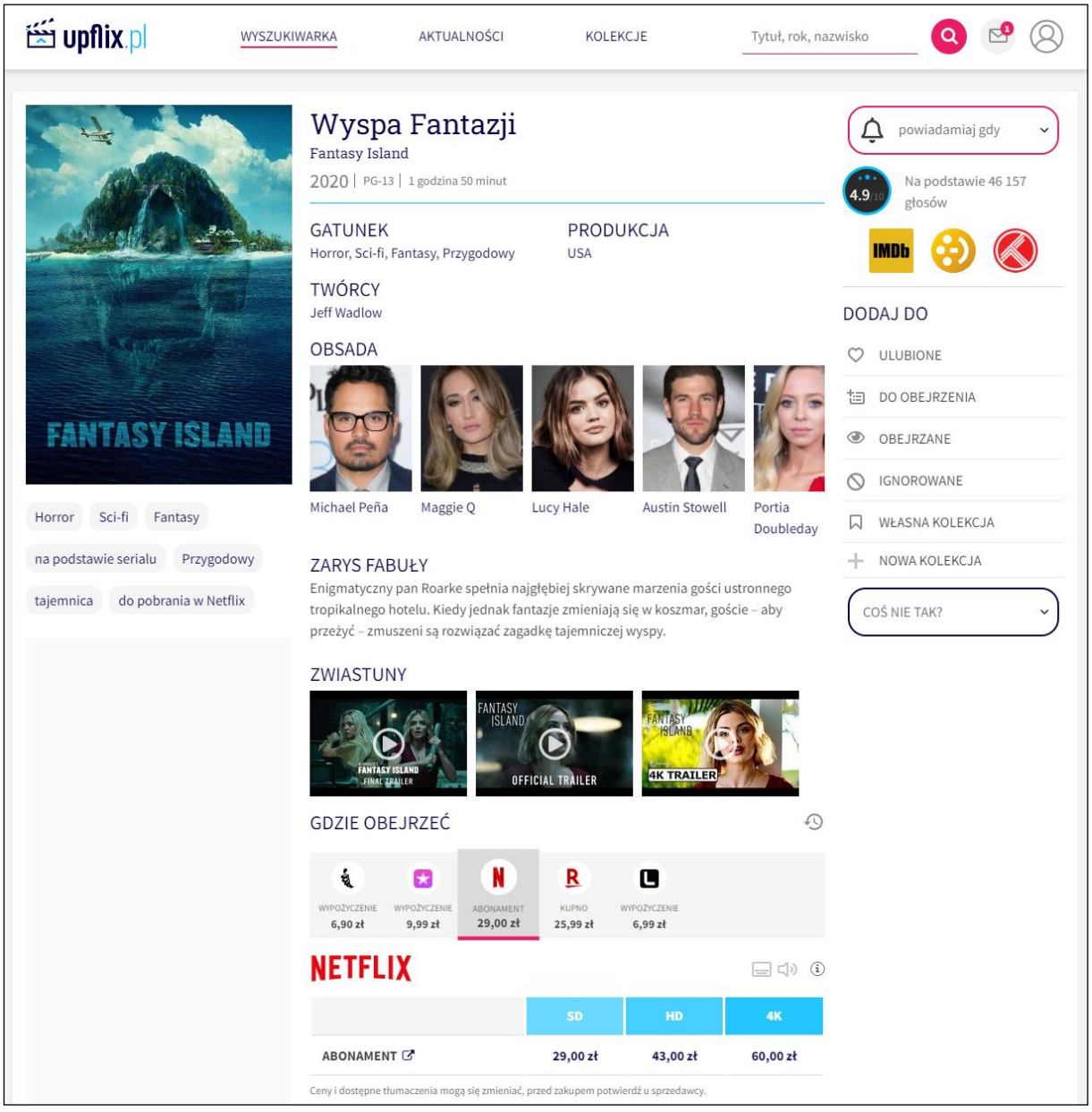
- w bazie znajduje się wiele albumów, w tym również takie, które nie ukazały się nigdy w formie cyfrowej.
- wsparcie dla wielu serwisów strumieniowych.
- możliwość pobrania migawki bazy danych, która może być wykorzystana przez zewnętrzne aplikacje.

Wady:

- baza tworzona jest przez społeczność, nie zaś w sposób automatyczny. W związku z tym może brakować w niej informacji na temat niektórych albumów, a część danych może być nieaktualna.
- brak interfejsu programistycznego API.

2.4.4 Upflix

Upflix to popularny w Polsce serwis internetowy, za pomocą którego można sprawdzić dostępność filmów oraz seriali w ramach serwisów strumieniowych oraz platform VOD. To złożone rozwiązanie, oferujące szereg funkcjonalności. Wśród nich warto wymienić zaawansowane wyszukiwanie tytułów, możliwość tworzenia własnych list filmów oraz ustawiania alarmów, wywołujących się, gdy dany tytuł zostanie dodany lub usunięty z jednej z platform. Widok przedstawiający dostępność filmu na platformach strumieniowych w aplikacji Upflix widoczny jest na rysunku 4.



Rysunek 4. Serwis internetowy Upflix. Źródło: [11]

Zalety:

- wsparcie dla wielu serwisów strumieniowych i platform VOD dostępnych w Polsce.

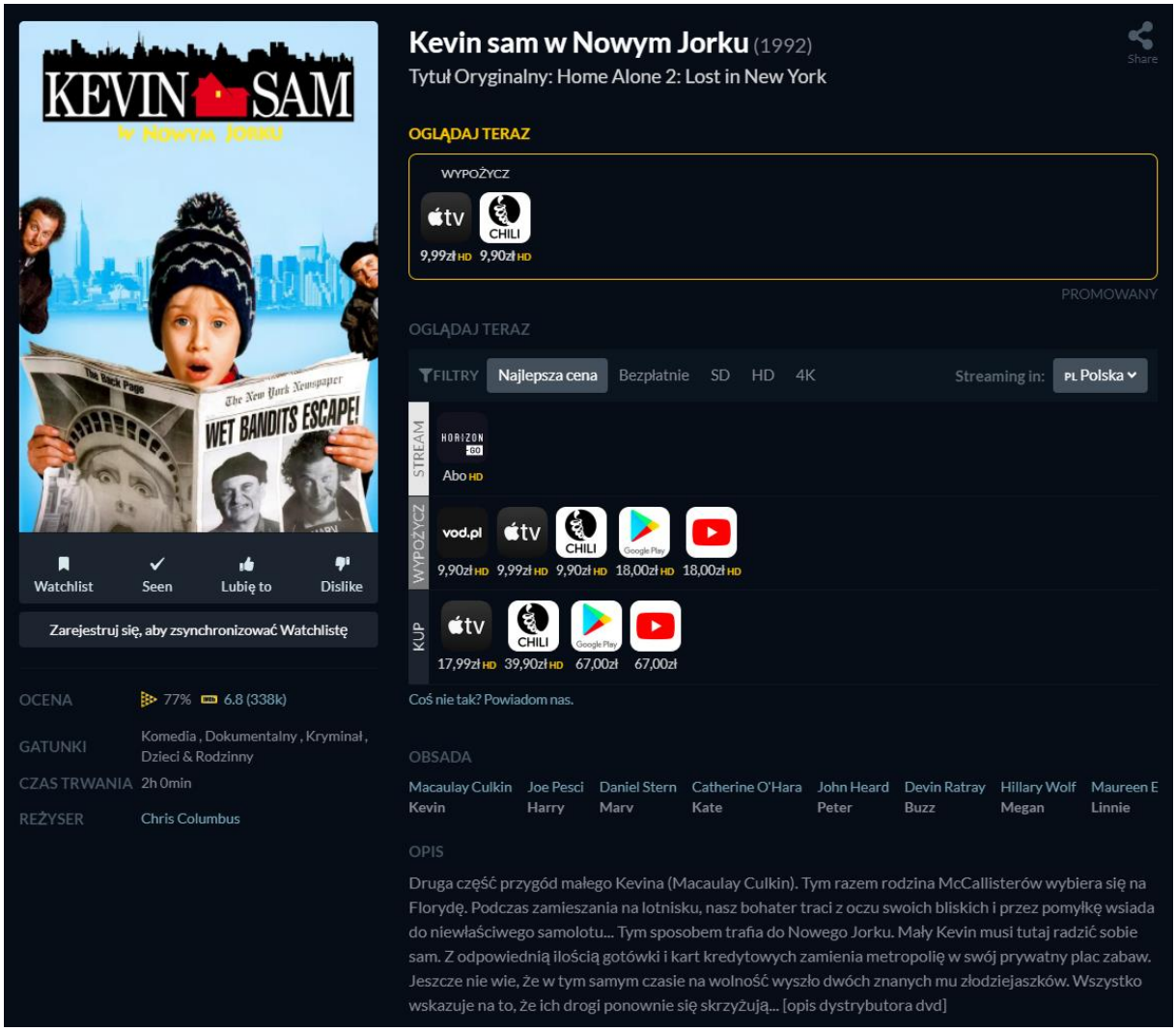
- zaawansowane opcje wyszukiwania tytułów.
- możliwość tworzenia własnych list filmów przez użytkownika.
- funkcja ustawienia alarmów.
- porównywarka cen filmu na różnych platformach.

Wady:

- serwis agreguje dane na temat dostępności tylko na terenie Polski.
- brak wsparcia dla niektórych mniej popularnych serwisów (np. Mubi, Nowe Horyzonty VOD)
- brak interfejsu programistycznego API.
- w bazie danych nie ma filmów, które nie są w danym momencie dostępne na żadnej platformie.

2.4.5 JustWatch

JustWatch to jeden z bardziej rozwiniętych przewodników oferty serwisów strumieniujących wideo. Za jego pomocą można uzyskać informację na temat dostępności tytułów na różnych platformach w wielu krajach na świecie, w tym w Polsce. Posiada wiele funkcjonalności pozwalających na sprawne przeszukiwanie bazy filmów i seriali oraz opcje personalizacji – np. tworzenie listy filmów do obejrzenia przez użytkownika. Przykładowe okno serwisu JustWatch zamieszczono na rysunku 5.



Rysunek 5. Serwis internetowy JustWatch. Źródło: [12]

Zalety:

- wsparcie dla wielu serwisów strumieniowych i platform VOD.
- usługa dostępna dla terytoriów wielu krajów, w tym w Polsce.
- zaawansowane opcje wyszukiwania tytułów.

Wady:

- brak wsparcia dla niektórych mniej popularnych serwisów (np. Mubi, Nowe Horyzonty VOD).
- brak interfejsu programistycznego API.

3. Propozycja rozwiązania

Niniejszy rozdział w sposób szczegółowy opisuje założenia jakimi kierowano się przy przygotowaniu prototypu rozwiązania. W jego pierwszej części przybliżono najważniejsze cechy, jakimi powinna charakteryzować się platforma do agregacji danych multimedialnych z serwisów strumieniowych. Następnie dokonano analizy potrzeb użytkownika systemu, w ramach której powstało zestawienie wymagań funkcjonalnych. W dalszej części znaleźć można rozważania nad różnego typu standardami, które powinna spełniać projektowana aplikacja. Na koniec przedstawiono koncepcję architektury całego systemu wraz z opisem jego komponentów składowych.

3.1. Wizja systemu

Na podstawie analizy istniejących rozwiązań oraz przemyśleń własnych autora powstała wizja systemu, który adresowałby problem dostępu do danych pochodzących z różnych serwisów strumieniowych. Poniżej zestawiono kilka najważniejszych zasad i koncepcji, którymi kierowano się w czasie projektowania aplikacji:

- **Uniwersalność** – aplikacja powinna być otwarta na różne rodzaje multimediów. W ramach analizy istniejących rozwiązań nie udało się znaleźć platformy, która podejmowałaby próbę agregacji danych pochodzących zarówno z muzycznych serwisów strumieniowych jak i wideo. Opracowywany system powinien agregować multimedia różnego typu: muzykę, filmy oraz seriale.
- **Automatyzacja** – wszelkie informacje z serwisów strumieniowych powinny być pozyskiwane w sposób całkowicie zautomatyzowany, za pomocą interfejsu API lub web scrapingu. Nie dopuszcza się sytuacji, w której część informacji uzyskiwana uzyskana jest w sposób ręczny, poprzez interakcję człowieka z przeglądarką.
- **Rozszerzalność** – aplikacja powinna mieć możliwość łatwego rozszerzania o moduły obsługujące inne serwisy strumieniowe.
- **Jakość** – system powinien charakteryzować się niskimi kosztami utrzymania. Wykorzystana architektura powinna stanowić solidną podstawę umożliwiającą łatwe wdrażanie nowych funkcjonalności. Oprogramowanie powinno spełniać nowoczesne standardy oraz być napisane w myśl dobrych zasad programowania.
- **Przejrzystość** - aplikacja powinna być napisana w zgodzie z obowiązującymi standardami designu stron internetowych oraz projektowania UX. W tym celu oparto się o zasady zebrane w ramach norm Material Design [13] [14].

3.2. Wymagania funkcjonalne

W niniejszemu podrozdziale dokonano dekompozycji projektowanego systemu, w ramach której powstała mapa wymagań funkcjonalnych. Zostały one zestawione w formie tzw. historii użytkownika, zgodnie z założeniami metodyk zwinnych wytwarzania oprogramowania [15]. Wyniki zestawiono w tabeli 1.

Tabela 1. Wymagania funkcjonalne projektowanego systemu

Nr	Historia
1	Jako Użytkownik chcę przeglądać nowododane w muzycznych serwisach strumieniowych albumy, aby być na bieżąco z nowinkami muzycznymi.
2	Jako Użytkownik chcę móc znaleźć konkretny album muzyczny, aby uzyskać informacje na jego temat.
3	Jako Użytkownik chcę wiedzieć na jakich serwisach strumieniowych dostępny jest dany album oraz uzyskać do niego odpowiednie hiperłącze, aby go odsłuchać.
4	Jako Zalogowany Użytkownik chcę zarejestrować swoje konta w serwisach Spotify, YouTube i Deezer, aby móc przeglądać wszystkie zapisane przeze mnie playlisty muzyczne w tych serwisach.
5	Jako Użytkownik chcę przeglądać popularne playlisty dostępne na muzycznych serwisach strumieniowych, aby orientować się co jest obecnie na czasie.
6	Jako Użytkownik chcę móc znaleźć konkretną playlistę muzyczną, aby uzyskać informację na jej temat.
7	Jako Użytkownik chcę wiedzieć na jakim serwisie strumieniowym dostępna jest dana playlista uzyskać do niej odpowiednie hiperłącze, aby ją odsłuchać.
8	Jako Użytkownik chcę przeglądać nowododane w serwisach strumieniowych filmy, aby być na bieżąco z nowinkami filmowymi.
9	Jako Użytkownik chcę przeglądać popularne filmy, aby orientować się co jest obecnie na czasie.
10	Jako Użytkownik chcę móc znaleźć konkretny filmy, aby uzyskać informację na jego temat.
11	Jako Użytkownik chcę wiedzieć na jakich serwisach strumieniowym dostępny jest dany film oraz uzyskać do niego odpowiednie hiperłącze, aby go obejrzeć.
12	Jako Użytkownik chcę przeglądać nowododane w serwisach strumieniowych seriale, aby być na bieżąco z nowinkami serialowymi.
13	Jako Użytkownik chcę przeglądać popularne seriale, aby orientować się co jest obecnie na czasie.
14	Jako Użytkownik chcę móc znaleźć konkretny serial, aby uzyskać informację na jego temat.
15	Jako Użytkownik chcę wiedzieć na jakich serwisach strumieniowym dostępny jest dany serial oraz uzyskać do niego odpowiednie hiperłącze, aby go obejrzeć.

Warto zwrócić uwagę na fakt, że w ramach prototypu nie przewidziano możliwości przeglądania wszystkich dostępnych w serwisach strumieniowych albumów, playlist, filmów i seriali. Decyzja ta zostało podjęta świadomie. Są to niezwykle liczne zbiory danych, których eksploracja nie wydaje się nieść ze sobą żadnej wartości z punktu widzenia użytkownika. Niemniej, informacje na temat każdego medium powinna być dostępna w serwisie, a dostęp do niej można uzyskać np. z poziomu wyszukiwarki.

3.3. Pozostałe wymagania

Aplikacja powinna również charakteryzować się następującymi cechami.

3.3.1 Szybkość działania

Projektowana aplikacja wykorzystuje zewnętrzne platformy, z którymi komunikuje się poprzez sieć web. Czas wymagany na uzyskanie danych może się różnić w zależności od serwisu strumieniowego, z którego są odczytywane. Przy dużych zapytaniach może okazać się, że czas potrzebny na jego wykonanie jest na tyle długi, że użytkownik może odnieść wrażenie, że system nie działa prawidłowo. W założeniu sytuacje takie powinny występować możliwie rzadko. W związku z tym podjęto decyzję o zapisywaniu (tzw. cachowaniu) części informacji uzyskanych z serwisów strumieniowych w bazie danych.

3.3.2 Modularność

Projektowana aplikacja powinna charakteryzować się dużą niezależnością poszczególnych komponentów. Moduły odpowiedzialne za obsługę komunikacji z poszczególnymi serwisami strumieniowymi powinny charakteryzować się cechami wtyczki. Pożądane jest, aby każdy z nich zawierał w sobie wszystkie zależności potrzebne do jego prawidłowego działania. System powinien przewidywać prosty mechanizm rejestrowania i wyrejestrowywania poszczególnych modułów. Ponadto, w ramach projektu opracowane powinny zostać interfejsy obiektowe dla komponentów odpowiedzialnych za obsługę konkretnych typów multimediów: muzyki, filmów i seriali.

3.3.3 Skalowalność

Serwerowa część opisywanego rozwiązania projektowana jest z myślą o działaniu w chmurze. Dobre praktyki wytwarzania tego typu oprogramowania wymagają, aby system taki zapewniał możliwość skalowania horyzontalnego [16]. W związku z tym aplikacja *back-end* nie powinna przechowywać danych trwałych z punktu widzenia systemu w pamięci operacyjnej. Od baza danych wymaga się, aby była przystosowana do obsługi równoległych operacji.

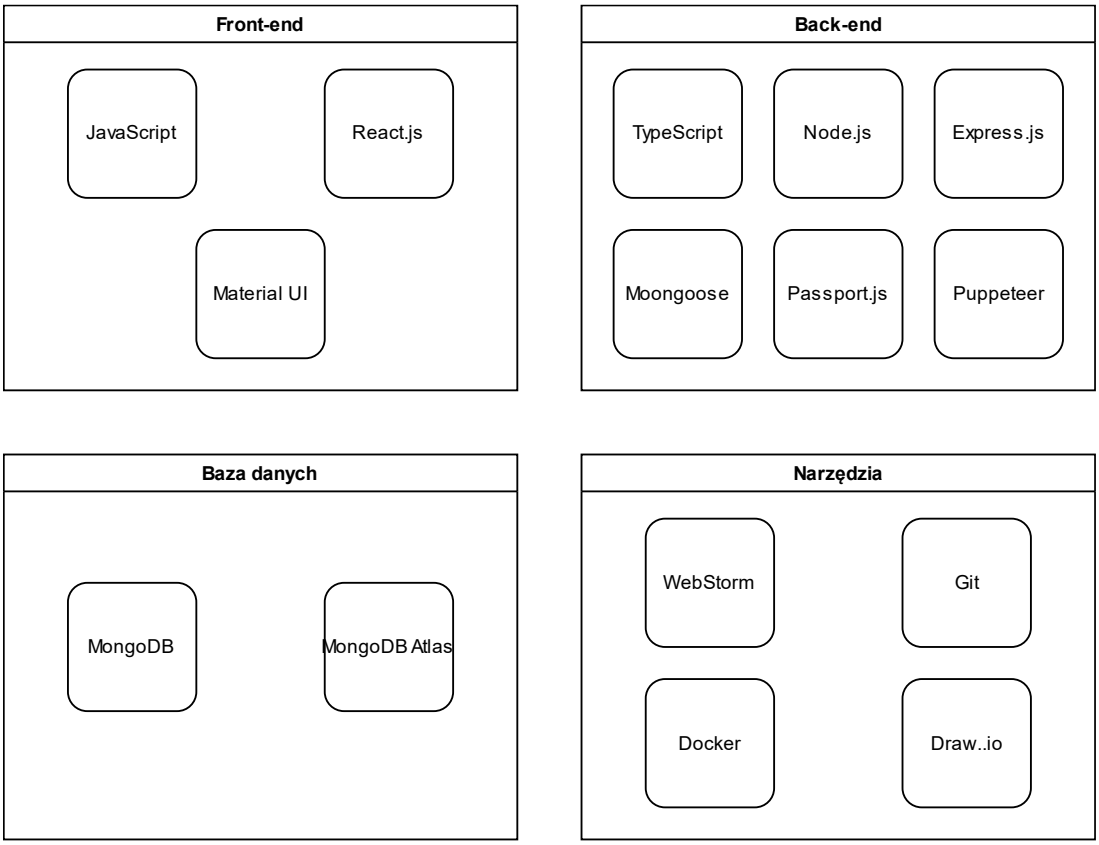
3.3.4 Generyczność

System powinien być możliwie uniwersalny i elastyczny. W ramach jego architektury przewidziano obsługę różnego rodzaju multimediów pochodzących z wielu serwisów strumieniowych. Pożądanym jest również, aby aplikacja umożliwiała łatwe rozszerzania o inne typy mediów oraz platformy strumieniowe.

4. Opis użytych technologii

W tym rozdziale znaleźć można opis najważniejszych technologii i narzędzi użytych do przygotowania prototypu aplikacji. Jego pierwsza część stawia sobie za cel pogrupowanie wykorzystanych rozwiązań w bloki tematyczne skupione wokół poszczególnych komponentów. Przybliża ona również podejście MERN, w którego duchu powstał omawiany system. W dalszych podrozdziałach znaleźć można charakterystykę części technologii zastosowanych w ramach niniejszej pracy, w tym języków programowania, frameworków oraz bibliotek.

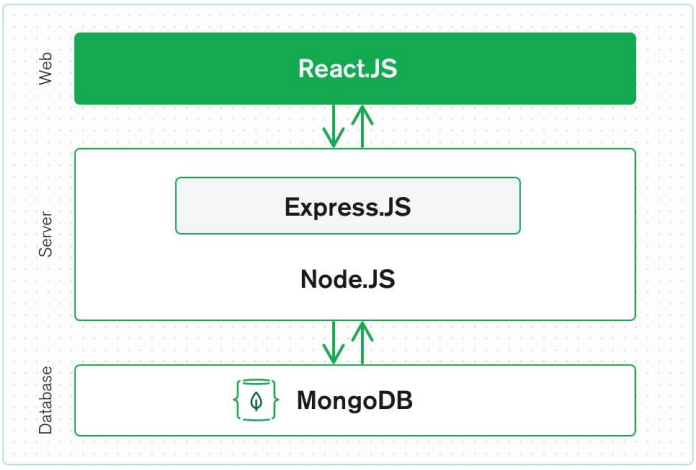
4.1. Mapa technologii



Rysunek 6**.** Mapa technologii użytych w ramach systemu

Na rysunku 6. przedstawione zostały technologie użyty w poszczególnych warstwach systemu. Zdecydowano się na wykorzystanie tzw. stosu technologicznego MERN (akronim pochodzący od pierwszych liter: MongoDB, Express.js, React.js, Node.js) [17], którego schemat zobaczyć można na rysunku 7. Podejście to zorientowane jest na wykorzystaniu języka JavaScript w ramach wszystkich poziomów aplikacji. Rozwiązanie to ma na celu redukcję czasu oraz liczby osób potrzebnych do rozwoju i utrzymania sytemu. Programista specjalizujący się w JavaScript może pracować zarówno nad częścią kliencką, jak i serwerową, bez potrzeby zmiany języka programowania. MERN charakteryzuje się również wysoką wydajnością systemu, co wynika z dobrej optymalizacji platformy Node.js. Firma TechEmpower przeprowadza cyklicznie testy obciążeniowe wielu rozwiązań stworzonych z myślą o

oprogramowaniu webowym [18]. Node.js wypada w nich zdecydowanie lepiej niż większość popularnych frameworków, wyprzedzając m.in. Spring, Ruby On Rails czy Django. Warto zaznaczyć, że podejście MERN nie sprawdza się przy aplikacjach, które wymagają wykonywania dużej ilości wymagających obliczeń, ze względu na brak wsparcia dla wielowątkowości w platformie Node.js.



Rysunek 7. Schemat ekosystemu MERN. Źródło: [17]

4.2. JavaScript

JavaScript to język programowania stworzony w połowie lat 90 w ramach kooperacji pomiędzy firmami Netscape Communications Corporation (twórcami najpopularniejszej w tamtych czasach przeglądarki internetowej Netscape) oraz Sun Microsystems (odpowiedzialnej za powstanie języka Java) [19]. Platforma ta od samego początku projektowana była z myślą o wytwarzaniu oprogramowania webowego. Jest to język skryptowy, charakteryzujący się dynamicznym typowaniem ze słabą kontrolą. Każda ze współcześnie dostępnych przeglądarek internetowych posiada możliwość interpretacji programów napisanych w JavaScript, co przyczyniło się do jego ogromnej popularności. Dokumentem standaryzującym język jest ECMAScript, a jego najnowsza wersja pochodzi z 2021. Współczesny JavaScript umożliwia wytwarzanie oprogramowania zgodnie z najpopularniejszymi paradygmatami programowania, m.in. obiektowym, funkcyjnym, imperatywnym.

4.3. React.js

Jest to biblioteka języka JavaScript powstała z myślą o tworzeniu interfejsów użytkownika na potrzeby witryn internetowych. React zapewnia wysokopoziomową warstwę abstrakcji, która pozwala opracować zaawansowane GUI (z ang. *graphical user interface*), o dynamiczny charakterze. Biblioteka ta zachęca programistę do stosowania architektury modularnej, dzięki czemu fragmenty kodu napisane z jej użyciem mogą być zazwyczaj wykorzystany ponownie. Za jej stworzenie i rozwój odpowiada Facebook. Jak przekonuje autor [20], najważniejszymi powodami, dla których warto skorzystać z React.js są:

- Możliwość bezpośredniego modyfikowania drzewa DOM (z ang. *Document Object Model*), a co za tym idzie interfejsu użytkownika.

- Możliwość rozszerzania za pomocą dowolnych, zewnętrznych bibliotek
- Popularność i zaangażowana społeczność.

Biblioteka dostępna jest w ramach licencji MIT.

Listing 1. Przykładowy komponent biblioteki React.js

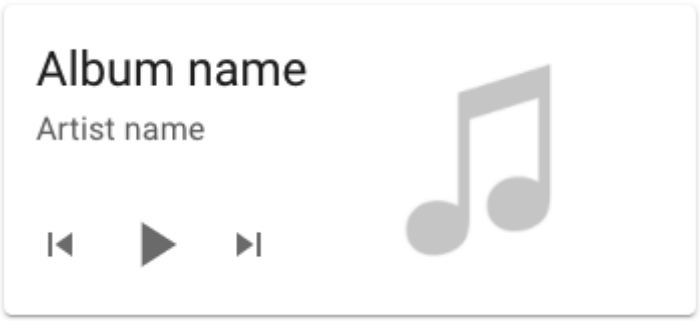
```
export function ExampleComponent() {
  return (
    <p> Hello World! </p>
  );
}
```

4.4. Material UI

Material UI to biblioteka komponentów React i stylów CSS. Służy do projektowania interfejsów graficznych w duchu zasad Material Design [14]. Jej niewątpliwą zaletą jest obszerna i dobrze zorganizowana dokumentacja. Komponenty dostarczane w ramach tej biblioteki charakteryzują się minimalistycznym wyglądem oraz dużą czytelnością. Licencja Material UI zezwala na nieograniczone użytkowanie w celach niekomercyjnych. Implementacja przykładowego komponentu napisanego przy użyciu biblioteki została zamieszczona na listingu 2, a jego widok na rysunku 8.

Listing 2. Przykład wykorzystania komponentów biblioteki Material UI

```
<Card sx={{ display: 'flex', maxWidth: 345, m: 4}}>
  <Box sx={{ display: 'flex', flexDirection: 'column' }}>
    <CardContent sx={{ flex: '1 0 auto' }}>
      <Typography component="div" variant="h5">
        Album name
      </Typography>
      <Typography variant="subtitle1" color="text.secondary" component="div">
        Artist name
      </Typography>
    </CardContent>
    <Box sx={{ display: 'flex', alignItems: 'center', pl: 1, pb: 1 }}>
      <IconButton aria-label="previous">
         <SkipPreviousIcon />
      </IconButton>
      <IconButton aria-label="play/pause">
        <PlayArrowIcon sx={{ height: 38, width: 38 }} />
      </IconButton>
      <IconButton aria-label="next">
        <SkipNextIcon />
      </IconButton>
    </Box>
  </Box>
  <CardMedia
    component="img"
    sx={{ width: 151 }}
    image="cover.jpg"
    alt="cover"
  />
</Card>
```



Rysunek 8. Widok elementu z listingu 2

4.5. TypeScript

TypeScript to stworzony w 2012 roku przez Microsoft dialekt języka JavaScript, który rozszerza go o koncept typowania statycznego. Zmiana ta pozwala na projektowanie aplikacji bardziej odpornych na błędy. Kod napisany w TypeScript kompilowany jest do JavaScript, a następnie wykonywany. Kontrola typów wykonywana jest na etapie kompilacji, dzięki czemu wszelkie nieprawidłowości mogą zostać wykryte przed uruchomieniem programu. Warto zauważyć, że TypeScript jest całkowicie kompatybilny ze swoim prekursorem i nie ma przeciwskazań, aby w jego ramach wykonać kod napisany w czystym JavaScript, co zapewnia mu dostęp do wielu sprawdzonych bibliotek.

Listing 3. Porównanie języków JavaScript i TypeScript

```
function hello(name)
{
    return "Hello " + name;
}

console.log(hello(1));
```

```
function hello(name: string)
{
    return "Hello " + name;
}

console.log(hello(1));
```

Listing 3 zestawia ze sobą tę samą funkcję napisaną w języku JavaScript i TypeScript. W pierwszym wypadku kod wykona się i wyświetli w terminalu konsoli napis `"Hello 1"`. Kod po prawej nie skompiluje się, gdyż następuje w nim niejawna konwersja typu `number` na `string`.

4.6. Node.js

JavaScript powstał w myślą o działaniu w przeglądarce internetowej. Koncepcja ta uległa zmianie w 2009 roku wraz z publikacją platformy Node.js. Jest to środowisko programistyczne przeznaczone dla serwerów webowych, umożliwiające wykonywanie oprogramowania napisanego w JavaScript, udostępnione na zasadach open-source. Jako jego podstawa wykorzystany został silnik V8, stworzony przez firmę Google. Napisany w C++, umożliwia on kompilację języka JavaScript do natywnego kodu maszynowego, zamiast jego interpretacji. Środowisko Node.js dostępne jest dla większość współczesnych systemów operacyjnych, w tym dla platform z rodzin Linux, Windows i MacOs [22].

4.7. Express.js

Express.js to minimalistyczny framework stworzony z myślą o tworzeniu oprogramowania działającego w ramach serwerowych aplikacji web. Cytując fragment pozycji [23], dostarcza on mechanizmy takie jak:

- *Tworzenie funkcji obsługujących żądania o różnych metodach HTTP i skierowanych do różnych ścieżek w URL (tzw. routing).*
- *Integrację z różnymi silnikami do generowania widoków, które są tworzone na podstawie osadzanych danych w szablonach stron.*
- *Konfigurowania typowych ustawień aplikacji webowych jak np. portu, lokalizacji szablonów do generowania widoków odpowiedzi.*
- *Dodatkowe przetwarzanie żądań w warstwie pośredniej (tzw. "middleware"), które może być umieszczone w dowolnym miejscu łańcucha obsługi żądania.*

Jedną z cech charakterystycznych Express.js jest jego elastyczność. Framework ten nie narzuca programiście konkretnej struktury i pozwala na dużą dowolność. Express zapewnia bardzo prostą konfigurację i umożliwia uruchomienie prostego serwera HTTP w zaledwie kilku krokach (patrz listing 4).

Listing 4. Przykład prostego serwera HTTP napisanego przy użyciu Express.js. Źródło: [23]

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('Hello World!');
});

app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

4.8. MongoDB

MongoDB to nierelacyjna baza danych napisana w języku C++. Informacje są w niej przechowywane jako kolekcje dokumentów w postaci JSON (z ang. *JavaScript Object Notation*). Jest to lekki format, który łatwo zdeserializować do postaci struktury w języku JavaScript. MongoDB posiada szereg funkcjonalności znanych z klasycznych baz SQL, takich jak mechanizm transakcji czy indeksów. Oferuje również szereg możliwości stworzonych z myślą o aplikacjach działających w chmurze. Jedną z nich jest MongoDB Atlas, która została wykorzystana w ramach niniejszej pracy. Pozwala ona w łatwy sposób wdrożyć bazę danych na platformę chmurową AWS, GCP lub Azure oraz udostępnia łatwy w obsłudze panel administracyjny.

4.9. Moongoose

Jest to popularny system ORM (z ang. *Object–relational mapping*) dla języka JavaScript, służący do mapowania obiektowego danych pochodzących z bazy MongoDB. To stosunkowo proste narzędzie pozwalające zbudować odpowiednią warstwę abstrakcji, która pozwala na odseparowanie szczegółów związanych z formatem bazy danych od reszty aplikacji.

4.10.Puppeteer

To biblioteka języka JavaScript umożlwiająca obsługę przeglądarki internetowej Chromium w sposób całkowicie zautomatyzowany. Pozwala na zaprogramowanie prostych akcji dostępnych z poziomu witryny web, takich jak wypełnianie formularzy czy klikanie przycisków. W związku z tym jest to narzędzie, które wykorzystać można na potrzeby tworzenia testów funkcjonalnych aplikacji webowych lub web scrapingu. Puppeteer posiada również możliwości odczytu danych znajdujących się na odwiedzanej stronie internetowej, korzystając z języka selektorów HTML. Przydatną funkcją biblioteki jest możliwość tworzenia zrzutów ekranu, co znacznie ułatwia proces wykrywania oraz naprawy ewentualnych błędów.

Listing 5. Przykład zastosowania biblioteki Puppeteer do ekstrakcji danych z witryny web

```
const puppeteer = require('puppeteer')

async function scrape() {
  const browser = await puppeteer.launch({});
  const page = await browser.newPage();

  await page.goto('https://example.com/');

  const data = await page.evaluate( () =>
    document.querySelector('body > div > h1').textContent );

  return data;
}
console.log(scrape());
```

```
Wyjście: Example Domain
```

5. Interfejs użytkownika

Graficzny interfejs użytkownika został zaimplementowany w ramach aplikacji klienckiej. Składa się on z 3 podstawowych komponentów:

1. **Menu nawigacyjne boczne** – odpowiedzialne za nawigację po podstronach aplikacji.
2. **Menu nawigacyjne górne** – służy do zarządzania kontem użytkownika.
3. **Ciało** – właściwa część każdego widoku, w ramach której wyświetlane są dane sczytane z serwera. Element ten jest zależny od aktualnej podstrony.

Aplikacja implementuje szereg ścieżek, za pomocą których użytkownik może uzyskać dostęp do różnych widoków. Tabela 2 zestawia wszystkie dostępne w serwisie podstrony, wraz z ich adresami.

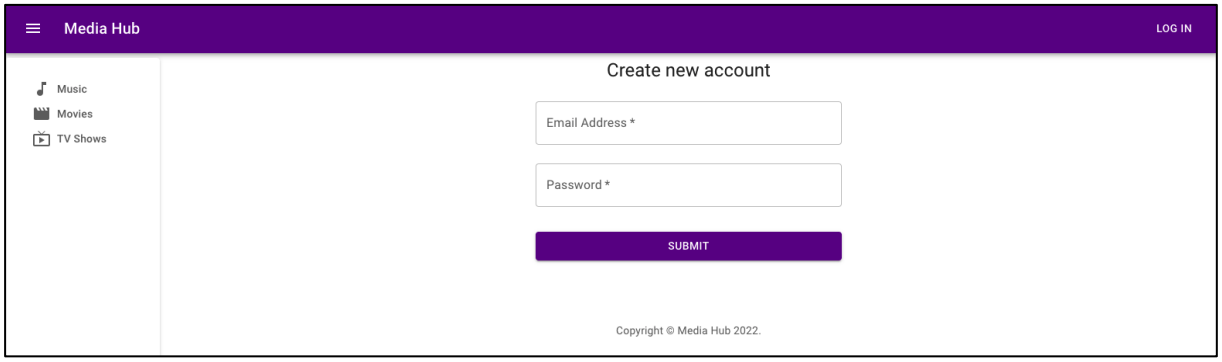
Tabela 2. Widoki dostępne z poziomu aplikacji *front-end*

Ścieżka	Widok
/user/login	Widok logowania użytkownika
/user/register	Widok rejestracji użytkownika
/user/dashboard	Panel użytkownika do zarządzania kontami w serwisach strumieniowych
/music/new	Widok nowododanych albumów
/music/album/:artist/:title	Szczegółowy widok albumu
/music/search/album/:query	Wyniki wyszukiwania albumów
/music/playlist/index	Przegląd popularnych playlist oraz playlist użytkownika
/music/playlist/:service/:id	Szczegółowy widok playlisty
/music/search/playlist/:query	Wyniki wyszukiwania playlist
/movie/overview	Przegląd popularnych filmów oraz filmów dodanych przez użytkownika do listy „do obejrzenia”
/movie/details/:id	Szczegółowy widok filmu
/movie/new	Widok nowododanych filmów
/movie/search/:query	Wyniki wyszukiwania filmów
/tv/new	Widok nowododanych seriali
/tv/details/:id	Szczegółowy widok serialu
/tv/overview	Przegląd popularnych seriali oraz seriali dodanych przez użytkownika do listy „do obejrzenia”
/tv/search/:query	Wyniki wyszukiwania seriali

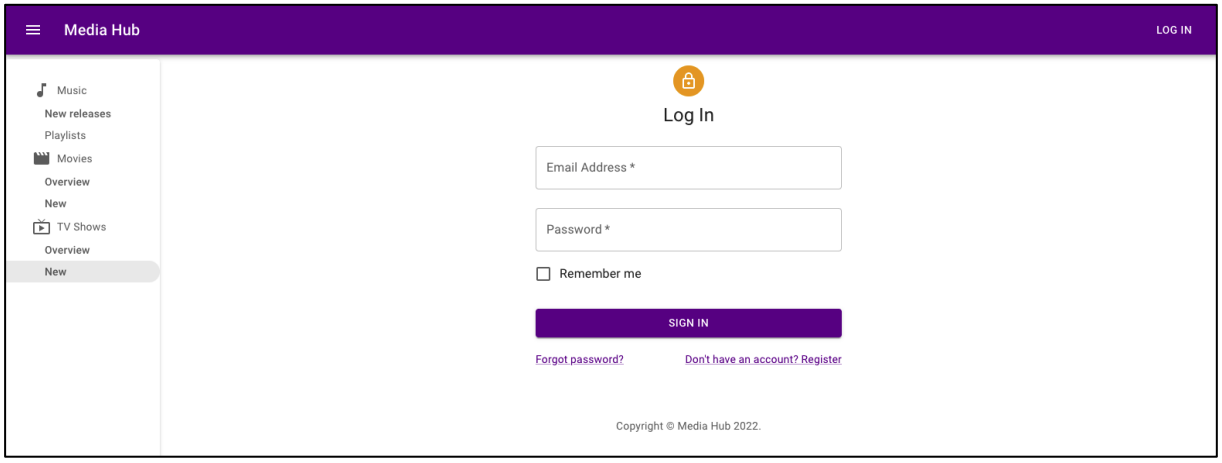
5.1. Logowanie i rejestracja

Aplikacja umożliwia stworzenie konta użytkownika. Nie jest to proces wymagany do korzystania z serwisu. Niemniej, niektóre funkcje systemu są niedostępne dla osób niezalogowanych. Widok ekranu rejestracji przedstawiony został na rysunku 9. Po stworzeniu konta użytkownik ma możliwość

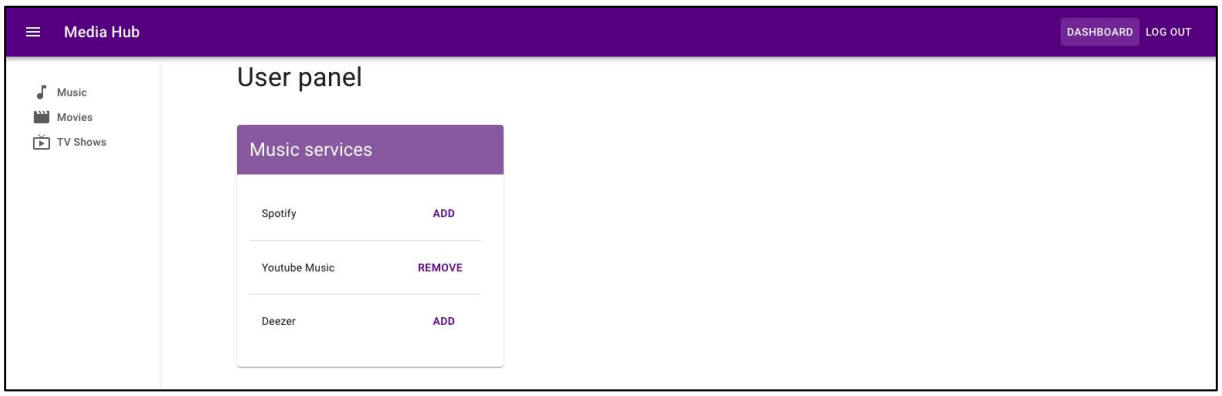
zalogowania się (rysunek 10), aby uzyskać dostęp do panelu użytkownika (rysunek 11). Z jego poziomu udostępniona została możliwość zarejestrowania kont z serwisów Spotify, YouTube Music i Deezer w usłudze (przykład: rysunek 12). Po poprawnym przeprowadzeniu tego procesu aplikacja może odczytać dane na temat playlist dodanych przez użytkownika w tych serwisach.



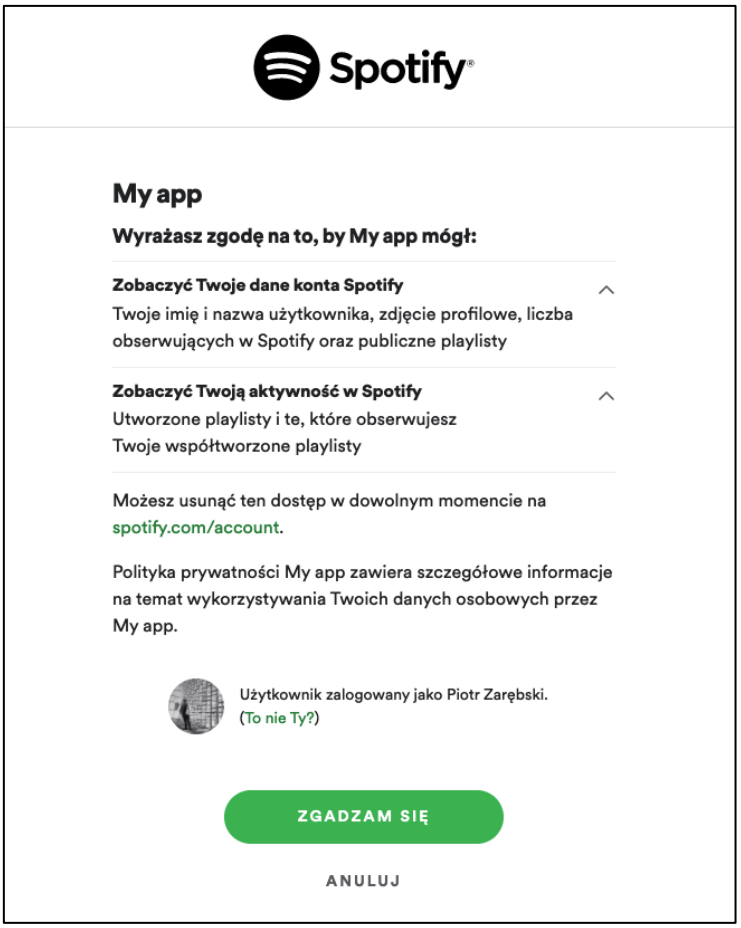
Rysunek 9. Ekran rejestracji użytkownika



Rysunek 10. Ekran logowania użytkownika



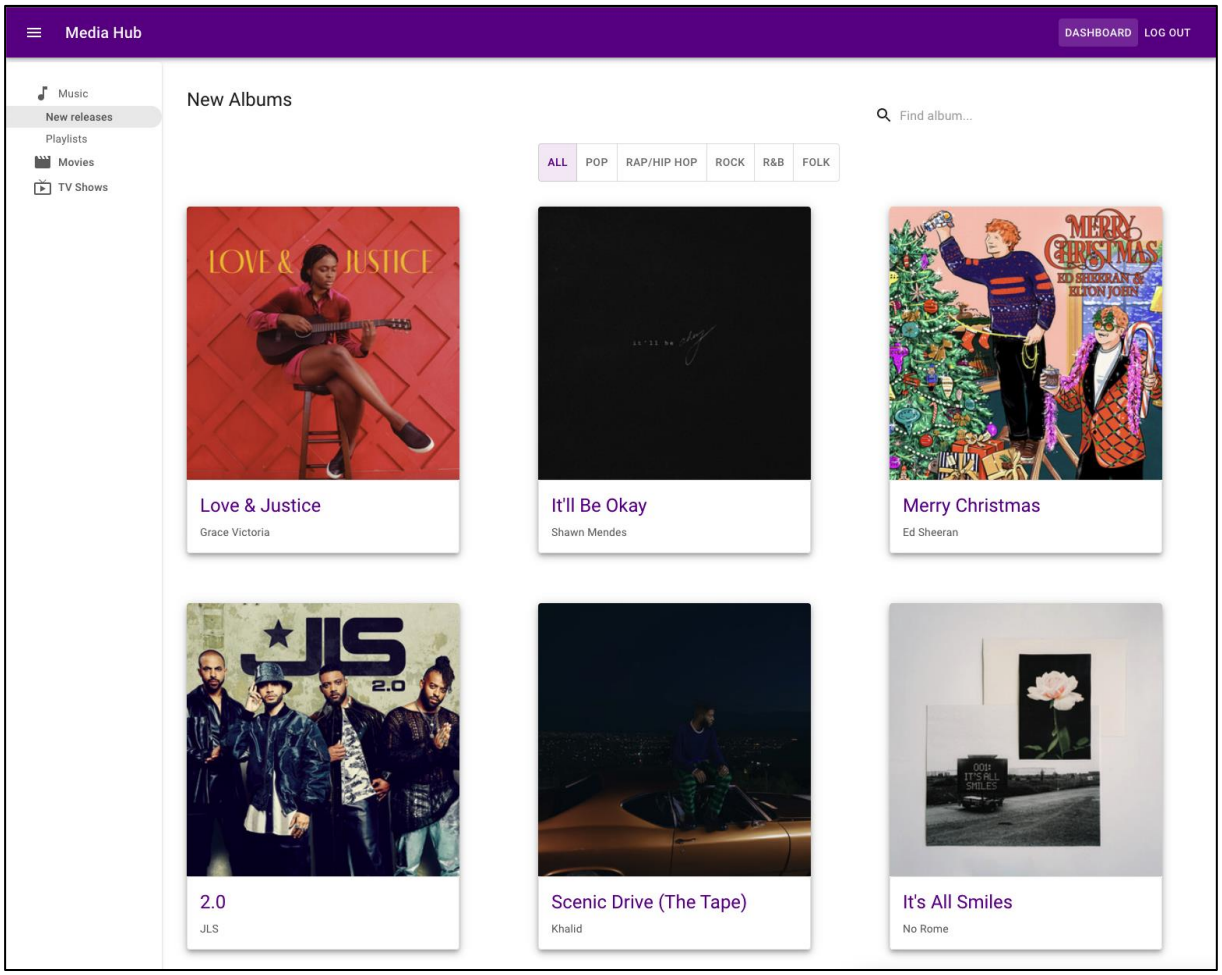
Rysunek 11. Panel użytkownika



Rysunek 12. Autoryzacja dostępu dla zewnętrznej aplikacji w serwisie Spotify

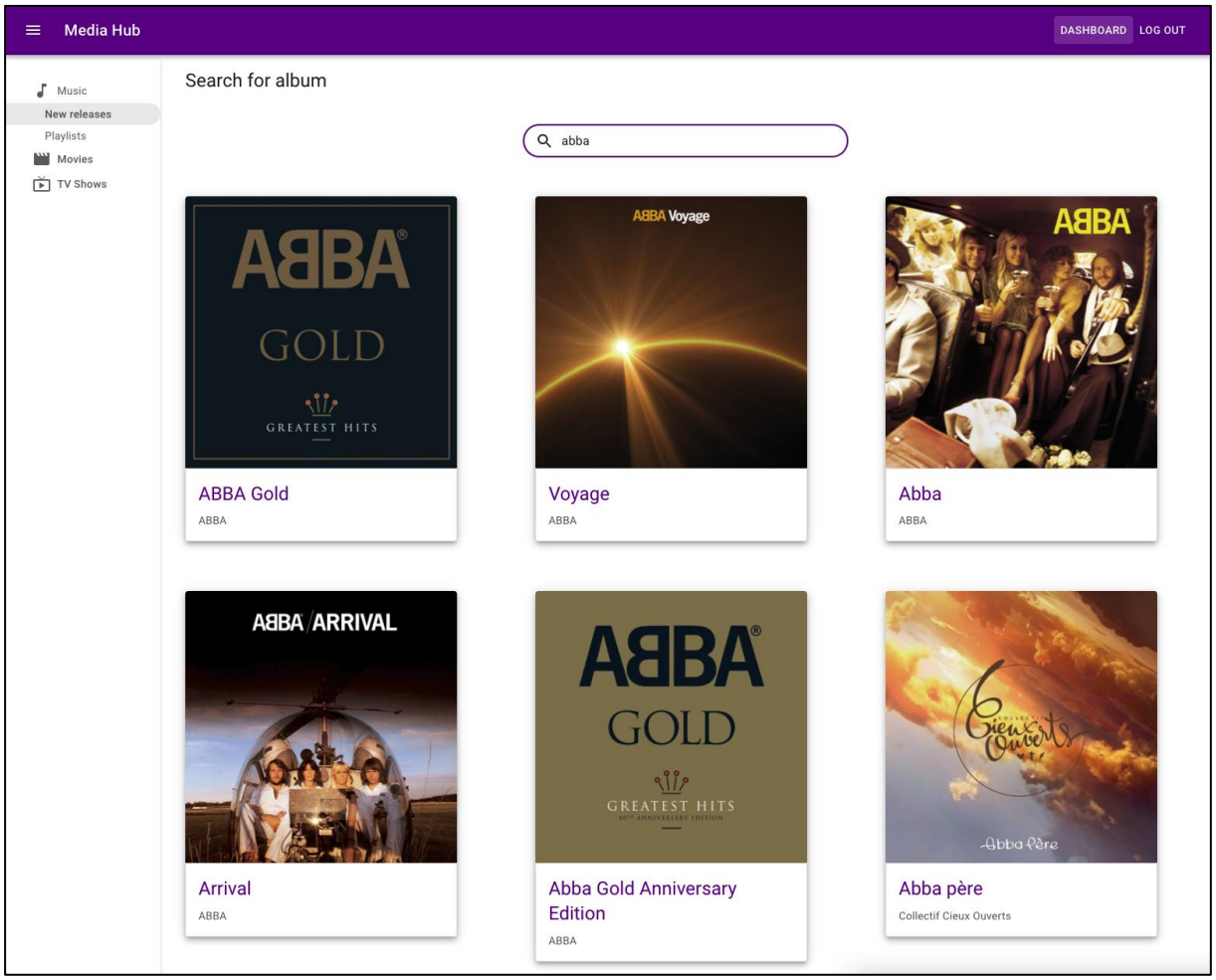
5.2. Albumy muzyczne

Projektowana platforma pozwala przeglądać dane muzyczne dostępne w serwisach strumień w 2 formach: jako albumy oraz jako playlisty. Scharakteryzujmy najpierw możliwości związane z pierwszym typem. Podgląd nowych wydawnictw muzycznych opublikowanych w ramach wszystkich dostępnych w aplikacji źródeł przedstawiony został na rysunku 13. Albumy wyświetlane są w kolejności chronologicznej, od najnowszych do najstarszych, z możliwością filtrowania po gatunku muzycznym.

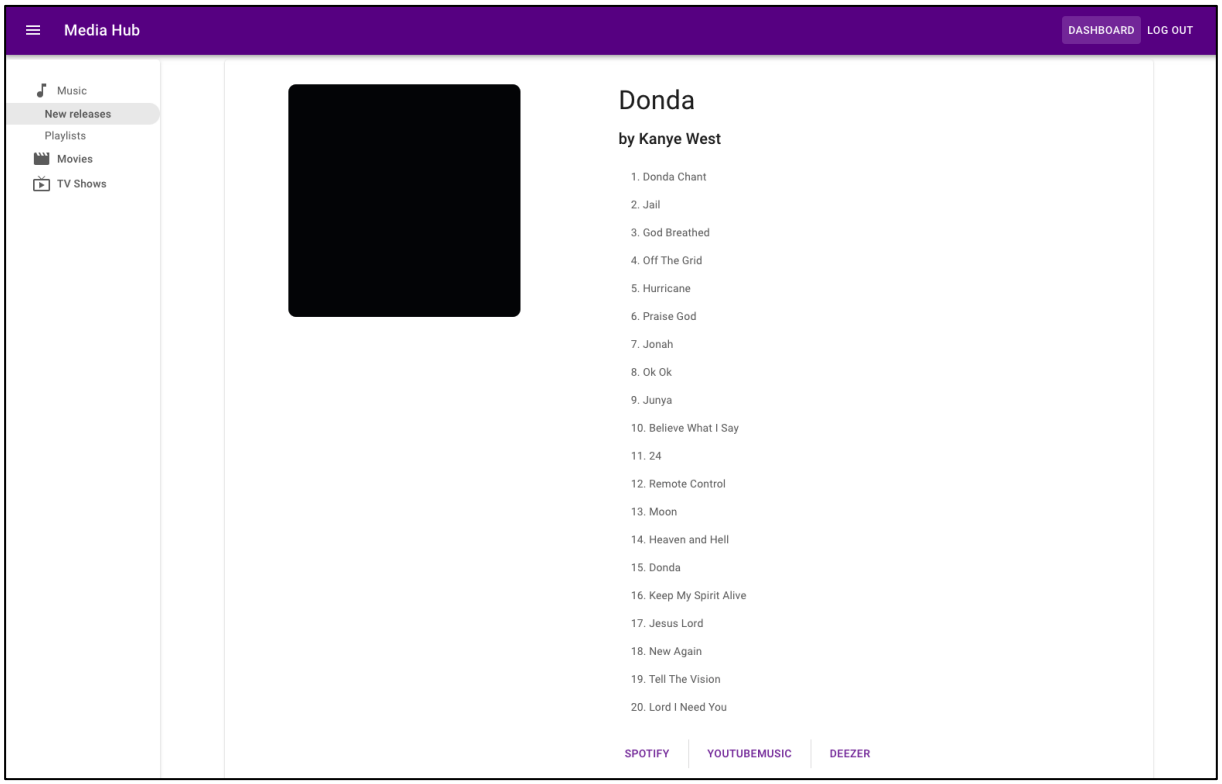


Rysunek 13**.** Nowododane albumy w serwisach strumieniowych

W prawym górnym rogu widoku z rysunku 13 znajduje się pasek wyszukiwania albumu. Za jego pomocą użytkownik może przeszukiwać ofertę serwisów strumieniowych w celu znalezienia interesujących go pozycji. Widok wyszukiwarki przedstawiony został na rysunku 14. Po kliknięciu w obiekt reprezentujący album następuje przekierowanie do podstrony, w ramach której prezentowane są szczegółowe informacje na jego temat (patrz rysunek 15). Oprócz autora i nazwy albumu są to również lista utworów oraz dane o dostępności w serwisach strumieniowych w postaci przycisków. Po ich naciśnięciu, następuje przekierowanie do odpowiedniej podstrony streamingu, w ramach której można odsłuchać album.



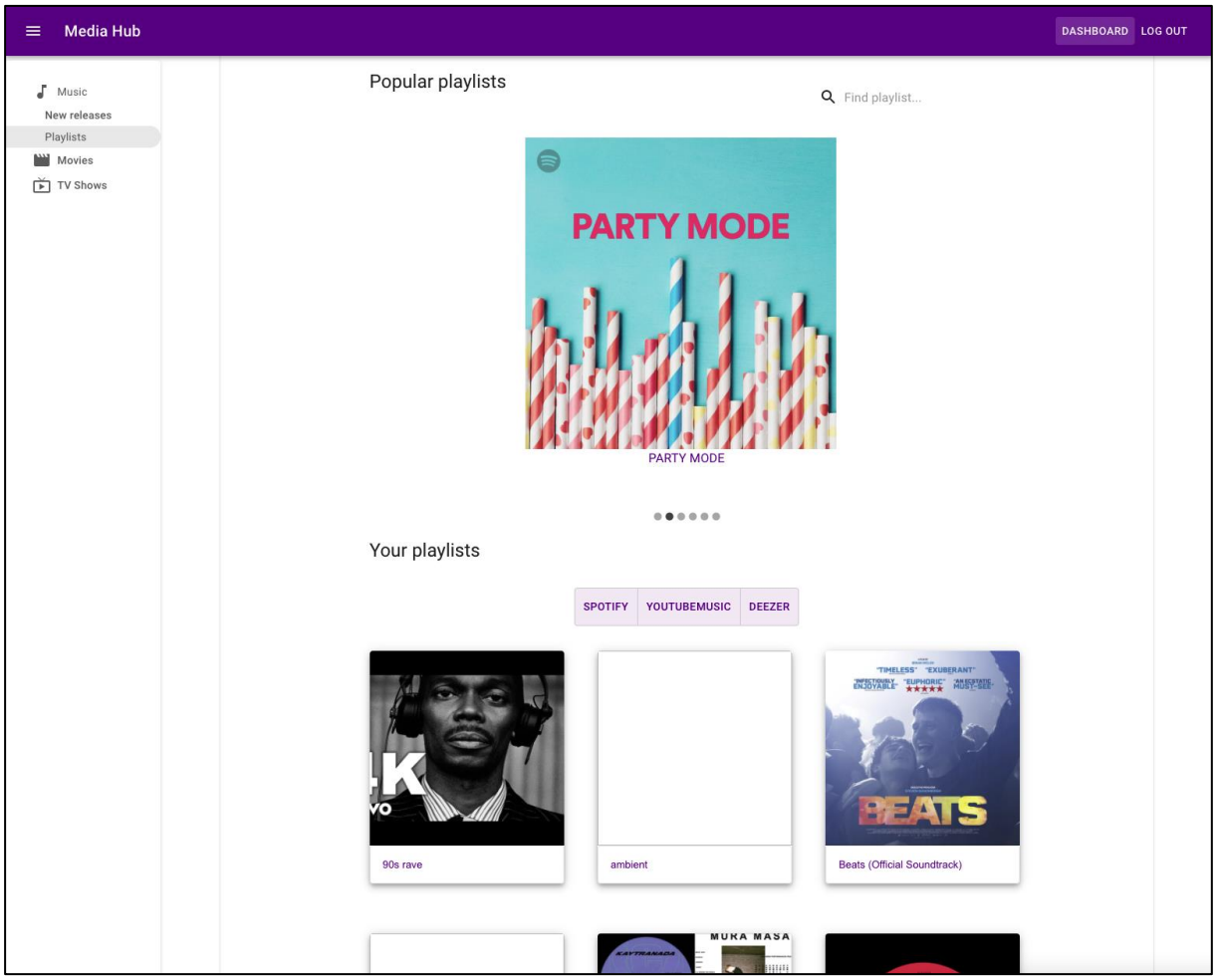
Rysunek 14**.** Widok wyszukiwania albumów



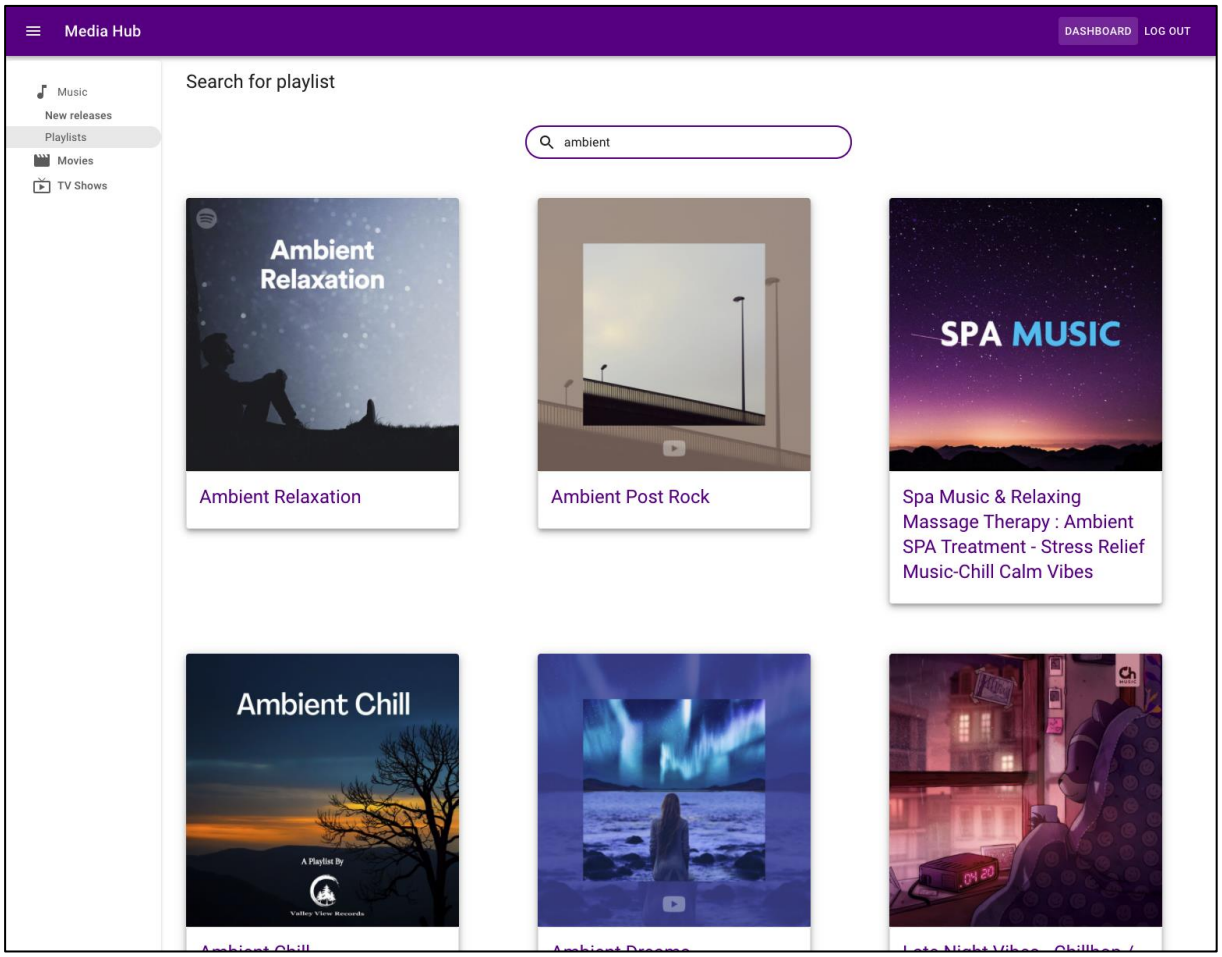
Rysunek 15. Podgląd informacji na temat albumu muzycznego

5.3. Playlisty

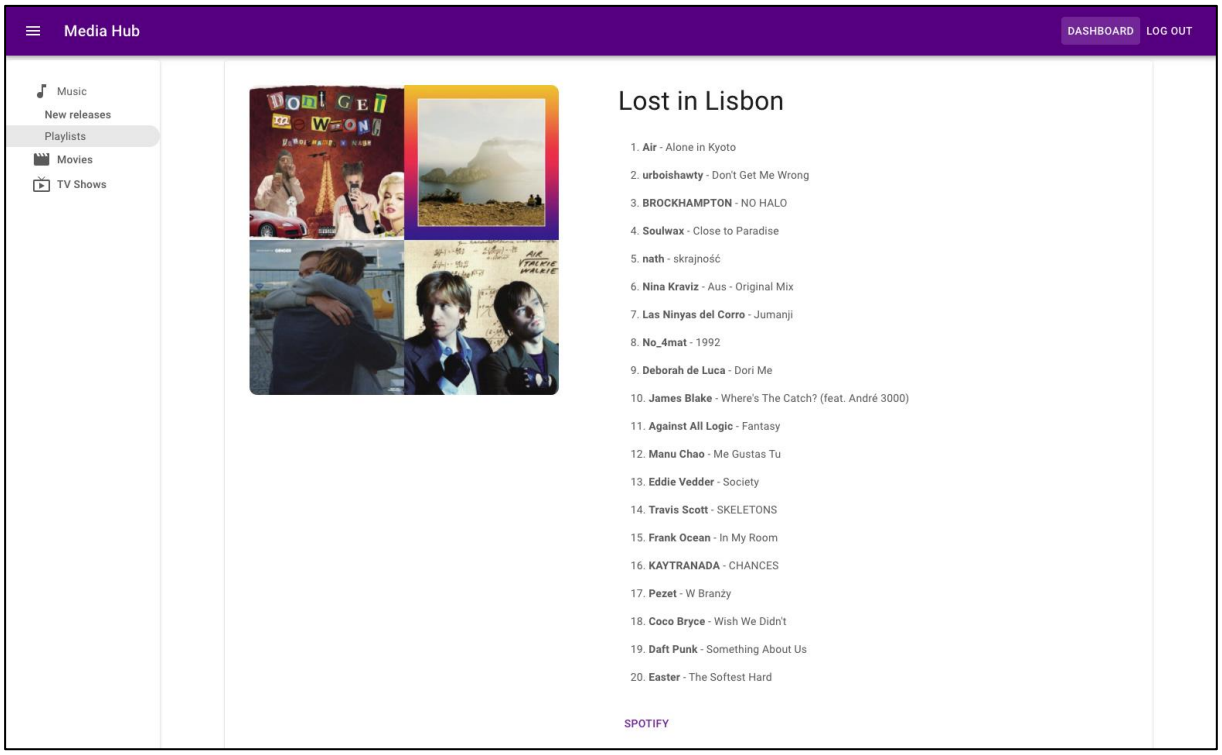
Główny widok playlist (rysunek 16) pozwala na eksplorację najpopularniejszych z nich dostępnych w usługach strumieniowych. Dane te zaprezentowane są w formie karuzeli. Po zalogowaniu się, w ramach tego widoku wyświetlane są również playlisty użytkownika, zebrane z różnych serwisów. Pozycje prezentowane są w porządku alfabetycznym z możliwością filtrowania po źródle. Aplikacja udostępnia również funkcję wyszukiwania publicznych playlist (rysunek 17). Poprzez naciśnięcie obiektu reprezentującego listę, można przejść bezpośrednio do jej szczegółowego (rysunek 18). W jego ramach wylistowane są wszystkie utwory znajdujące się na playliscie oraz link pozwalający na przejście do serwisu strumieniowego.



Rysunek 16. Widok popularnych oraz zapisanych przez użytkownika playlist



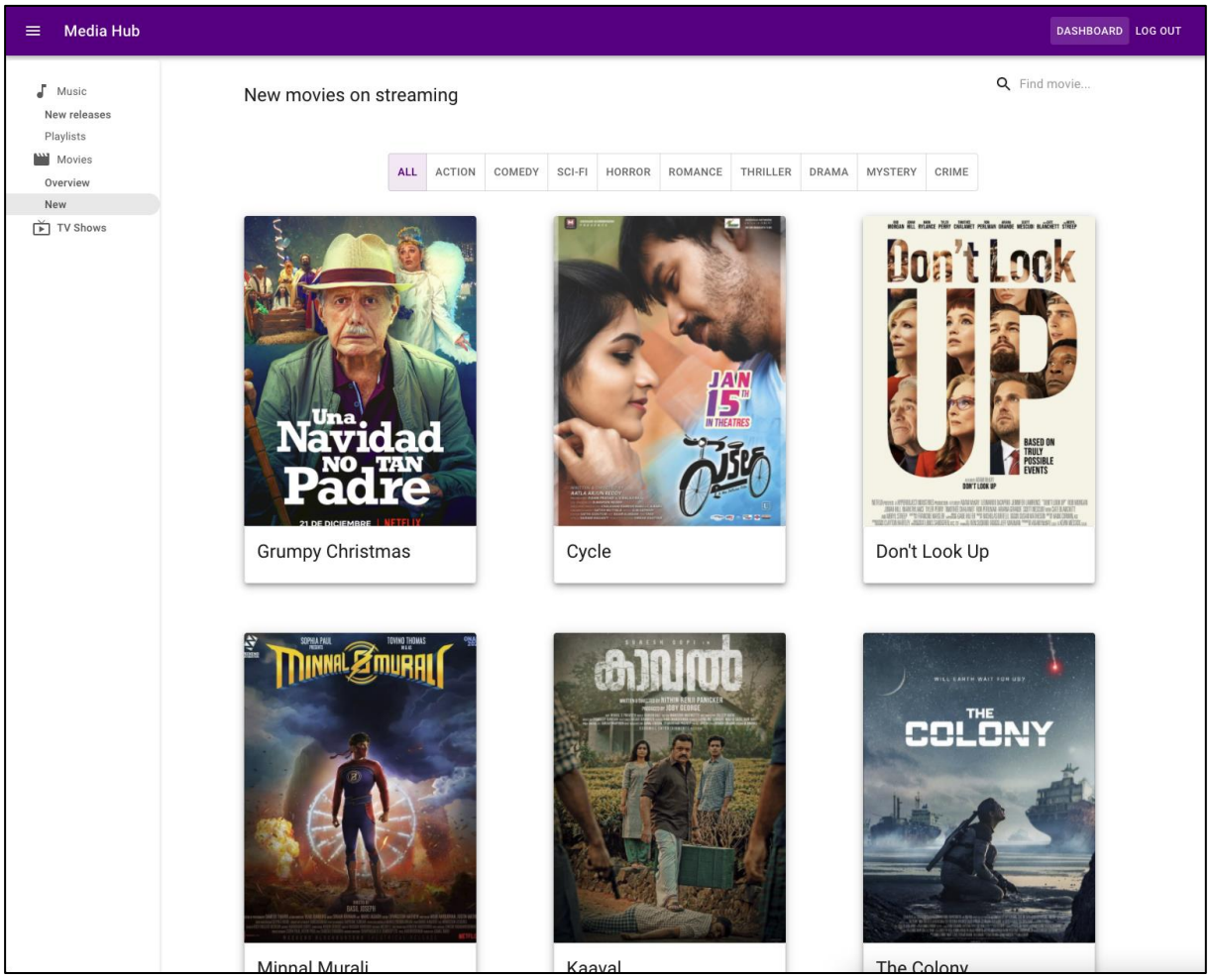
Rysunek 17. Widok wyszukiwania playlist



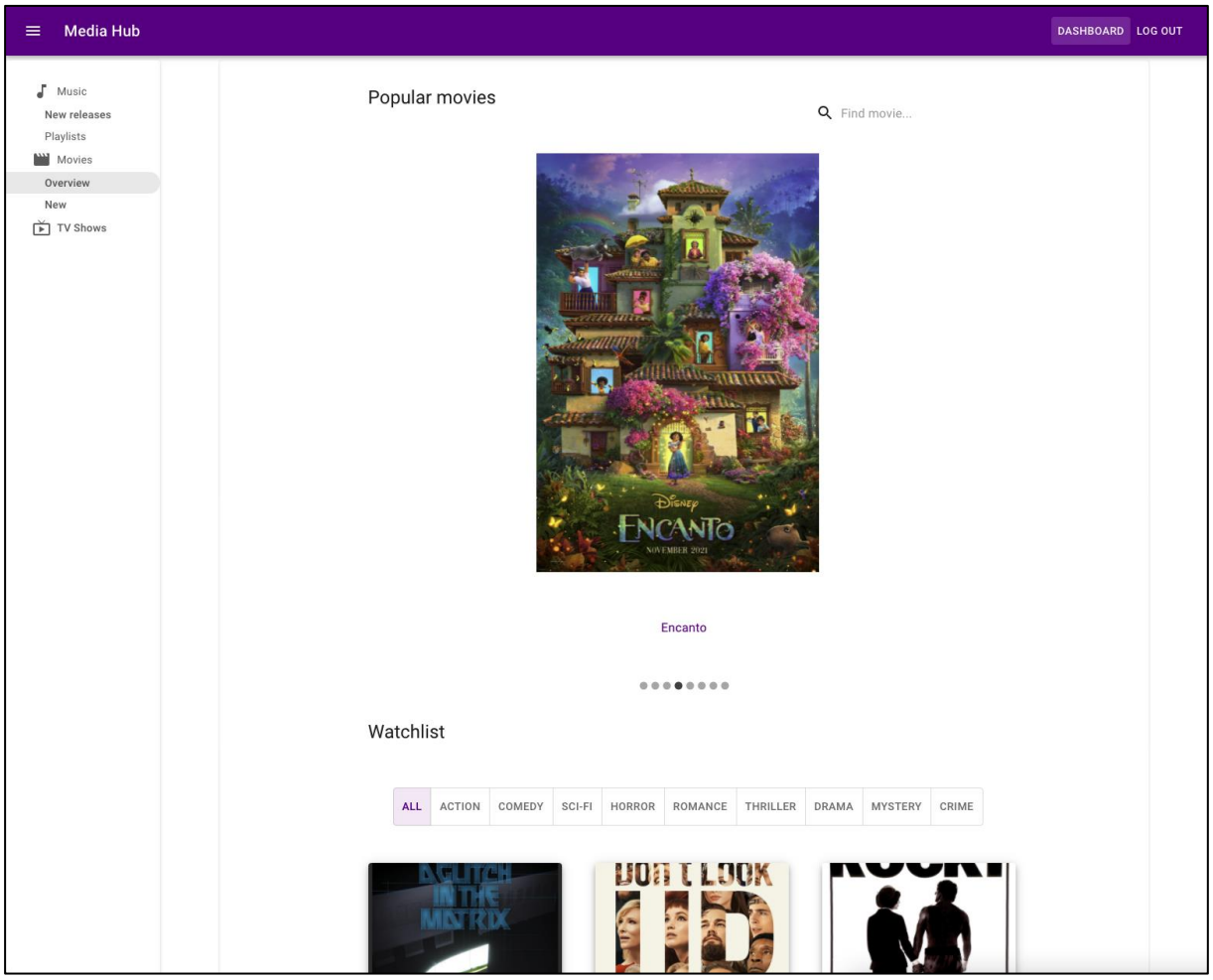
Rysunek 18**.** Podgląd informacji na temat playlisty

5.4. Filmy

Na potrzeby funkcjonalność przeglądania filmów dostępnych w serwisach strumieniowych wydzielono 2 główne widoki dostępne w ramach ścieżek: /movie/new (rysunek 19) i /movie/overview (rysunek 20). Na pierwszej z nich użytkownik może przeglądać nowe pozycje dostępne na streamingach. Ze względu na wymagania wydajnościowe, dane dostępne z poziomu tego widoku są przechowywane w pamięci tymczasowej systemu i aktualizowane są w interwale 1 dnia. Filmy wyświetlane są w porządku chronologicznym, od najnowszych do najstarszych, z możliwość filtrowania po gatunku.

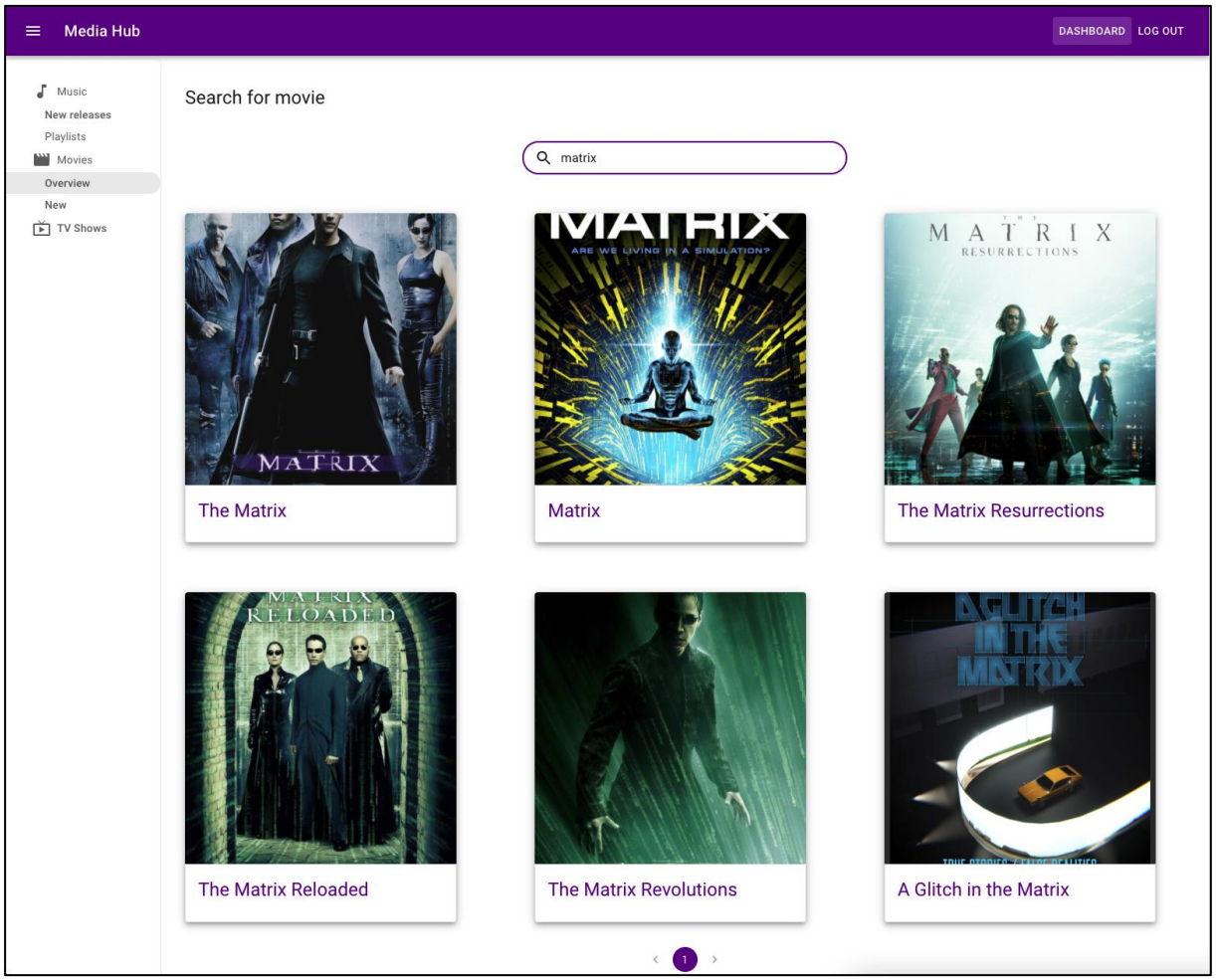


Rysunek 19**.** Nowododane filmy w serwisach strumieniowych

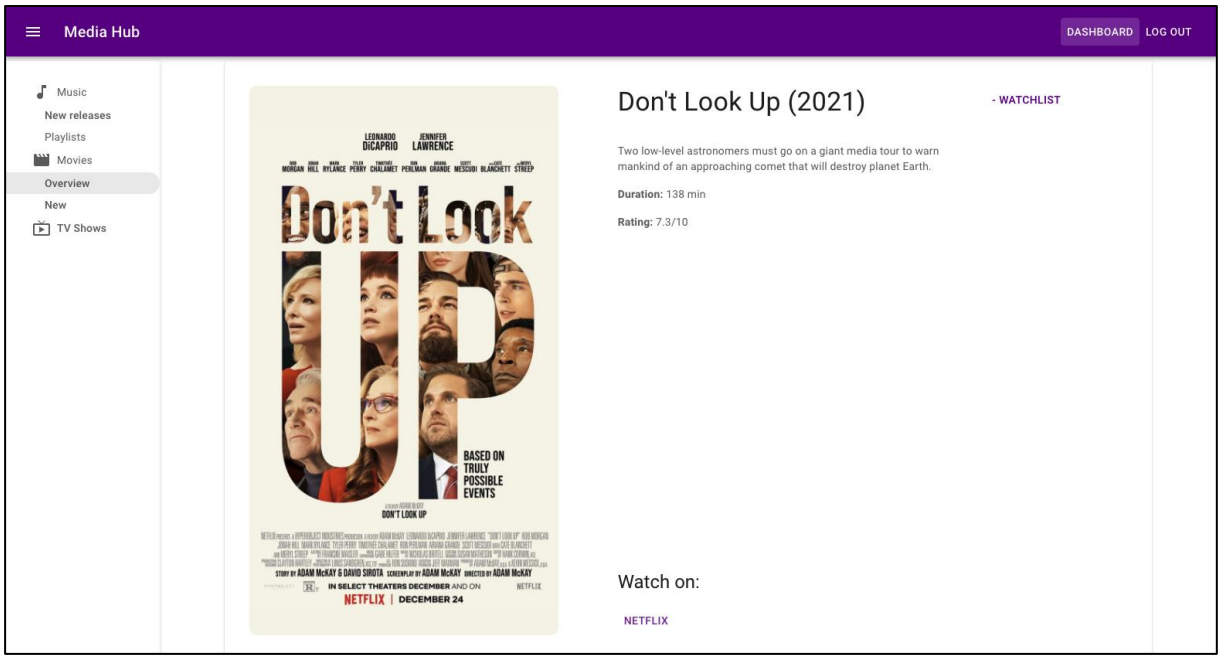


Rysunek 20**.** Widok popularnych filmów oraz lista „do obejrzenia”

Widok zaprezentowany na rysunku 20 zestawia najpopularniejsze filmy (na podstawie danych z serwisu IMDb) w formie karuzeli. Zalogowani użytkownicy mają ponadto dostęp do pozycji zapisanych jako „do obejrzenia”. Platforma przewiduje również funkcję wyszukiwania filmów (rysunek 21). Informację na temat konkretnej pozycji dostępne są z poziomu dedykowanej podstronie (rysunek 22). W jej ramach dowiedzieć się można o czasie trwania filmu, ocenie użytkowników w serwisie IMDb oraz dostępności w serwisach strumieniowych. Zalogowani użytkownicy mogą również dodać lub usunąć filmu z listy „do obejrzenia”.



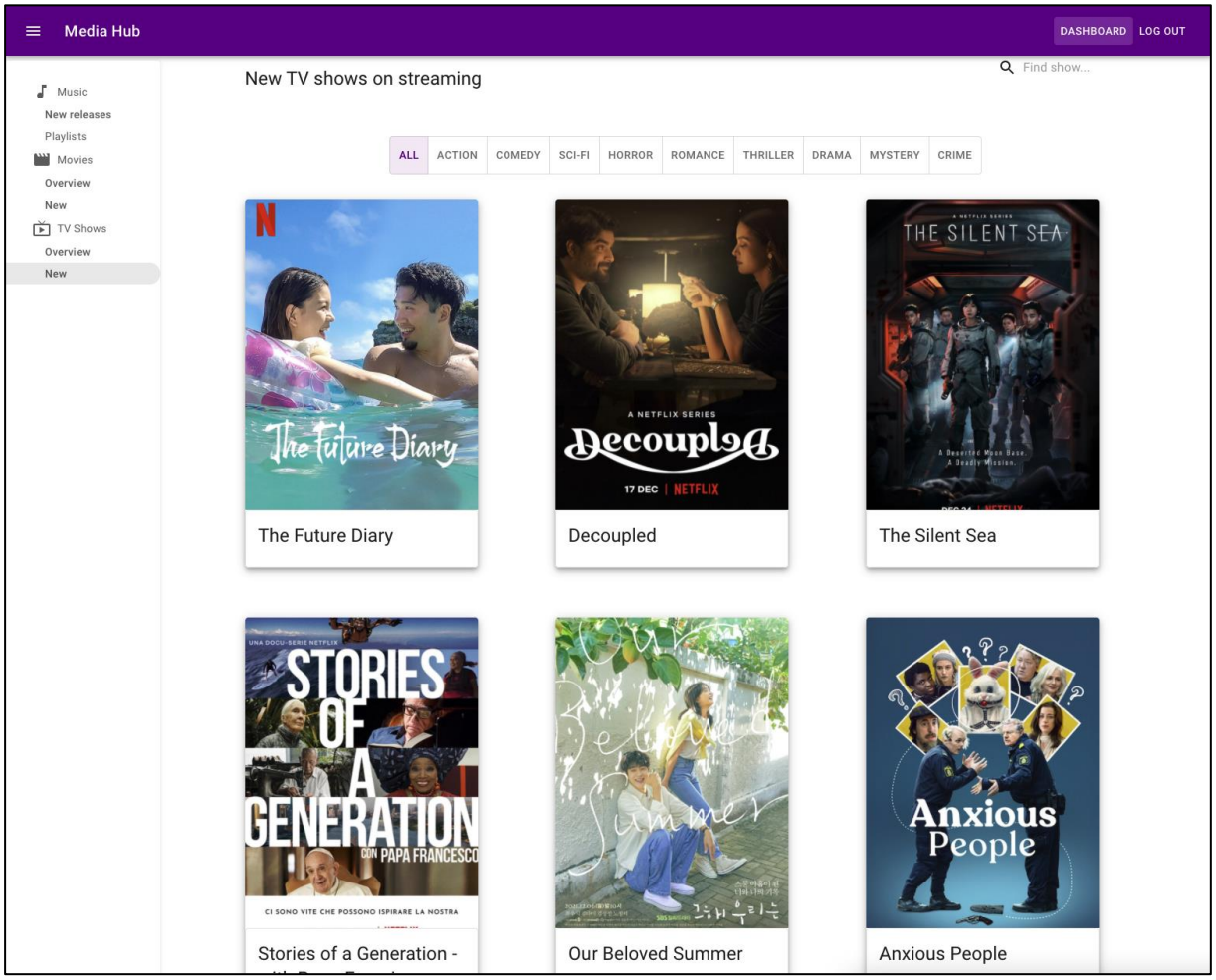
Rysunek 21. Widok wyszukiwania filmów



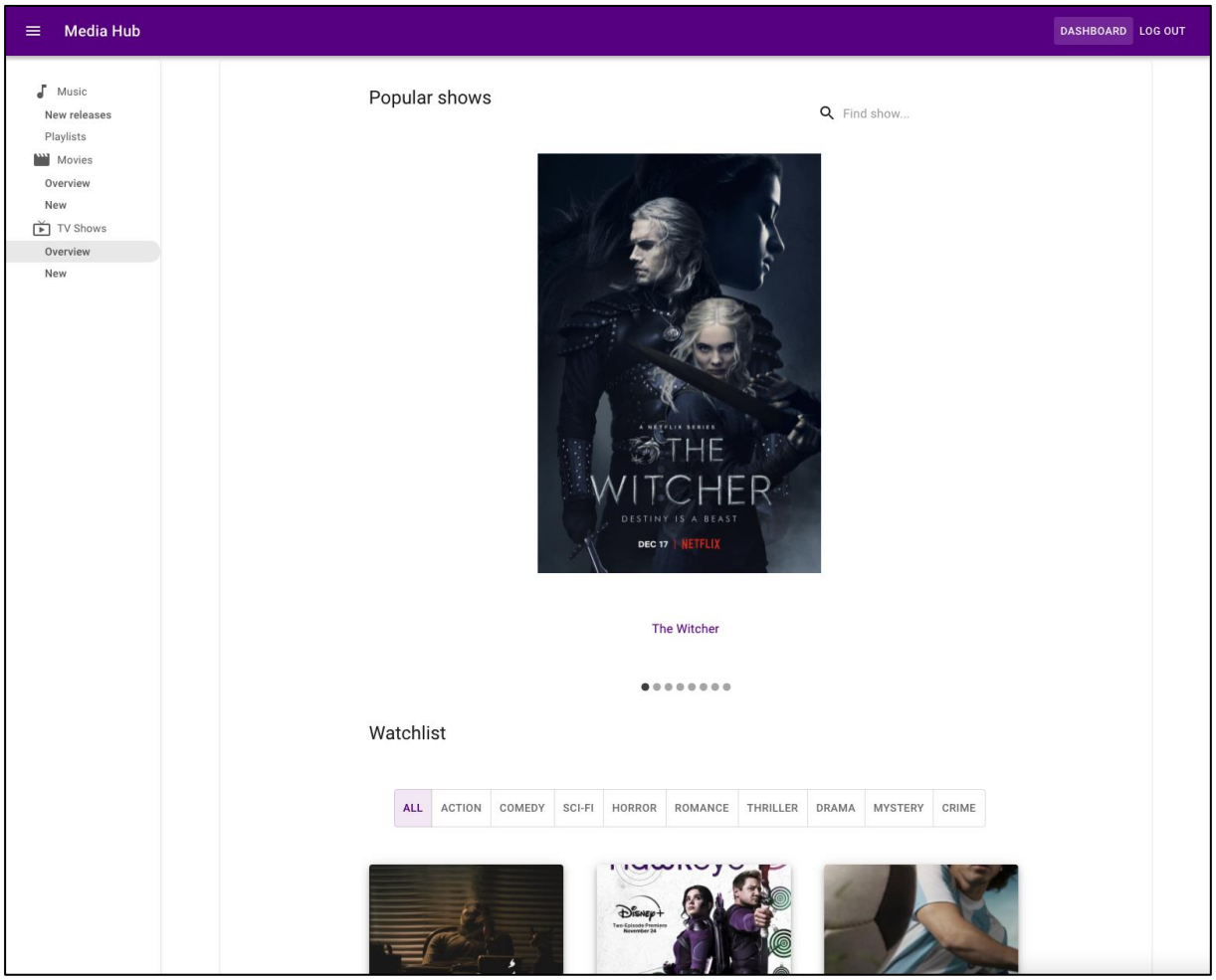
Rysunek 22. Podgląd informacji na temat filmu

5.5. Seriale

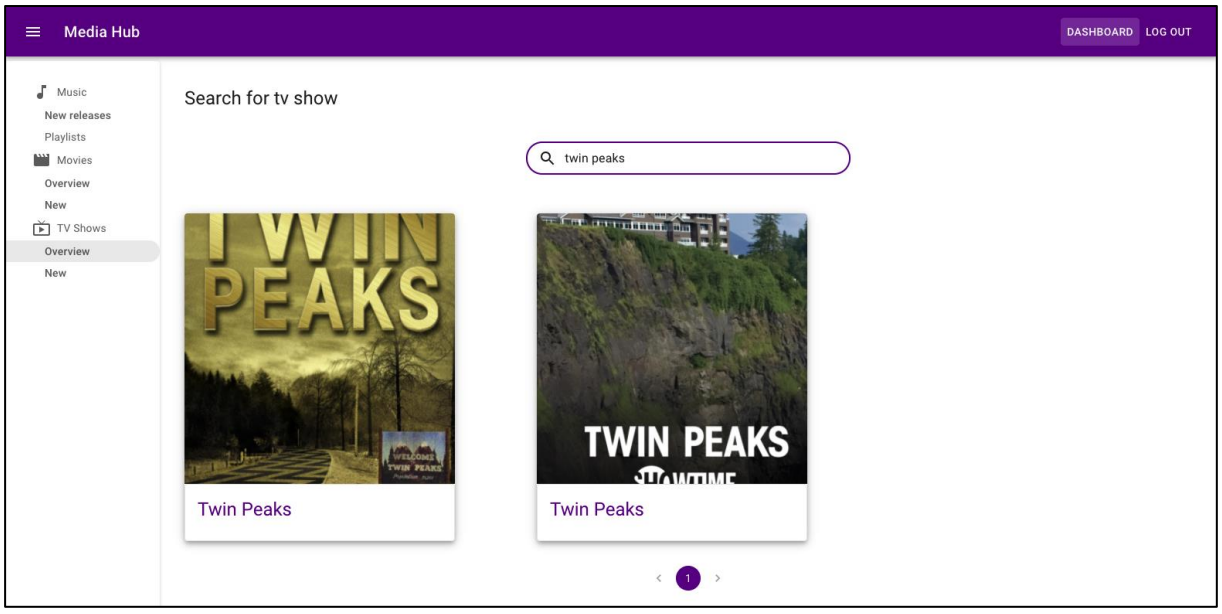
Funkcjonalności przewidziane dla modułu odpowiedzialnego za obsługę seriali są bliźniaczo podobne do tych związanymi z filmami. Widok nowododanych seriali zamieszczony został na rysunku 23. Eksploracja popularnych oraz zapisanych pozycji dostępna jest z poziomu podstrony przedstawionej na rysunku 24.System przewiduje również funkcjonalność wyszukiwania (patrz rysunek 25). Podgląd szczegółowych informacji na temat serialu dostępny jest z poziomu widoku przedstawionego na rysunku 26.



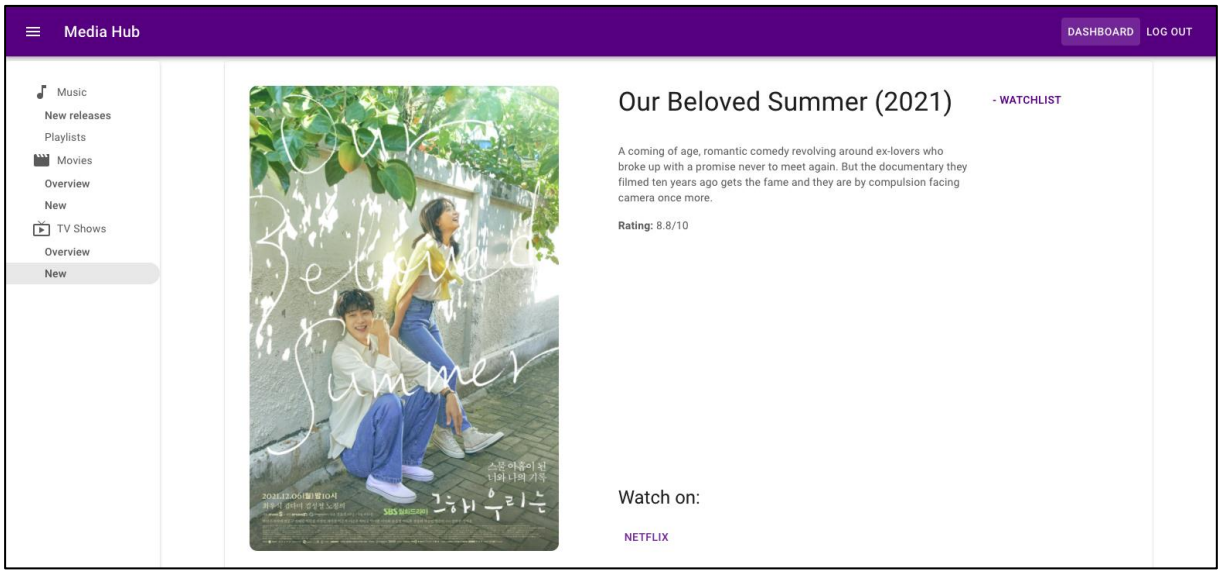
Rysunek 23**.** Nowododane seriale w serwisach strumieniowych



Rysunek 24. Widok popularnych seriali oraz lista „do obejrzenia”



Rysunek 25. Widok wyszukiwania seriali



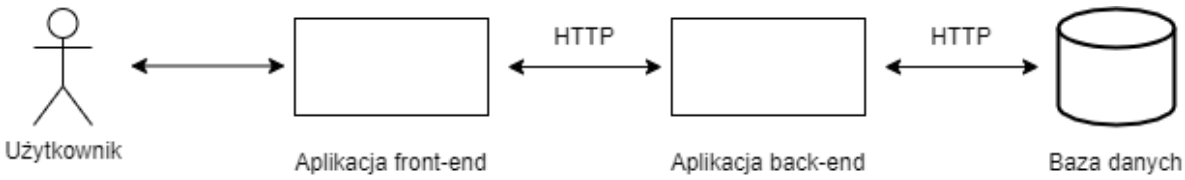
Rysunek 26**.** Podgląd informacji na temat serialu

6. Opis implementacji

Rozdział ten poświęcony został szczegółowemu opisowi implementacji poszczególnych komponentów, z których składa się projektowany system. W jego ramach przybliżono również napotkane w czasie tworzenia aplikacji problemy techniczne oraz strategie, które zastosowano w celu ich rozwiązania. Pierwsza część zajmuje się charakterystyką graficznego interfejsu użytkownika. Opisuje jego elementy składowe oraz dostępne w jego ramach widoki. Następne podrozdziały poświęcone zostały aplikacji serwerowej. Zawierają charakterystykę poszczególnych modułów wchodzących w jej skład, ze szczególnym naciskiem na wyzwania programistyczne związane z ich implementacją. W części tej opisano również zewnętrzne interfejsy API, z których korzysta aplikacja oraz metody web scraping użyte w celu uzyskania danych z serwisów strumieniowych.

6.1. Architektura systemu

W toku prac nad przygotowaniem systemu do agregacji danych multimedialnych zdecydowano się wydzielić z niego dwie zasadnicze części. Aplikację *front-end*, działającą w przeglądarce internetowej po stronie klienta oraz aplikację *back-end*, wykonującą się po stronie. Oba komponenty komunikują się za pomocą interfejsu API zaprojektowanego zgodnie z zasadami stylu REST (ang. representational state transfer) po protokole HTTP. Schemat architektury systemu przedstawia rysunek 27.



Rysunek 27. Schemat architektury systemu

6.1.1 Aplikacja *front-end*

Aplikacja *front-end* to oparte o technologie HTML, CSS oraz JavaScript oprogramowanie wykonujące się po stronie klienta systemu. Moduł ten odpowiada za bezpośrednią komunikację z użytkownikiem i stanowi dla niego interfejs graficzny, dostępny za pośrednictwem przeglądarki. Aplikacja zaprojektowana została zgodnie z nowoczesnymi standardami witryn internetowych. Ma charakter dynamiczny, tzn. jest w stanie sczytać dane z serwera „w locie", bez potrzeby przeładowania strony.

6.1.2 Aplikacja *back-end*

Moduł ten powstał z myślą o działaniu jako serwer HTTP. Do jego głównych zadań polega obsługa API serwisów strumieniowych, obsługa web scrapingu oraz zapis i odczyt z bazy danych. Dostęp do tej aplikacja uzyskać można za pośrednictwem interfejsu API, opracowanego zgodnie z zasadami REST.

6.2. Aplikacja kliencka

Aplikacja *front-end* powstała przy wykorzystaniu języka JavaScript i biblioteki React.js, w oparciu o wzorzec projektowy SPA (z ang. *Single Page Application*). Widok, w postaci pliku HTML, generowany jest jednokrotnie. W wyniku interakcji użytkownika jest on dynamicznie modyfikowany, bez potrzeby przeładowywania całej strony. Dane z serwera odczytywane są w formacie JSON przy wykorzystaniu techniki AJAX. Sprawia to, że kliencka część systemu jest znacznie bardziej złożona niż w przypadku klasycznych, statycznych stron. Podejście to niesie ze sobą również kilka trudności technicznych, których rozwiązania przybliżono w tym podrozdziale.

6.2.1 Nawigacja

W ramach projektowanej witryny dostępnych jest wiele podstron. W zależności od wybranej ścieżki, użytkownikowi prezentowane są inne dane, które najpierw muszą zostać pobrane z serwera. Wymagana jest więc implementacja odpowiedniego mechanizmu routingu. Zdecydowano się na wykorzystanie narzędzia `BrowserRouter`, będącego częścią biblioteki React. Ułatwia ono mapowanie podstrony z elementem, który powinien być w jej ramach wyświetlony. Użycie narzędzia przedstawiono na listingu 6. Rozszerzenie to udostępnia również inne możliwości, które wykorzystano przy implementacji aplikacji, jak np. odczytywanie parametrów z adresów URL czy funkcja dynamicznego przekierowywania do innej podstrony.

Listing 6. Mapowanie ścieżek w ramach aplikacji *front-end*

```
<Routes>
  <Route path="/user/login" element={<LogIn setToken={saveToken}/>} />
  <Route path="/user/register" element={<Register />} />
  <Route path="/user/dashboard" element={<Dashboard token={token}/>} />
  …
</Routes>
```

6.2.2 Komunikacja z serwerem

W odpowiedzi na interakcję ze strony użytkownika aplikacja odpytuje serwer przy użyciu protokołu HTTP. Zapytanie to nie powinno być jednak blokujące, aby uniknąć sytuacji, w której nie możliwe jest korzystanie ze strony w oczekiwaniu na odpowiedź. W celu rozwiązania tego problemu wykorzystano bibliotekę axios, która udostępnia warstwę abstrakcji służącą do wysyłania oraz przetwarzania żądań HTTP w sposób asynchroniczny. Przykład funkcji komunikującej się z serwerem przedstawiono na listingu 7.

Listing 7. Funkcja odpytująca serwer o nowododane albumy muzyczne

```
useEffect(() => {
  axios.get("http://localhost:8080/music/new",
    { params : { page, limit, gender }})
  .then(resp => {
    console.log(resp);
    if (resp.status === 200) {
      setNewReleases(resp.data.newReleases);
      setPageCount(resp.data.pageCount);
    }
    else {
      setErr(new Error(resp.data.message));
    }

  })
  .catch(err => {
    console.log(err);
    setErr(err);
  });
}, [page, gender]);
```

6.2.3 Zarządzanie stanem

Komponenty zaimplementowane w ramach aplikacji klienckiej są od siebie zasadniczo niezależne, a co za tym idzie nie posiadają współdzielonego stanu. Wyjątek stanowi token JWT zalogowanego użytkownika, który wykorzystywany jest przez różne moduły do podpisywania żądań HTTP wysyłanych do serwera. W celu implementacji globalnego stanu wykorzystano wbudowany w bibliotekę React mechanizm `useState`. Dane uzyskane po autoryzacji użytkownika zapisywane są w nadrzędnym komponencie aplikacji, w ramach którego zaimplementowano obsługę routingu przedstawioną na listingu 6. W momencie wyświetlania poszczególnych podstron, token przekazywany jest jako argument do odpowiedniego elementu.

Dodatkowo, informacja na temat obecnie zalogowanego użytkownika przechowywana jest w lokalnej przestrzeni dyskowej przeglądarki, jak pokazano na listingu 8. W związku z tym, proces logowania nie jest wymagany przy ponownym otwarciu strony. Dane te wyczyścić można poprzez wylogowanie się lub usunięcie plików cookie.

Listing 8. Funkcje do zarządzania stanem oraz zapisywania informacji o zalogowanym użytkowniku w lokalnej przestrzeni dyskowej przeglądarki

```
const [token, setToken] = useState(getToken());

const getToken = () => {
    return localStorage.getItem('token')
  };

  const saveToken = userToken => {
    localStorage.setItem('token', userToken);
    setToken(userToken);
  };

  const clearToken = () => {
    setToken(null);
    localStorage.clear('token');
  };
```

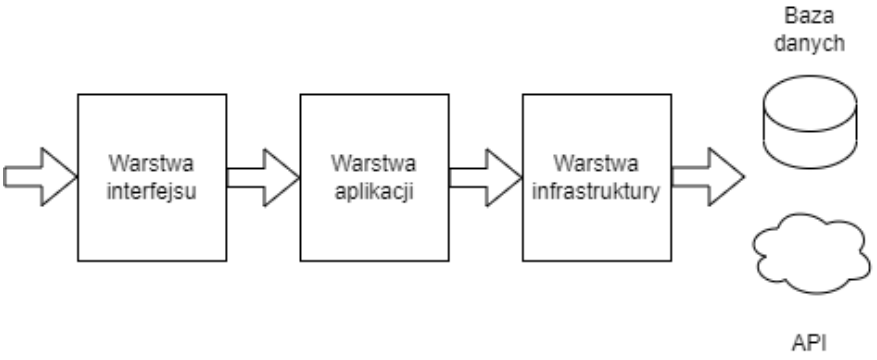
6.3. Aplikacja serwerowa

Kluczowe z punktu widzenia systemu funkcjonalności, takie jak sczytywanie informacji z serwisów strumieniowych oraz obsługa bazy danych zaimplementowane zostały w ramach modułu *back-end*. Aplikacja ta implementuje interfejs API zaprojektowany zgodnie z zasadami REST, który umożliwia jej komunikację z modułem *front-end*.

6.3.1 Architektura

Przy tworzeniu komponentu serwerowego zdecydowano się na zastosowanie wzorca architektury heksagonalnej [25] [26]. Wdraża on w praktycy postulaty metodyki DDD (ang. *Domain Driven Design*). Jego głównymi założeniami jest zapewnienie odpowiedniej hermetyzacji na 2 płaszczyznach. Z jednej strony podejście to zaleca wyodrębnienie powiązanych ze sobą tematycznie modułów, tzw. domen. Są one ze sobą luźno powiązane. Wzorzec ten narzuca również drugi podział przebiegający w poprzek domeny, który dzieli ją na warstwy:

1. **Interfejsu** – obsługującej sygnały wpływające do systemu (np. zapytania HTTP)
2. **Aplikacji** – implementującą właściwą logikę biznesową
3. **Infrastruktury** – odpowiadającą za odczyt i zapis danych, np. do bazy danych, brokera wiadomości czy API.



Rysunek 28**.** Przepływ danych w architekturze heksagonalnej

Choć literatura jest zgodna co do funkcji jakie spełniają powyższe warstwy, nazewnictwo znacząco różni się w zależności od pozycji. Autor pracy zdecydował się na przyjęcie konwencji, która jego zdaniem wydaje się najbardziej intuicyjna. Przepływ danych w ramach pojedynczej domeny przedstawia rysunek 28.

Ważną zasadą narzucaną przez architekturę heksagonalną jest ograniczenie komunikacji pomiędzy modułami. Wymiana informacji dozwolona jest wyłącznie w ramach warstwy aplikacji. Podejście to sprawia, że domeny są od siebie bardzo mało zależne. Tworzenie więc nowych modułów jest procesem stosunkowo mało kosztownym. Architektura ta, pomimo że stanowi przykład monolitu, dzięki dużej hermetyzacji poszczególnych części jest niejako przygotowana pod wyodrębnienia ich w przyszłości do osobnych mikroserwisów.

W ramach aplikację *back-end* wydzielono 4 funkcjonalne domeny: movie, music, tv oraz user. Organizacji plików w projekcie przedstawiona została na listingu 9.

Listing 9. Struktura plików w aplikacji *back-end*

```
.
├── app.ts
├── movie
│   ├── models
│   │   ├── dto
│   │   │   ├── MovieInfo.ts
│   │   │   └── MovieService.ts
│   │   ├── movieMetaModel.ts
│   │   ├── movieModel.ts
│   │   └── movieTypesMapper.ts
│   ├── movieController.ts
│   ├── movieServiceManager.ts
│   └── services
│       ├── IMovieService.ts
│       ├── ImdbService.ts
│       ├── NetflixService.ts
│       ├── PrimeScrapper.js
│       └── PrimeService.ts
├── music
│   ├── models
│   │   ├── albumModel.ts
│   │   ├── albumTypesMapper.ts
│   │   └── dto
│   │       ├── AlbumDetails.ts
│   │       ├── AlbumInfo.ts
│   │       ├── AlbumUrl.ts
│   │       ├── PlaylistDetails.ts
│   │       ├── PlaylistInfo.ts
│   │       └── PlaylistTrack.ts
│   ├── musicController.ts
│   ├── musicServiceManager.ts
│   └── services
│       ├── DeezerService.ts
│       ├── IMusicService.ts
│       ├── SpotifyService.ts
│       └── YoutubeMusicService.ts
├── tv
│   ├── models
│   │   ├── dto
│   │   │   ├── TvService.ts
│   │   │   └── TvShowInfo.ts
│   │   ├── showModel.ts
│   │   ├── showTypesMapper.ts
│   │   └── tvMetaModel.ts
│   ├── services
│   │   ├── ITvService.ts
│   │   ├── ImdbService.ts
│   │   ├── NetflixService.ts
│   │   └── PrimeService.ts
│   ├── tvController.ts
│   └── tvServiceManager.ts
```

```
├── user
│   ├── passport.ts
│   ├── userController.ts
│   ├── userModel.ts
└── utils
    └── arrayUtils.ts
```

6.3.2 API

Interfejs API aplikacji serwerowej implementuje szereg metod zestawionych w tabeli 3. Każda z akcji, która nie jest bezpośrednio związana z kontem użytkownika ma charakter publiczny i może zostać wykorzystana przez systemy zewnętrzne. Autoryzacja użytkownika wymaga pozyskania tokenu JWT (z ang. *JSON Web Token*) za pomocą metody /user/login. Każde z zapytań HTTP, które wymaga autoryzacji (w tabeli 3 oznaczone pogrubieniem) powinno zawierać odpowiedni nagłówek zawierający token.

Tabela 3. Metody interfejsu API aplikacji *back-end*

Metoda	Adres	Opis
POST	/user/login	Metoda wywoływana w celu uzyskania tokenu JWT autoryzowanego użytkownika.
POST	/user/register	Tworzy nowe konto użytkownika.
GET	**/user/current**	Na podstawie tokenu JWT zwraca podstawowe informacje na temat użytkownika.
GET	/music/new	Zwraca listę nowych albumów dostępnych w serwisach strumieniowych.
GET	/music/album/info	Zwraca metadane albumu.
GET	/music/playlist/popular	Zwraca listę popularnych playlist w serwisach strumieniowych.
GET	**/music/playlist/user**	Zwraca listę playlist zapisanych przez użytkownika w serwisach strumieniowych.
GET	/music/search/album	Zwraca listę albumów spełniających zapytanie.
GET	/music/search/playlist	Zwraca listę playlist spełniających zapytanie.
GET	**/music/:service/add**	Zwraca adres URL do uwierzytelnienia konta użytkownika w serwisie strumieniowym.
POST	**/music/:service/save**	Metoda służy do zapisu kodu uwierzytelnienia pochodzącego z serwisu strumieniowego.
POST	**/music/:service/remove**	Usuwa dostęp do konta użytkownika w serwisie strumieniowym.
GET	/movie/popular	Zwraca listę popularnych filmów w serwisach strumieniowych.
GET	/movie/info	Zwraca metadane filmu.
GET	/movie/new	Zwraca listę nowych filmów dostępnych w serwisach strumieniowych.
GET	/movie/search	Zwraca listę filmów spełniających zapytanie.

POST	**/movie/watchlist /add**	Dodaje film do listy „do obejrzenia” użytkownika.
POST	**/movie/watchlist /remove**	Usuwa film z listy „do obejrzenia” użytkownika.
GET	**/movie/watchlist /get**	Zwraca listę filmów „do obejrzenia” użytkownika.
GET	/tv/popular	Zwraca listę popularnych seriali w serwisach strumieniowych.
GET	/tv/info	Zwraca metadane seriali.
GET	/tv/new	Zwraca listę nowych seriali dostępnych w serwisach strumieniowych.
GET	/tv/search	Zwraca listę seriali spełniających zapytanie.
POST	**/tv/watchlist/add**	Dodaje serial do listy „do obejrzenia” użytkownika.
POST	**/tv/watchlist/remove**	Usuwa serial z listy „do obejrzenia” użytkownika.
GET	**/tv/watchlist /get**	Zwraca listę seriali „do obejrzenia” użytkownika.

6.3.3 Moduł user

Moduł user odpowiada za obsługę funkcjonalności związanych bezpośrednio z kontem użytkownika. Większość udostępnianych w jego ramach akcji wymaga autoryzacji, której dokonać można poprzez metodę /user/login. Do implementacji funkcji uwierzytelniania wykorzystano bibliotekę passport.js. Konfiguracja przedstawiona została na listingu 10.

Listing 10. Konfiguracja biblioteki passport.js

```
const JWTstrategy = passportJwt.Strategy;
const ExtractJWT = passportJwt.ExtractJwt;

passport.use('jwt',
    new JWTstrategy(
        {
            secretOrKey: process.env.JWT_SECRET,
            jwtFromRequest: ExtractJWT.fromUrlQueryParameter('token')
        },
        async (token, done) => {
            try {
                return done(null, token.user);
            } catch (error) {
                done(error);
            }
        }
    )
);

passport.serializeUser((user, done) => {
    done(null, user._id);
});
passport.deserializeUser(async (id,done) => done(null, await
UserModel.findById(id)));

export {passport};
```

Dane na temat użytkownika przechowywane są w bazie danych zgodnie ze schematem zamieszczonym na listingu 11. Ze względów bezpieczeństwa hasło przechowywane jest w formie zaszyfrowanej.

Listing 11. Schemat obiektów User w bazie danych

```
const UserSchema = new Schema({
    email: String,
    hash: String,
    salt: String,
    spotify: {
        token: String,
        refreshToken: String,
        expireDate: String
    },
    youtubemusic: {
        token: String,
        refreshToken: String,
        expireDate: String
    },
    deezer: {
        token: String,
        refreshToken: String,
        expireDate: String
    },
    movies: Array,
    shows: Array
}) as PassportLocalSchema;
```

6.3.4 Moduł music

Moduł ten z założenia powinien zapewniać dostęp do danych pochodzących z różnych serwisów strumieniowych. W tym celu skorzystano z polimorfizmu typów. Sercem domeny jest komponent `musicServiceManager`, w ramach którego rejestrowane są obiekty implementujące interfejs `IMusicService` (patrz listing 12).

Listing 12. Interfejs IMusicService

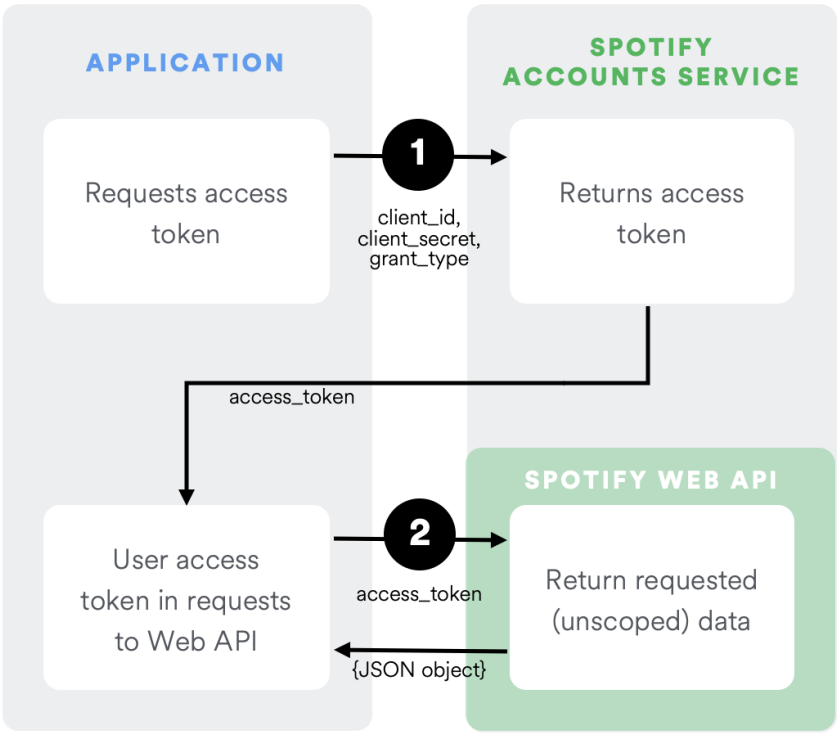
```
export interface IMusicService {
    getNewReleases: (count: number) => Promise<AlbumUri[]>;
    getAlbumDetails: (artist: string, name: string) => Promise<AlbumDetails>;
    getUserPlaylists: (userId: any) => Promise<PlaylistInfo[]>;
    getPlaylist: (id: string, userId: any) => Promise<PlaylistDetails>;
    createAuthorizeURL: () => string;
    removeAccount: (userId: any) => void;
    getAndSaveToken: (authorizationCode: string, userId: any) => void;
    getPopularPlaylists: () => Promise<PlaylistInfo[]>;
    searchAlbum: (query: string, count: number) => Promise<AlbumInfo[]>;
    searchPlaylist: (query: string, count: number) =>
Promise<PlaylistInfo[]>;
}
```

Dla każdego z muzycznych serwisów strumieniowych dostępnych w aplikacji zaimplementowano klasę dziedziczącą po `IMusicService`. Obiekty tego typu stanowią rodzaj adaptera. Potrafią mapować dane w formacie zapewnianym przez zewnętrzne platformy na typy używane wewnątrz projektowanej aplikacji. W przypadku platform Spotify i Deezer usługodawca udostępnia

dobrze udokumentowane API [4] [6], umożliwiające pozyskanie wszystkich informacji potrzebnych w ramach systemu. Oba serwisy strumieniowe pozwalają na dostęp do prywatnych danych użytkownika w zewnętrznych aplikacjach, jeżeli ten wyrazi na to zgodę. Procedura przebiega w następujący sposób:

1. Klient wysyła do API zapytanie z prośbą o stworzenie URL służącego do uwierzytelniania konta. Jako argument przekazany jest adres na jaki użytkownik powinien zostać przekierowany po procesie weryfikacji.
2. Użytkownik zostaje przekierowany na wygenerowany adres. Loguje się i wyraża zgodę na udostępnienie danych do Klienta. Warto zaznaczyć, że dane wymagane do uwierzytelniania podawane są bezpośrednio na stronie w domenie serwisu strumieniowego. Nie ma zatem obawy o ich przechwycenie i kompromitację.
3. Użytkownik zostaje przekierowany na podany w punkcie 1 adres. Jako parametr query w URL podany jest kod uwierzytelniający.
4. Klient sczytuje kod uwierzytelniający i wysyła go do API. W odpowiedzi otrzymuje token JWT, który może posłużyć do autoryzację przyszłych zapytań odnoszących się do danych użytkownika.

Jako ilustrację tego procesu zamieszczono schemat autoryzacji użytkownika w Spotify, zamieszczony na rysunku 29. Warto również zwrócić uwagę, że serwis Spotify udostępnia dedykowane dla platformy Node.js SDK. Z drugiej strony, API oferowane przez platformę Deezer posiada inne zalety. Oferuje ono zacznie więcej metadanych na temat dostępnej muzyki. Przykładowo, z API Deezera można sczytać gatunki muzyczne, w ramach których powstał album. Dodatkowo, zostało ono zaprojektowane zgodnie z zasadami HATEOAS, co w niektórych wypadkach ułatwia korzystanie z niego.



Rysunek 29. Schemat autoryzacji użytkownika w serwisie Spotify. Źródło [28]

Dostęp do danych z YouTube Music wydaje się z pozoru trudniejszy, gdyż twórcy nie udostępniają publicznego API dla tego serwisu. Metodą prób i błędów udało się jednak autorowi pracy opracować sposób, który pozwala ten problem ominąć. Kluczowe okazało się spostrzeżenie na temat

charakteru usługi YouTube Music. Nie posiada ona własnego systemu plików, tylko opiera się na danych zgromadzonych w ramach starszego serwisu YouTube, który posiada publiczne, rozbudowane API. Każdy utwór zamieszczony na YouTube Music jest w rzeczywistości filmem zamieszczonym na YouTube, a album muzyczny jest playlistą. Korzystając z wcześniej wspomnianego API oraz prostej metody web scraping, polegającej na wysyłaniu zapytań HTTP oraz przeszukiwaniu treści odpowiedzi za pomocą wyrażeń regularnych, udało się zaimplementować wszystkie wymagane funkcjonalności. Przykład zastosowania tej techniki przedstawiono na listingu 13. Metoda ta do działania wykorzystuje szczegóły implementacyjne witryny web YouTube Music. Jest to o tyle niekorzystne, że struktura strony może w przyszłości ulec zmianie, co wiązać się będzie z nieprzewidzianymi pracami utrzymaniowymi w ramach projektowanego systemu.

Listing 13. Metoda umożliwiająca odczyt nowych albumów z serwisu YouTube Music

```
async getNewReleases(count: number): Promise<AlbumUri[]> {
    const newReleasesUrl = 'https://music.youtube.com/new_releases/albums';
    const response = await got(newReleasesUrl, {
        headers: {
            'user-agent': USERAGENT
        }
    });

    const rgx = /playlistId\\x22:\\x22OLAK5uy_.{33}\\x22\\x7d\\x7d\\x7d,/g;
    const playlistIds = response.body.match(rgx).slice(0, count).map((phrase)
=> {
        return phrase.slice(19, 60);
    });
    const albums = await this.youtube.playlists.list({id: playlistIds, part:
['snippet']});

    const newReleases = [];

    for (const album of albums.data.items) {
        let artist = "";
        await this.youtube.playlistItems
            .list({playlistId: album.id, part: ['snippet']})
            .then(
                resp => artist =
resp.data.items[0].snippet.videoOwnerChannelTitle,
                err => console.log('Something went wrong: ', err));

        newReleases.push(new AlbumUri(album.snippet.title.slice(8),
artist.slice(0, -8)));
    }

    return newReleases;
}
```

Jak wspomniano we wstępie tego podrozdziału, centralnym elementem modułu music jest komponent `musicServiceManager`. Odpowiada on za złączanie danych pochodzących z różnych źródeł w jeden obiekt, który później przekazywany jest do interfejsu API. Wymaga to rozwiązania problemu identyfikacji tych samych zasobów, pochodzących z różnych serwisów. W przypadku albumów, nie można skorzystać po prostu z identyfikatora, gdyż każda z platform strumieniowych posiada własną, niezależną bazę danych i ich wartości nie będą ze sobą kompatybilne. Istnieje niezależny, stworzony jeszcze na potrzeby katalogowania fizycznych wydawnictw system kodów UPC. Wspierany jest on jednak niestety tylko przez platformę Spotify, przez co również nie można go wykorzystać w sposób

uniwersalny. Ostatecznie, zdecydowano się na identyfikację albumu za pomocą jego nazwy oraz wykonawcy, bez rozróżniania wielkości znaków. Nie jest to rozwiązanie idealne. W bardzo rzadkich przypadkach mechanizm może nie zadziałać prawidłowo. W testach manualnych nie udało się jednak doprowadzić do takiej sytuacji, a rozwiązanie zostało uznane za satysfakcjonujące.

Moduł music zapewnia odpowiednią hermetyzację danych. Na potrzeby interfejsu API przewidziano 4 typy DTO (ang. *Data Transfer Object*): `AlbumInfo, AlbumDetails, PlaylistInfo, PlaylistDetails` przedstawione na listing 14.

Listing 14. Typy DTO dla modułu music

```
export class AlbumInfo {
    name: string;
    artist: string;
    image: string;
    releaseDate: Date;
    genres: string[];
}

export class AlbumDetails {
    name: string;
    artist: string;
    image: string;
    tracks: string[];
    albumUrls: AlbumUrl[];
    releaseDate: Date;
    genres: string[];
}

export class PlaylistInfo {
    name: string;
    image: string;
    id: string;
    service: string;
}

export class PlaylistDetails {
    name: string;
    image: string;
    tracks: PlaylistTrack[];
    url: string;
    service: string;
}
```

W aplikacji zdecydowano się na przechowywanie metadanych na temat albumów w bazie danych. Są to informację z zasady niezmienne, nie ma więc obawy, że ulegną dezaktualizacji. W wyniku tego zabiegu, system powinien działać wydajniej, unikając wielokrotnego wysyłania tych samych zapytań do wielu serwisów strumieniowych. Podejście to powinno również zapewnić większą niezawodność aplikacji. W wypadku awarii serwisu strumieniowego, poprawne dane wciąż powinny być dostępne w ramach platformy. Schemat obiektów reprezentujących album w bazie danych widoczny jest na listingu 15.

Listing 15. Schemat obiektów Album w bazie danych

```
const albumSchema = new Schema<Album>({
    name: String,
    artist: String,
    image: String,
    tracks: Array,
    albumUrls: Array,
    releaseDate: Date,
    name_lower: String,
    artist_lower: String,
    genres: Array
});
```

Na potrzeby niniejszej pracy dokonano również analizy wstępnej innych muzycznych serwisów strumieniowych: Apple Music oraz Tidal. Pierwsza z tych platform posiada interfejs API. Jest on jednak dostępny wyłącznie po wykupieniu pakietu deweloperski Apple. W ramach usługi Tidal nie zostało udostępnione API, co znacznie utrudnia zczytawanie z niej informacji. Cześć danych można uzyskać przy wykorzystaniu web scrapingu, jednak niektóre funkcjonalności, jak np. odczyt playlist użytkownika, mogą okazać się niemożliwe do zaimplementowania dla tej platformy.

Na koniec tego podrozdziału, w ramach podsumowania, zdecydowano się zestawić ze sobą niektóre cechy serwisów Spotify, YouTube Music i Deezer, istotne z punktu widzenia problematyki niniejszej pracy. Wyniki przedstawia tabela 4.

Tabela 4. Porównanie serwisów Spotify, YouTube Music i Deezer

	Spotify	YouTube Music	Deezer
W pełni dostępne, publiczne API	✓	✗	✓
SDK	✓	✓	✗
Kategoryzacja albumów ze względu na gatunek	✗	✗	✓
Klasyfikacja UPC	✓	✗	✗
HATEOAS	✗	✗	✓

6.3.5 Moduł movie

Implementacja modułu movie może wydawać się z pozoru podobna do wcześniej omawianej domeny music. Dostępne do danych w serwisach strumieniowych wideo jest jednak trudniejszy, gdyż nie posiadają one publicznego interfejsu API. Ze względu na ten fakt, na potrzeby wymagań funkcjonalnych tego modułu, wykorzystano inne, nieszablonowe rozwiązania.

Analogicznie jak w przypadku modułu music, centralny element stanowi `movieServiceManager`. Z jego poziomu rejestrowane są serwisy odpowiedzialne za obsługę danej platformy strumieniowej, dziedziczące po interfejsie `IMovieService` (patrz listing 16). Do identyfikacji filmów wykorzystano system IMDb, dostępny za pośrednictwem interfejsu API imdb8. W związku z tym, powstało zapotrzebowanie na nową klasę, odpowiadającą za obsługę tego serwisu, której publiczne metody zestawiono na listingu 17. Za jego pomocą sczytywane są również metadane na temat filmu: czas trwania, ocena, opis oraz informacje o popularnych w danym momencie pozycjach.

Listing 16. Interfejs IMovieService

```
export interface IMovieService {
    getMovieUrl: (id: string) => Promise<string>;
    getNewMovies: () => Promise<string[]>;
}
```

Listing 17. Klasa ImdbService dla modułu movie

```
export class ImdbService {

    async getPopularMovies(): Promise<string[]> {
      …
    }

    async getMovieInfo(id: string): Promise<MovieInfo> {
      …
    }

    async searchMovie(query: string): Promise<string[]> {
      …
    }
}
```

W celu uzyskania informacji na temat dostępności filmów w serwisie Netflix wykorzystano usługę uNoGS *(unofficial Netflix online Global Search*) [29], dostępną za pośrednictwem API. Przykład użycia zamieszczono na listingu 18.

Listing 18. Wykorzystanie API unogs do uzyskania informacji na temat nowych filmów dostępnych na Netflix

```
async getNewMovies(): Promise<string[]> {
    const options = {
        method: 'GET' as Method,
        url: 'https://unogs-unogs-v1.p.rapidapi.com/aaapi.cgi',
        params:  {q: 'get:new14:PL', p: '1', t: 'ns', st: 'adv'},
        headers: {
            'x-rapidapi-host': 'unogs-unogs-v1.p.rapidapi.com',
            'x-rapidapi-key': process.env.RAPIDAPI_KEY
        }
    };

    return Axios.request(options)
        .then(async resp =>
        {
            return resp.data.ITEMS
                .filter((item :any) =>
                  item.type === "movie" && item.imdbid !== "")
                .map((result: any) => result.imdbid)

        })
        .catch(err => {
            console.log(err);
            throw err;
        });
}
```

Ze względu na brak usług oferujących podobne możliwości dla serwisu Amazon Prime, zdecydowano się opracować autorskie rozwiązanie, wykorzystujące metodę web scraping. W tym celu przygotowano skrypt napisany w języku JavaScript, z użyciem biblioteki puppeteer, służącej do symulacji przeglądarki internetowej (więcej szczegółów w podrozdziale 4.10). Dane dostępne na Amazon Prime są niestety całkowicie prywatne, tzn. dostępn do nich można uzyskać dopiero po zalogowaniu. W związku z tym, przed przystąpieniem do ekstrakcji informacji, program musi wprowadzić prawidłowe dane użytkownika. Do eksperymentu posłużyło prywatne konto autora pracy. Skrypt przedstawiono na listingu 19.

Listing 19. Funkcja do sczytywania nowych filmów i seriali w Amazon Prime

```
const puppeteer = require('puppeteer')

export async function scrape(type) {
  let newReleasesUrl = "";
  if (type === "movie") {
    newReleasesUrl = "https://www.primevideo.com/search/...";
  }
  if (type === "tv") {
    newReleasesUrl = "https://www.primevideo.com/search/...";
  }

  const browser = await puppeteer.launch({});
  const page = await browser.newPage();

  await page.goto('https://www.amazon.com/ap/signin?clientContext=…');
  await page.type('#ap_email', process.env.PRIME_USER);
  await page.type('#ap_password', process.env.PRIME_PASS);
  await page.$eval('input[id=signInSubmit]', el => el.click());
  await page.waitForNavigation();
  await page.goto(newReleasesUrl);

  const hrefs = await page.evaluate(
    () => Array.from(
      document.querySelectorAll('a[href]'),
      a => a.getAttribute('href')
    )
  );

  let rgx = /\/detail\/.{26}\/ref/g;
  const uniqHrefs = [...new Set(hrefs.join().match(rgx))];
  let titles = [];

  for (const href of uniqHrefs.slice(0, 15)) {
    await page.goto('https://www.primevideo.com/' + href);
    if (type === "movie") {
      await page.waitForSelector('[data-automation-id="title"]');
      titles.push(await page.evaluate(() => document.querySelector('[data-
automation-id="title"]').textContent));
    }
    if (type === "tv") {
      const html = await page.content();
      rgx = /"parentTitle":"([^"]+)"/g;
      let matched = html.match(rgx)[0];
      titles.push(matched.slice(15,-1));
    }
  }

  await browser.close();
  const uniqTitles = [...new Set(titles)];
  return uniqTitles.slice(0, 15);
}
```

Wykorzystanie biblioteki puppeteer niesie ze sobą pewne konsekwencję. Narzędzie to uruchamia przeglądarkę internetową Chromium, co jest operacją stosunkowo długą i wymagającą wielu zasobów. W celu uniknięcia sytuacji, w której nowa sesja przeglądarki jest tworzona w odpowiedzi na każde

żądanie HTTP, zdecydowano się na przechowywanie danych na temat nowych filmów w bazie danych, zgodnie ze schematem z listingu 20. Obiekty te posiadają listę identyfikatorów nowych filmów oraz datę utworzenia. Za każdym razem, gdy odczytywana jest informacja z `MovieMeta`, system sprawdza czy zapisana data jest aktualna. Jeżeli nie, tworzy nowy obiekt. Rozwiązanie to sprawia, że informacje dostępne w aplikacji są zawsze nie starsze niż 24 godziny.

Listing 20. Schemat obiektów MovieMeta w bazie danych

```
const movieMetaSchema = new Schema<MovieMeta>({
    new: Array,
    date: Date
});
```

Ze względu na trwały charakter metadanych na temat filmów, zdecydowano się również na przechowywanie ich w bazie danych, zgodnie ze schematem z listingu 21. Formatem wyjściowym do API z modułu movies jest typ DTO zamieszczonych na listingu 22.

Listing 21. Schemat obiektów Movie w bazie danych

```
const movieSchema = new Schema<Movie>({
    id: String,
    name: String,
    image: String,
    time: Number,
    year: Number,
    rating: Number,
    plot: String,
    services: Array,
    genres: Array
});
```

Listing 22. Typy DTO dla modułu movie

```
export class MovieInfo {
    id: string;
    name: string;
    image: string;
    time: number;
    year: number;
    rating: number;
    plot: string;
    services: MovieService[];
    genres: string[];
}

export class MovieService {
    url: string;
    service: string;
}
```

6.3.6 Moduł tv

Moduł tv operuje na danych z tych samych serwisów strumieniowych co domena movie. Z zasady więc struktura i sposób implementacji są dla tych części podobne. W ramach komponentu

`tvServiceManager` rejestrowane są serwisy odpowiedzialne za obsługę platform strumieniowych implementujące interfejs `ITvService` (listing 23) oraz obiekt klasy `ImdbService` (listing 24).

Listing 23. Interfejs ITvService

```
export interface ITvService {
    getNewShows: () => Promise<string[]>;
    getShowUrl: (id: string) => Promise<string>;
}
```

Listing 24. Klasa ImdbService dla modułu tv

```
export class ImdbService {
    async getShowInfo(id: string): Promise<TvShowInfo> {
        …
    }

    async getPopularShows(): Promise<string[]> {
        …
    }

    async searchShow(query: string): Promise<string[]> {
        …
    }
}
```

Dostęp do informacji na temat seriali w usłudze Netflix uzyskiwany jest za pośrednictwem API uNoGS. Wykorzystano również skrypt do zaciągania danych z Amazon Prime, który posiada możliwość konfiguracji w zależności od pożądanego typu medium (filmy lub seriale). Przy zapisie do bazy danych kierowano się taką samą koncepcją, jak w przypadku modułu movie. Schemat obiektu typu meta zamieszczony został na listingu 25. Informacje na temat serialu przechowywane są w formacie widocznym na listingu 26. Na potrzeby obsługi interfejsu API przewidziano również typ DTO `TvShowInfo` (listing 27).

Listing 25. Schemat obiektów TvMeta w bazie danych

```
const tvMetaSchema = new Schema<TvMeta>({
    new: Array,
    date: Date
});
```

Listing 26. Schemat obiektów Show w bazie danych

```
const showSchema = new Schema<Show>({
    id: String,
    name: String,
    image: String,
    year: Number,
    rating: Number,
    plot: String,
    services: Array,
    genres: Array
});
```

Listing 27. Typy DTO dla modułu tv

```
export class TvShowInfo {
    id: string;
    name: string;
    image: string;
    year: number;
    rating: number;
    plot: string;
    services: TvService[];
    genres: string[];}

export class TvService {
    url: string;
    service: string;
}
```

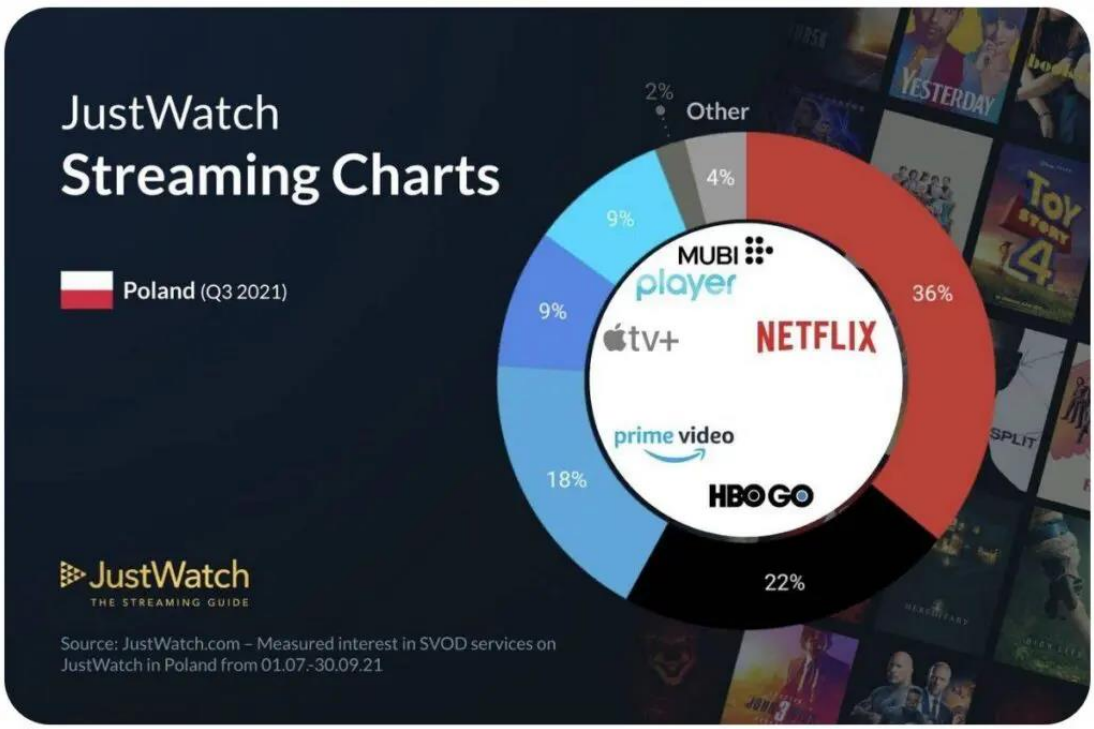
7. Przegląd możliwości rozwoju projektu

Opracowany system stanowi jedynie prototyp. Jego możliwości wymagają rozszerzenia na wielu płaszczyznach, aby móc konkurować z podobnymi rozwiązaniami komercyjnymi. W tym rozdziale przedstawiono kilka pomysłów na rozwój projektu. Wśród najważniejszych z nich wymienić można integrację z większą ilością serwisów strumieniowych audio i wideo oraz obsługę innych typów mediów. Na koniec przedstawiono również propozycje związane z infrastrukturą systemu: implementację klienta mobilnego oraz podział aplikacji *back-end* na mikroserwisy.

7.1. Serwisy strumieniowe audio i wideo

W obecnym kształcie aplikacja wspiera jedynie niektóre z popularnych serwisów strumieniowych, zarówno muzycznych jak i wideo. W celu zapewnienia większej uniwersalności platformy, należy rozszerzyć jej funkcjonalność o wsparcie dla pozostałych streamingów. W kategorii muzycznej są to platformy Apple Music oraz Tidal. Analiza wymagań integracji tych serwisów została przedstawiona w rozdziale 6.3.4

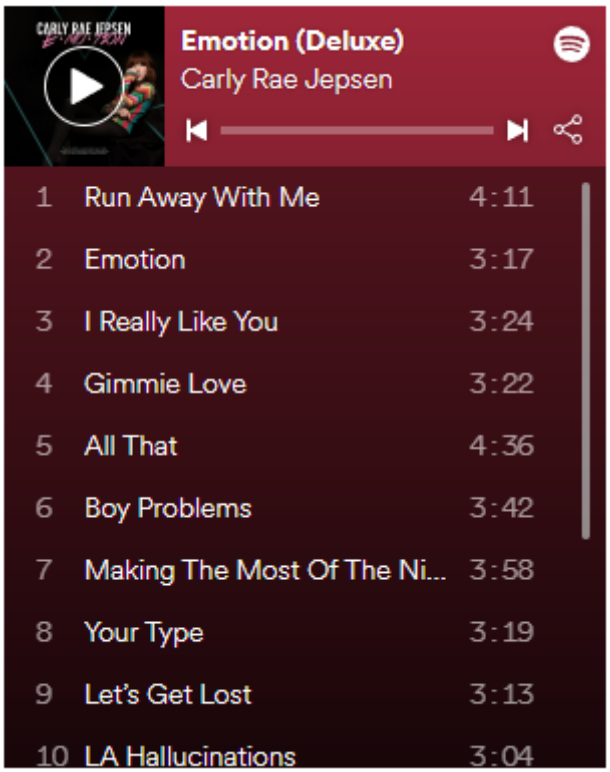
W wypadku platform strumieniowych wideo sytuacja wydaje się być bardziej skomplikowana, gdyż nie są dla nich dostępne publiczne interfejsy API. Opierając się o statystyki pochodzące z serwisu JustWatch [12] przedstawione na rysunku 30, do najpopularniejszych platform strumieniowych wideo w Polsce zaliczyć można Netflix, HBO GO, Amazon Prime, Apple tv+, Player oraz Mubi, które stanowią razem 96% rynku. W planach rozwojowych projektu warto uwzględnić integrację z tymi serwisami. Zaznaczyć należy, że jest to zadanie czasochłonne, gdyż dostęp do danych pochodzących z tych usług wymaga uwierzytelnienia za pomocą konta użytkownika, co znacznie komplikuje proces ich pozyskiwania.



Rysunek 30**.** Popularność serwisów strumieniowych wideo w 3 kwartale 2021 roku w Polsce. Źródło: [12]

7.2. Odtwarzanie multimediów

Serwis strumieniowy Spotify udostępnia bibliotekę Web Playback SDK przeznaczoną dla języka JavaScript. Umożliwia ona odtwarzanie albumów i playlist z poziomu klienta przeglądarki. Podobne biblioteki udostępniają również serwisy YouTube (a co za tym idzie również YouTube Music) oraz Apple Music. Interesującą funkcją wzbogacającą możliwości systemu byłaby implementacja możliwości odtwarzania utworów muzycznych bezpośrednio z jego poziomu. Co ciekawe, biblioteka Web Playback SDK dostarcza również komponenty React, których wykorzystanie powinno znacząco ułatwić integrację z projektowaną platformą. Przykład zastosowania Web Playback SDK zaprezentowano na rysunku 31.



Rysunek 31. Wykorzystanie biblioteki Web Playback SDK do wygenerowania odtwarzacza Spotify. Źródło: [4]

7.3. Inne typy mediów

W prototypowanej aplikacji przewidziano obsługę muzyki, filmów oraz seriali. Nie są to jedyne typu multimediów dostępne za pośrednictwem streamingu. W dłuższej perspektywie czasu projektowany system może zostać rozszerzony o wsparcie dla innych typów mediów. W niniejszym podrozdziale postanowiono po krótce scharakteryzowano najciekawsze z nich. Warto zaznaczyć, że dzięki zastosowaniu wzorca architektury heksagonalnej, rozszerzenie funkcjonalności o nowe rodzaje źródeł nie powinno wymagać modyfikacji dotychczasowo napisanych modułów.

7.3.1 Podcasty

Obok muzyki, to jedna z najpopularniejszych form multimediów w formacie audio, dostępnych w ramach serwisów strumieniowych. Podcasty znaleźć można są m.in. w serwisie Spotify. Dane na ich temat dostępne są w ramach tego samego interfejsu API, który został wykorzystany w module muzycznym. Do innych popularnych katalogów podcastów, których obsługę warto rozpatrzeć zaliczyć można usługi Apple Podcasts oraz Google Podcasts. Oba serwisy posiadają dedykowane interfejsy API.

7.3.2 Audiobooki

Innym popularnym medium w formacie audio są audiobooki. Istnieje wiele serwisów strumieniowych oferujących tego typu treści w Polsce, np. Storytel czy Audioteka. Oba nie posiadają publicznego API, jednak dostęp do danych nie wymaga wcześniejszej autoryzacji, a ich ekstrakcja wydaje się stosunkowo prostym zadaniem.

7.3.3 Gry wideo

Zagadnienie strumieniowania gier wideo jest o tyle złożone, że wymaga dwukierunkowej komunikacji pomiędzy klientem a serwerem. Z jednej strony transmitowany jest obraz wideo, z drugiej instrukcje w postaci naciśniętych klawiszy oraz ruchów myszką służące do sterowania grą. Oczywiście, ze względu na komunikację przez sieć web, w procesie tym wystąpić mogą opóźnienia, co znacząco utrudnia rozgrywkę. W ostatnich latach szybkość i stabilność Internetu wzrosła do poziomu, który umożliwia komfortowe korzystanie z tego typu usług. Na rynku pojawiło się wiele platform do strumieniowania gier wideo, jak np. GeForce Now, Google Stadia, PlayStation Now, Xbox Cloud Gaming czy Amazon Luna. Choć stanowią one nadal niszę, ze względu na braki na rynku półprzewodników oraz dynamiczny rozwój sieci 5g, niektórzy analitycy prognozują znaczny wzrost tego segmentu [30]. Przypuszczenia te legitymizować może duże zainteresowanie gigantów cyfrowych tego typu rozwiązaniami. W ramach rozwoju projektu, warto wyjść tym przewidywaniom naprzeciw i zaplanować integrację serwisów strumieniowych z grami wideo na stosunkowo wczesnym etapie.

7.4. Pozostałe

System opracowany na potrzebę niniejszej pracy dostępny jest jedynie za pośrednictwem przeglądarki internetowej. Strona została zaprojektowana w sposób responsywny, a co za tym idzie jest dostosowana również do smartfonów i tabletów. Nie jest to jednak najwygodniejsza forma przeglądania treści na tego typu urządzeniach. W ramach rozwoju projektu przewiduje się implementację dedykowanych aplikacji mobilnych na urządzenia z system operacyjnym Android i iOS. Kolejnym zagadnieniem, nad którym warto się na chwilę pochylić jest podział aplikacji serwerowej na działające całkowicie niezależnie mikroserwisy. Przy implementacji kolejnych funkcjonalności kodu źródłowego może rozrosnąć się do rozmiarów, przy których utrzymywanie go w formie monolitu może okazać się nie praktyczne, szczególnie jeżeli nad jego rozwojem pracować będzie wiele zespołów deweloperskich. Warto zaznaczyć, że zastosowany wzorzec architektury heksagonalnej znacznie ułatwia przeprowadzenie tego typu transformacji.

8. Podsumowanie

Ostatni rozdział podsumowuje działania podjęte w ramach pracy oraz odnosi się do pewnych założeń poczynionych na jej wstępie. W jego ramach podjęto próbę sformułowania uniwersalnych wniosków dotyczących pozyskiwania danych z serwisów multimedialnych w oparciu o przeprowadzaną analizę 5 z nich: Spotify, Deezer, YouTube Music, Netflix ora Amazon Prime.

W ramach podrozdziału 2.3 zaproponowano klasyfikację platform strumieniowych ze względu na poziom ekspozycji danych, która prezentuje się w następujący sposób:

1. **Dostęp publiczny za pomocą API** – serwisy: Spotify, Deezer.
2. **Dostęp publiczny, bez API –** serwis YouTube Music.
3. **Dostęp prywatne –** serwisy: Netflix, Amazon Prime.

Postawiono również tezę, w ramach której przynależność serwisu, do jednej z tych klas stanowi główny czynnik warunkujący sposób ekstrakcji danych z platformy, jak również stopień jego skomplikowania. Analiza przeprowadzona w ramach tej pracy pokazuje, że teza ta jest uzasadniona. Można więc sformułować konkluzję, która mówi, że poziom trudności w uzyskaniu informacji na temat multimediów dostępnych w ramach serwisu zależy bezpośrednio od poziomu ekspozycji danych, który ustalany jest przez jego właściciela. Niektóre możliwości, jak np. integracja konta użytkownika, dostępna jest jedynie dla usług z publicznym interfejsem API, po pomyślnym przejściu procesu autentykacji.

Pozyskanie informacji z platform, które udostępniają API okazały się zadaniem stosunkowo łatwym. Interfejsy zapewniają kompleksowe dane na temat multimediów znajdujących się w serwisach. Ponadto, każde z analizowanych API posiada możliwość odczytywania informacji na temat prywatnych danych użytkownika, takich jak np. zapisane przez niego playlisty.

Dane pochodzące z serwisów o dostępie publicznym bez API są zdecydowanie trudniejsze do pozyskania. Proces ten wymaga wykorzystania tzw. web scrapingu. Dla serwisu YouTube Music udało się opracować metodę polegającą na wysłaniu zapytania HTTP, a następnie przeszukiwania zawartości odpowiedzi przy pomocy wyrażeń regularnych w celu ekstrakcji znaczących danych z punktu widzenia systemu.

Największym wyzwaniem okazała się ekstrakcja informacji z serwisów, których zawartość jest dostępna wyłącznie dla zalogowanych użytkowników. Na potrzeby pozyskiwania danych z Amazon Prime opracowano skrypt symulujący przeglądarkę internetową. Automat wypełnia formularze logowania, a następnie sczytuje dane korzystając z selektorów języka HTML. Rozwiązanie to nie jest idealne. Zużywa wiele zasobów oraz jest podatne na zmiany w strukturze strony. Niemniej, dla niektórych serwisów strumieniowych stanowi ono jedyny sposób na automatyczne pozyskiwanie informacji.

Choć w ramach pracy dokonano analizy wielu popularnych platform multimedialnych, nie wyczerpuje ona zagadnienia agregacji dostępnych na nich danych. Nie istnieje żaden standard, na podstawie którego projektowane są serwisy strumieniowe. W związku z tym, usługi tego typu mają unikalny charakter, a przedstawione w tej pracy metody mogą okazać się niewystarczające dla niektórych z nich.

Bibliografia

[1]. Digital 2021: Global Overview Report. https://datareportal.com/reports/digital-2021-global-overview-report, Styczeń 2022

[2]. Nielsen provides topline U.S. web data for March 2010. https://www.nielsen.com/us/en/insights/article/2010/nielsen-provides-topline-u-s-web-data-for-march-2010, Styczeń 2022

[3]. Papadopoulos, Panagiotis i Kourtellis, Nicolas i Markatos, Evangelos. (2018). Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask.

[4]. Spotify Web API documentation. https://developer.spotify.com/documentation/web-api, Styczeń 2022

[5]. YouTube Data API documentation. https://developers.google.com/youtube/v3, Styczeń 2022

[6]. Deezer API documentation. https://developers.deezer.com/api, Styczeń 2022

[7]. Definicja serwisu strumieniowego w Cambridge Dictionary. https://dictionary.cambridge.org/pl/dictionary/english/streaming-service, Styczeń 2022

[8]. Jacobson, Brail, Woods (2012), APIs: A Strategy Guide: Creating Channels with Application Programming Interfaces, ISBN: 9781449308926

[9]. Serwis RouteNote. https://www.routenote.com, Styczeń 2022

[10]. Serwis MusicBrainz. https://musicbrainz.org, Styczeń 2022

[11]. Serwis Upflix. https://upflix.pl, Styczeń 2022

[12]. Serwis JustWatch https://www.justwatch.com, Styczeń 2022

[13]. Przeodnik po Material Design. https://material.io/design, Styczeń 2022

[14]. Mew, Learning Material Design (2015), ISBN: 9781785289811

[15]. User stories with examples and a template. https://www.atlassian.com/agile/project-management/user-stories, Styczeń 2022

[16]. Ruparelia (2016), Cloud Computing, ISBN: 9780262529099

[17]. MERN Stack Explained. https://www.mongodb.com/mern-stack, Styczeń 2022

[18]. Test wydajnościowy frameworków webowych. https://www.techempower.com/benchmarks, Styczeń 2022

[19]. Netscape and Sun announce JavaScript, https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html, Wrzesień 2007 (za pośrednictwem archive.org)

[20]. Kamiński, Paweł. React. Wstęp do programowania, Helion 2022, ISBN: 9788328389809

[21]. Dokumentacja Material UI. https://mui.com, Styczeń 2022

[22]. About Node.js, and why you should add Node.js to your skill set? http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html, Styczeń 2022

[23]. Wprowadzenie do Express/Node. https://developer.mozilla.org/pl/docs/Learn/Server-side/Express_Nodejs/Introduction, Styczeń 2022

[24]. MongoDB. https://www.mongodb.com, Styczeń 2022

[25]. Ready for changes with Hexagonal Architecture. https://netflixtechblog.com/ready-for-changes-with-hexagonal-architecture-b315ec967749, Styczeń 2022

[26]. Hexagonal Architecture by example - a hands-on introduction. https://blog.allegro.tech/2020/05/hexagonal-architecture-by-example.html, Styczeń 2022

[27]. Introduction to JSON Web Tokens. https://jwt.io/introduction, Styczeń 2022

[28]. Client Credentials Flow. https://developer.spotify.com/documentation/general/guides/authorization/client-credentials/, Styczeń 2022

[29]. Netflix Global Search, https://unogs.com, Styczeń 2022

[30]. Cloud gaming przyszłością rozrywki. Pomaga pandemia i 5G. https://cyfrowa.rp.pl/gry-e-sport/art19265951-cloud-gaming-przyszloscia-rozrywki-pomaga-pandemia-i-5g, Styczeń 2022

[31]. Cisco Visual Networking Index: Forecast and Trends 2017-2022. https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf, Styczeń 2022

[32]. Mitchell, Ryan. Web Scraping with Python, O'Reilly Media 2015, ISBN: 9781491910290

Spis tabel

Spis rysunków

Spis listingów