|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | **Wydział Informatyki i Zarządzania**  kierunek studiów: informatyka  Praca dyplomowa – inżynierska  ***Looking Glass* – Aplikacja do spersonalizowanej selekcji programów telewizyjnych**  Przemysław Zender  słowa kluczowe:  1 linia  2 linia  3 linia  krótkie streszczenie:  1 linia  2 linia  3 linia  4 linia  5 linia  6 linia   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | opiekun pracy  dyplomowej | .................................................. | | ....................... | | ....................... | | | *Tytuł/stopień naukowy/imię i nazwisko* | | *ocena* | | *podpis* | | | Ostateczna ocena za pracę dyplomową | | | | | | | | Przewodniczący Komisji egzaminu dyplomowego | | ..................................................  *Tytuł/stopień naukowy/imię i nazwisko* | | ....................... | | ....................... | | *ocena* | | *podpis* |   *Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**   1. *kategorii A (akta wieczyste)* 2. *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*   *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław  2018 |

Spis treści

[1. Wstęp 3](#_Toc25614146)

[1.1. Uzasadnienie tematu 3](#_Toc25614147)

[1.2. Cel pracy 3](#_Toc25614148)

[1.3. Zakres prac 3](#_Toc25614149)

[2. Przegląd istniejących rozwiązań 5](#_Toc25614150)

[2.1. Standardowe przewodniki programowe 5](#_Toc25614151)

[2.2. Silniki rekomendacji 6](#_Toc25614152)

[2.3. Serwisy streamingowe 7](#_Toc25614153)

[2.4. Filmweb 8](#_Toc25614154)

[2.5. Tabela porównawcza 9](#_Toc25614155)

[2.6. Wnioski 10](#_Toc25614156)

[3. Założenia projektowe 11](#_Toc25614157)

[3.1. Opis problemu 11](#_Toc25614158)

[3.2. Wizja rozwiązania 11](#_Toc25614159)

[3.3. Źródła danych 14](#_Toc25614160)

[3.4. Wymagania funkcjonalne 14](#_Toc25614161)

[4. Projekt aplikacji 15](#_Toc25614162)

[4.1. Architektura aplikacji 15](#_Toc25614163)

[4.2. Zastosowane technologie 15](#_Toc25614164)

[4.3. Projekt bazy danych 21](#_Toc25614165)

[4.4. Komunikacja między warstwami 23](#_Toc25614166)

[5. Implementacja 24](#_Toc25614167)

[5.1. Serwer 24](#_Toc25614168)

[5.2. Baza danych 33](#_Toc25614169)

[5.3. Klient 34](#_Toc25614170)

[6. Testy 40](#_Toc25614171)

[6.1. Jednostkowe serwera – xUnit 40](#_Toc25614172)

[6.2. Jednostkowe klienta – Jasmine i Karma 42](#_Toc25614173)

[6.3. Automatyczne end-to-end – Protractor 45](#_Toc25614174)

[7. Instalacja 46](#_Toc25614175)

[7.1. Założenia 46](#_Toc25614176)

[7.2. Zastosowanie 47](#_Toc25614177)

[7.3. Komunikacja 49](#_Toc25614178)

[8. Podsumowanie 50](#_Toc25614179)

[9. Bibliografia 51](#_Toc25614180)

# Wstęp

## Uzasadnienie tematu

Mimo rosnącej konkurencji ze strony internetu i serwisów streamingowych (jak Netflix lub Hulu), telewizja wciąż jest popularną formą rozrywki. Najpopularniejsze programy oglądają miliony ludzi, a gust każdego z nich jest inny. Ogromna różnorodność dostępnych opcji (ponad 400 kanałów na terenie Polski) sprawia, że znalezienie interesującego nas programu wymaga przejrzenia wielu stron pełnych spisów nadawanych audycji. W efekcie często zamiast zrelaksować się przy ulubionym programie, przełączamy po kolei kanały, licząc że w końcu trafimy na coś ciekawego.

## Cel pracy

Celem pracy jest ułatwienie użytkownikom oglądania interesujących ich programów przez opracowanie aplikacji pozwalającej na 3 kluczowe czynności:

* określenie i zapisanie swoich preferencji,
* pobranie informacji o nadawanych w najbliższym czasie programach
* rekomendacja dotycząca pozycji, które mogą uznać za interesujące na podstawie preferencji i aktualnego programu.

## Zakres prac

W przewidzianym zakresie projektu mieści się zbieranie danych dotyczących nadawanych programów oraz preferencji użytkowników, jak również wykorzystanie zebranych danych do przekazania użytkownikom rekomendacji.

Dane dotyczące programów pochodzą z dostępnych w internecie zasobów zebranych z użyciem programu do web scrapingu. Zebrane dane są później poddane analizie w celu wyznaczenia cech charakterystycznych poszczególnych programów i określenia podobieństwa między nimi. Szczegółowy opis wyróżnianych cech znajduje się w dalszej części pracy. Dodatkowo w ramach programu telewizyjnego dostępne są opisy poszczególnych pozycji – na podstawie zawartych w nich słów kluczowych można określić tematyczne podobieństwo między programami.

Dane dotyczące swoich preferencji użytkownik wprowadza sam, oznaczając poszczególne seriale, pojedyncze programy, kanały telewizyjne oraz interesujące go cechy charakterystyczne (aktorzy, drużyny, kategorie itd.) jako ulubione. Ponadto, ponieważ nawet najlepsza rekomendacja nie ma sensu, jeżeli użytkownik w danym czasie nie ma dostępu do telewizora, bardzo ważne są godziny w jakich użytkownik może i chce spędzić swój czas oglądając rekomendowane programy. Na podstawie tak zdefiniowanego profilu użytkownika można zaoferować rekomendacje zgodnych z nim programów.

O oferowanych rekomendacjach możemy informować uzytkownika na dwa sposoby. Pierwszym jest standardowy interfejs aplikacji, który jednak wymaga, żeby użytkownik w danej chwili aktywnie z niej korzystał. Drugim sposobem są powiadomienia *push* przekazywane użytkownikowi w czasie gdy aplikacja działa w tle. Najlepiej działa to w przypadku aplikacji mobilnych, ponieważ powiadomienie można przekazać w dowolnym momencie, ale obecnie aplikacje webowe również pozwalają na wyświetlanie powiadomień w przeglądarce użytkownika.

Ze względu na wieloplatformowość i dostępność zbliżonego interfejsu na urządzeniach desktopowych i mobilnych, aplikacja będzie korzystać z rozwiązań webowych. Część serwerowa zostanie wykonana w formie RESTowego API napisanego w języku *C#* z wykorzystaniem frameworka *ASP.NET Core*. Część kliencka będzie wykorzystywać framework *Angular* w połączeniu z językiem *TypeScript*. Na funkcjonalność dotyczącą powiadomień pozwala zastosowanie technologii PWA (*Progressive Web Apps*) opisanej w dalszej części pracy.

Praca zawiera rozdziały skupiające się na szczegółach dotyczących kolejnych etapów procesu powstawania aplikacji, od przeglądu istniejących na rynku rozwiązań, przez przyjęte założenia projektowe i projekt, do implementacji i testów.

# Przegląd istniejących rozwiązań

Istniejące rozwiązania można podzielić na 3 główne grupy – standardowe przewodniki programowe, silniki rekomendacji i serwisy streamingowe. W dalszej części nastąpi przegląd każdej z tych grup zawierający przykłady, ich funkcjonalność i ograniczenia.

## Standardowe przewodniki programowe



Standardowe przewodniki programowe pozwalają na przegląd aktualnie nadawanych programów. Jest to bardzo ważna, podstawowa funkcjonalność, jednakże z wielu powodów niewystarczająca. Brakuje po pierwsze, możliwości zapamiętania ustawień, a po drugie dostosowania wyświetlanych programów do preferencji. W przypadku programu dostępnego przez portal WP.pl , jedynym elementem personalizacyjnym jest zapisywany w pliku *cookie* filtr kanałów wg dostawcy, podobnie w przypadku Onetu i Interii (w 2 ostatnich dochodzi jeszcze możliwość zmiany widoku z pionowego na poziomy).

Wszystkie 3 główne przykłady oferują możliwość wyszukiwania na podstawie tytułu programu. Szukanie na podstawie cech charakterystycznych umożliwia tylko wp.pl, ale nawet tam jest dość ograniczone. Ponieważ nie zapamiętują żadnych danych poza filtrem kanałów, trzeba za każdym razem określać czego szukamy, co jest uciążliwe. Poza tym brakuje jakichkolwiek sugestii dla użytkownika który nie wie dokładnie czego szuka. Pomiędzy poszczególnymi dostawcami nie stwierdzono istotnych różnic w funkcjonalności.

## Silniki rekomendacji

Przykłady:



Silniki rekomendacji są doskonałym rozwiązaniem dla niezdecydowanego użytkownika. Wymagają wpisania pozycje (w tym przypadku seriale, ale istnieją również takie systemy dla, np. filmów, książek czy muzyki), które użytkownik zna i lubi, a w odpowiedzi przedstawiają inne, podobne. Dla zestawu na przykładowym screenie aplikacja zaproponuje podobną produkcję opartą na komiksach – The Flash.

Silnik rekomendacji CableTV nie pozwala na żadną dodatkową personalizację, nie można zapamiętać rekomendacji, jedyny sposób na dokładniejsze rekomendacje to 2 przyciski – „not that one” i „start again”. Prostota użytkowania jest niewątpliwym plusem, ale brakuje wielu istotnych funkcjonalności, przede wszystkim integracji z przewodnikiem programowym.

Istnieją również silniki rekomendacji korzystające z serwisów społecznościowych do określenia preferencji użytkownika, ich przykładem jest tastedive.com . Tastedive działa w 2 trybach – pierwszy jest podobny do poprzedniego, ale pozwala tylko na wprowadzenie jednej pozycji (serialu, filmu, książki, artysty muzycznego lub autora). Drugi wymaga zalogowania się i pozwala oznaczać istniejące pozycje jako ulubione (jeżeli logujemy się za pośrednictwem portalu Facebook, Tastedive zbiera nasze polubienia i wyznacza rekomendacje na tej podstawie).

Problemem, który ogranicza przydatność wszystkich aplikacji z tej grupy jest to, że nie uwzględniają w żaden sposób aktualnego programu telewizyjnego. Znalezienie serialu lub filmu, który okaże się interesujący (o czym użytkownik wie dzięki rekomendacjom) nadal wymaga przeszukiwania tradycyjnych programów telewizyjnych.

## Serwisy streamingowe



Serwisy streamingowe są mocno zróżnicowaną grupą, pod względem dostarczanej treści, kosztów użytkowania jak i kwestii prawnych.

Jednym przykładem jest oferowany przez portal *wp.pl* serwis *WP pilot* – pozwala on oglądać dokładnie to, co jest aktualnie nadawane w telewizji. Podstawowa oferta 28 kanałów jest dostępna za darmo, pełna wymaga opłat. Wyszukiwarka pozwala znaleźć kanał lub aktualnie nadawany program na podstawie tytułu lub opisu, dostępny jest też podgląd programów na najbliższe godziny. Nadal brakuje możliwości zapamiętania ulubionych kanałów lub programów, nawet po zalogowaniu, a także rekomendacji.

Przykładem z drugiej strony jest serwis *ipla.tv* (7). Dostarcza on ofertę filmów, seriali i programów sportowych uboższą od telewizyjnej, ale dostępną na żądanie, w większości za opłatą, choć częśc programów jest darmowa. Rekomendacje sprowadzają się jedynie do wskazania najpopularniejszych, nie są personalizowane. Podobne możliwości, poszerzone o rekomendacje, oferują serwisy takie, jak *Netflix*, *Hulu*, czy *Amazon Video.*

## Filmweb



Polski *Filmweb* (8) jest dość unikalnym przypadkiem, który wydaje się oferować wszystko, czego można by wymagać. Przewodnik programowy, rekomendacje uwzględniające preferencje użytkownika i nadawane programyboth, możliwość wyszukiwania na podstawie różnych cech, filtrowania kanałów wg dostawców i tematycznie.

Brakuje jednak kilku elementów, po pierwsze programów sportowych, popularnonaukowych czy ogólnie niezwiązanych ze światem filmów. Nie powinno to być zaskoczeniem, biorąc pod uwagę to, że Filmweb od początku skupiony był, jak sama nazwa wskazuje, na filmach, częściowo także na serialach. Jest to jednak funkcjonalność, która ma duże znaczenie dla wielu użytkowników.

Poza tym brakuje powiadomień. Dostępne są powiadomienia dotyczące aktywności w serwisie – opublikowanych recenzji, dyskusji na forach itd., ale nic związanego z programem telewizyjnym. Poza tym, nawet te powiadomienia, które są dostępne, wykorzystują e-mail lub komunikator GG zamiast korzystania z nowszych technologii powiadomień push – przez przeglądarkę lub aplikację mobilną.

## Tabela porównawcza

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| funkcjonalność | programtv .onet.pl (2) | tv.wp.pl (1) | pilot.wp.pl (6) | filmweb.pl (8) | tastedive.com (5) | cabletv.com (4) | ipla.tv (7) | *Looking Glass* |
| przegląd nadawanych programów | TAK | TAK | TAK | TAK | NIE | NIE | NIE | TAK |
| personalizacja kanałów | TAK | TAK | TAK | TAK | NIE | NIE | NIE | TAK |
| personalizacja treści | NIE | NIE | NIE | TAK | NIE | TAK | TAK | TAK |
| personalizacja czasu | TAK | TAK | NIE |  | NIE | NIE | TAK | TAK |
| wyszukiwanie programów | TAK | TAK | TAK | *2 niżej* | TAK | TAK | TAK | TAK |
| widok chronologiczny | TAK | TAK | NIE | TAK | NIE | NIE | NIE | TAK |
| rekomendacje filmów | *3 niżej* | *3 niżej* | NIE | TAK | TAK | NIE | *3 niżej* | TAK |
| rekomendacje seriali | *3 niżej* | *3 niżej* | NIE | TAK | TAK | TAK | *3 niżej* | TAK |
| rekomendacje programów sportowych | *3 niżej* | *3 niżej* | NIE | NIE | NIE | NIE | *3 niżej* | TAK |
| VOD | NIE | NIE | NIE | NIE | NIE | NIE | TAK | NIE |
| streaming treści na żywo | NIE | NIE | TAK | NIE | NIE | NIE | TAK | NIE |
| powiadomienia | NIE | NIE | NIE | NIE | NIE | NIE | NIE | TAK |
| darmowy | TAK | TAK |  | TAK | TAK | TAK |  | TAK |

**Uwagi:**

1. ograniczenia w personalizacji czasu nadawania polegają na możliwości wyboru tylko określonych pór dnia (cały dzień, po południu lub wieczorem)
2. ograniczenia w wyszukiwaniu programów - można znaleźć tylko tytuły
3. ograniczenia w rekomendacjach - wyświetlane są "najpopularniejsze" programy danego typu, bez uwzględnienia preferencji użytkownika
4. *pilot.wp.pl* ma darmowy zestaw kanałów, *ipla.tv* pozwala za darmo obejrzeć tylko pojedyncze programy (głównie informacyjne). *popcornflix.com* jest darmowy, ale dostępne treści są ograniczone do tych mniej popularnych

## Wnioski

Funkcjonalność dotyczącą rekomendacji i programu w pewien sposób łączy serwis Filmweb , jednak skupia się on tylko na filmach i serialach. W związku z tym nie uwzględnia programów sportowych ani muzycznych, a dane dotyczące np. programów popularnonaukowych są mocno ograniczone.

Brakuje na rynku rozwiązań oferujących rekomendacje z uwzględnieniem aktualnego programu. Legalnie działające aplikacje dostarczające treści (w formie VOD lub strumienia na żywo) najczęściej są płatne (co nie powinno dziwić ze względu na koszty samej treści). Z tego powodu implementacja tej funkcjonalności nie mieści się w zakresie pracy.

Wszystkie rozwiązania pozwalające na wyszukiwanie mają jeden problem, polegający na tym, że wymagają poprawnie wprowadzonych tytułów, nie są w stanie skorygować błędów użytkownika. Literówka w tytule, oznacza że wyszukiwarka nie zwróci żadnych wyników, mimo że istnieją tytuły podobnie brzmiące, które lepszy algorytm byłby w stanie wychwycić. Jest to szczególnie istotne w przypadku aplikacji skupionych na rekomendacjach.

# Założenia projektowe

## Opis problemu

Obecnie nie ma na rynku żadnego rozwiązania, które pozwalałoby stwierdzić który z aktualnie nadawanych programów zainteresowałby użytkownika. Takie rozwiązanie jest potrzebne, ze względu na dużą liczbę możliwości, która sprawia że ręczne szukanie jest niewygodne, a wielu użytkowników szuka „czegoś do oglądania na wieczór”, bardziej niż konkretnego programu.

## Wizja rozwiązania

Aplikacja ma łączyć w sobie rozwiązania z silników rekomendacji i standardowych programów TV. Dzięki temu użytkownicy będą mogli dowiedzieć się jaki interesujący ich program jest nadawany, w momencie kiedy jest nadawany. To pozwoli im skupić się na oglądanym programie, zamiast na poszukiwaniach i spędzić więcej czasu na oglądaniu.

Po stronie klienta główny widok zawiera chronologiczny wykaz rekomendowanych programów nadawanych w najbliższym tygodniu. Wykaz uwzględnia programy powtarzane – zawiera tylko jedno wystąpienie, wybrane na podstawie godzin w sposób unikający ewentualnych konfliktów z innymi programami, a także uwzględniający godziny w jakich użytkownik ma dostęp do telewizora. Ponadto główny widok zawiera formularz do konfiguracji filtrów, pozwalający użytkownikowi ustalić godziny, kanały i zakres tematyczny. Konflikty, których nie udało się uniknąć są zaznaczone kolorem. Schematy interfejsu zostaną przedstawione w części projektowej.

Do dodatkowych widoków należy widok przewodnika programowego, zawierający wszystkie nadawane programy bez uwzględniania rekomendacji, widok wyszukiwania oraz widok pokazujący szczegóły danego programu. Z poziomu każdego z tych widoków użytkownik może zaznaczyć, że dany program mu się podobał lub nie, może też zapamiętać że chce obejrzeć dany program. Ustawione filtry czasowe i kanałowe oraz oznaczenia programów są zapisywane na serwerze na bieżąco.

Po stronie serwera, rekomendacje generowane są na podstawie cech charakterystycznych programów, do których zaliczamy:

* Osoby pracujące przy produkcji programu (aktorzy, reżyserowie, prezenterzy, scenarzyści, w przypadku transmisji sportowych – grające drużyny, w przypadku programów muzycznych – wykonawcy, itd.)
* Kategorie (dla filmów i seriali są to, np. thriller, fantasy, komedia, dokument; dla sportu dyscyplina i ranga zawodów – liga, puchar)

Obsada i kategoria są znane dzięki danym zebranym z programów telewizyjnych. Słowa kluczowe są dobierane automatycznie. Pozwoli to na wychwycenie tematycznych podobieństw, które nie są widoczne w zdefiniowanych cechach (jak kategorie czy obsada).

**Metoda generowania rekomendacji**

Rekomendacje generowane są w oparciu o ważoną średnią ocen użytkownika wyrażoną wzorem:

Gdzie:

* – program, którego współczynnika rekomendacji poszukujemy
* – programy wcześniej ocenione przez użytkownika
* – ocena wystawiona j-temu programowi przez użytkownika, wynosi 1, jeżeli użytkownik zaznaczył, że program mu się podobał, 0 jeżeli zaznaczył, że program mu się nie podobał. Do tych celów, zapisanie programu jako ten, który użytkownik chce obejrzeć, jest traktowane jako ocena pozytywna.
* – podobieństwo dwóch programów określone wg wzoru:

Gdzie – wagi dla i-tej grupy cech, – podobieństwo i-tej grupy cech danego programu. Wyróżniamy 5 grup cech:

* Gatunki
* Aktorzy
* Reżyser
* Kraj produkcji
* Rok produkcji

Dla gatunków, słów kluczowych, reżyserów i aktorów, podobieństwo wyrażamy współczynnikiem Dice’a:

Dla kraju produkcji podobieństwo jest binarne, wynosi 1 gdy wartości są te same, 0 w przeciwnym wypadku.

Dla roku produkcji podobieństwo wyraża się wzorem:

Jest to funkcja homograficzna, przyjmująca wartość 0, gdy argument jest równy 0, wartość gdy i asymptotycznie dążąca do wartości 1. Dla parametru przyjmujemy wartość 20 lat.

Wagi dostosowywane są indywidualnie do każdego z użytkowników. Przy rejestracji nowego użytkownika przyjmuje się wartości średnie dla wszystkich użytkowników, a w przypadku ich braku, wartości następujące:

* Aktorzy: *0,1*
* Kategorie: *0,3*
* Reżyser: *0,1*
* Kraj produkcji: *0,1*
* Rok produkcji: *0,1*

Za każdym razem, kiedy użytkownik oceni program, dokonywana jest korekcja wag o wartość:

Gdzie

* – parametr z przedziału ,
* – ocena wystawiona programowi przez użytkownika,
* – wartość funkcji wsparcia rekomendacji dla programu

## Źródła danych

Przewodnik programowy pobierany jest z portalu *Onet.pl* , udostępniającego listę nadawanych programów dla wszystkich 156 kanałów oferowanych przez polskich dostawców telewizji (naziemna telewizja cyfrowa – oznaczona skrótem DVB-T, Cyfrowy Polsat, Multimedia, nc+, UPC i Vectra). W przypadku gdy dany kanał dostępny jest w wersji standardowej i HD (a treść pozostaje taka sama), wybrano wersję standardową. Lista branych pod uwagę kanałów dostępna jest w załączniku.

## Wymagania funkcjonalne

Gość:

* ST-001: Jako gość, chcę przeglądać program TV, żeby sprawdzić co mogę obejrzeć
* ST-002: Jako gość, chcę wybrać tylko te kanały które mnie interesują, żeby uprościć szukanie
* ST-003: Jako gość, chcę zawęzić kanały do oferty mojego dostawcy, żeby uprościć szukanie
* ST-004: Jako gość, chcę sprawdzić o której godzinie nadawany jest interesujący mnie program
* ST-005: Jako gość, chcę zarejestrować się w systemie, żeby zapisać moje preferencje

[Zarejestrowany] użytkownik

* ST-006: Jako użytkownik chcę sprawdzić kiedy nadawane są interesujące mnie programy
* ST-007: Jako użytkownik chcę otrzymywać powiadomienia o interesujących mnie programach, żeby ich nie przegapić
* ST-008: Jako użytkownik chcę otrzymać rekomendacje programów, które mogą mnie zainteresować, żeby ułatwić wyszukiwanie
* ST-009: Jako użytkownik chcę potwierdzić lub odrzucić rekomendację, żeby otrzymywać dokładniejsze wyniki

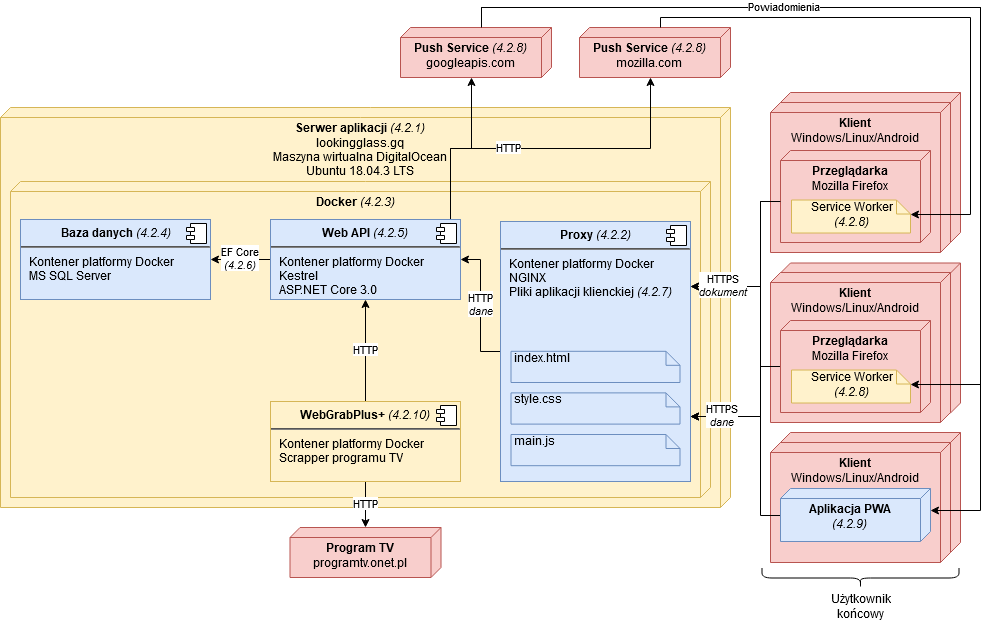
# Projekt aplikacji

## Architektura aplikacji

Aplikacja składa się z 3 głównych warstw:

* Baza danych
* Interfejs API
* Klienci
  + Interfejs webowy
  + Scrapper pobierający dane

Interfejs webowy, z uwagi na łatwość dołączenia elementów typowych dla aplikacji natywnych (takich jak powiadomienia push), jest wykonany w stylu SPA *(Single Page Application)* – tzn. nie wymagającym przeładowania całej strony pomiędzy poszczególnymi widokami. Oznacza to, że nie można skorzystać wbudowanych w większość serwerowych frameworków mechanizmów renderowania po stronie serwera.



Na diagramie zastosowano następujące kolory:

* Niebieski – elementy opracowane od podstaw w ramach pracy
* Żółty – elementy konfigurowane w ramach pracy
* Czerwony – elementy zewnętrzne

## Zastosowane technologie

* + 1. **Hosting – DigitalOcean**

Aplikacja jest hostowana na maszynie wirtualnej (Virtual Private Server) dostarczonej przez platformę DigitalOcean. Maszyna wirtualna ulokowana jest w należącej do DigitalOcean serwerowni we Frankfucie (jedna z wielu dostępnych lokalizacji, wybrana ze względu na geograficzną bliskość Polski, a przez to głównych odbiorców aplikacji) i ma dostęp do 1 rdzenia procesora, 2 GB pamięci RAM (minimalna wymagana ilość dla bazy danych MS SQL Server) i 25 GB przestrzeni na dysku SSD. Dodatkowo DigitalOcean zapewnia obsługę systemów DNS.

* + 1. **Serwer aplikacji webowej – NGINX**

NGINX służy jako serwer proxy aplikacji, spełniający trzy główne funkcje

* Obsługa statycznych plików (w tym pliki HTML, CSS i JavaScript składające się na aplikację kliencką)
* Proxy przekazujące żądania dotyczące danych do serwera Kestrel obsługującego API
* Obsługa certyfikatów SSL (i kierowanie niezaszyfrowanych żądań HTTP przychodzących z Internetu na zabezpieczony protokół HTTPS)
  + 1. **Konteneryzacja – docker**

Aplikacja korzysta z platformy Docker , co pozwala na zautomatyzowanie procesu instalacji i gwarantuje, że aplikacja będzie działała na dowolnym serwerze tak samo jak na deweloperskim. Docker pozwala stworzyć obraz środowiska, w którym działa aplikacja, środowisko zawiera system operacyjny i wszystkie potrzebne zależności (biblioteki, pakiety etc.), nie wymagając jednocześnie emulacji sprzętu (jak w przypadku maszyn wirtualnych). Zawartość każdego kontenera i kolejne czynności podczas jego budowy są określone w pliku Dockerfile.

* + 1. **Baza danych – Microsoft SQL Server** (12)

Na platformie docker dostępne są wszystkie wiodące systemy zarządzania bazą danych (przede wszystkim Oracle, MSSQL, MySQL, PostgreSQL i MariaDB) . Decyzja dotycząca wykorzystania rozwiązania Microsoftu jest podyktowana przede wszystkim wsparciem po stronie kolejnej warstwy *(ASP.NET)*

* + 1. **Interfejs API – ASP.NET Core 3.0** (14)

Rozwiązania pozwalające na tworzenie serwisów typu Web API dostępne są w większości języków programowania, przykładowo:

* Spring (Java)
* Flask (Python)
* Django (Python)
* Express.js (node.js, Javascript)
* *ASP.NET (C#)*

Na wybór frameworka ASP.NET Core wpłynęła wcześniejsza znajomość środowiska .NET oraz dostępność bibliotek ułatwiających implementację potrzebnych funkcjonalności (m.in. komunikacja z bazą danych i powiadomienia push)

Kluczową funkcjonalnością dostarczaną przez framework ASP.NET Core (i inne tego typu rozwiązania) jest powiązanie kontrolerów i ich metod z odpowiednimi żądaniami HTTP przekazywanymi pod poszczególne adresy. Jest to zrealizowane za pomocą odpowiednich anotacji. Ponieważ każda klasa kontrolera dziedziczy po należącej do frameworka klasie *BaseController,* do programisty należy tylko implementacja odpowiednich metod i oznaczenie ich anotacjami, powiązanie z adresem URL i metodą HTTP następuje automatycznie.

* + 1. **Dostęp do bazy danych – EntityFramework Core** (15)

Do komunikacji z bazą danych aplikacja wykorzystuje EntityFramework Core, udostępnione przez Microsoft narzędzie ORM (*Object – Relational Mapping*) w podejściu *code – first*. W tym podejściu programista definiuje klasy obiektów domenowych (typu POCO, nie wykonujące żadnej logiki, ani nie posiadające żadnych zależności) oraz klasę kontekstu (w tym przypadku *TvAppContext*) przechowującą parametry połączenia z bazą (i dodatkowe informacje dotyczące schematu nieokreślone przez klasy domenowe – np. indeksy). Na podstawie tych klas EntityFramework generuje kod migracji, a potem polecenia DDL wywoływane na bazie danych, w tym przypadku w języku SQL.

Główną zaletą takiego podejścia jest znaczne uproszczenie komunikacji z bazą danych przez dodatkową warstwę abstrakcji. Zapis i odczyt danych z bazy sprowadza się do interakcji z kolekcją (dostarczona w EF Core klasa bazowa *DbSet* implementuje interfejs *IEnumerable*, ten sam który wykorzystują wszystkie dostępne w C# kolekcje). W podejściu *code – first* całkowicie wyeliminowana została konieczność pisania zapytań do bazy za pośrednictwem SQL.Dodatkowo pozwala to korzystać z biblioteki LINQ, która udostępnia metody pozwalające na filtrowanie, sortowanie i przekształcanie danych w kolekcji.

Dodatkowo wykorzystanie narzędzia ORM zwiększa poziom bezpieczeństwa aplikacji blokując ataki typu *SQL injection* – polegające na „wstrzyknięciu” kodu SQL do składanych dynamicznie zapytań.

* + 1. **Interfejs webowy – Angular 7** (16)

Aplikacja webowa typu *Single Page Application* ogranicza dostępny wybór technologii dla klienta do trzech głównych frameworków działających w oparciu o język Javascript:

* Angular
* React.JS
* Vue.JS

Ze względu na popularność, ilość dostępnych źródeł oraz łatwość implementacji aplikacji typu SPA, wybrany został Angular. Główne funkcjonalności zawarte w Angularze (oraz, w większości, w pozostałych frameworkach) dotyczą:

* Przekształcania wcześniej przygotowanych szablonów do gotowego kodu HTML
* Komunikacji z zewnętrznym API przez wywołania AJAX
* Powiązania poszczególnych komponentów strony z adresem URL wpisanym w przeglądarce (przez wbudowany router)
  + 1. **Powiadomienia – Push API** (17)

Komunikacja inicjowana przez serwer jest częstym problemem przy projektowaniu i implementacji aplikacji webowych. Obecnie dostępnych jest kilka rozwiązań pozwalających na taką komunikację, należą do nich:

* Server Side Events
* Web Sockety
* Firebase Cloud Messaging
* Web Push API

Server Side Events i Web Sockety są wspierane przez dostępny w .NET Core pakiet SignalR , co w pewnym stopniu ułatwiałoby implementację. Szczególnie Web Sockety są obecnie często stosowane w aplikacjach czasu rzeczywistego (czaty itp.) Problem z ich zastosowaniem polega na tym, że wymagają stałego działania „głównej” aplikacji klienckiej. Jeżeli aplikacja nie jest w danym momencie aktywna, komunikat nie dotrze. Jednym z wymagań dotyczących powiadomień jest możliwość przekazywania ich użytkownikom, którzy w danym momencie nie korzystają aktywnie z aplikacji, co dyskwalifikuje te technologie.

Web Push API dostarcza podobną funkcjonalność pozwalającą na przekazywanie powiadomień z serwera do klienta, nawet jeśli klient nie jest aktywny. Korzysta z par kluczy VAPID do uwierzytelniania aplikacji i serwisów zarządzanych przez Google i fundację Mozilla do przekazywania właściwych powiadomień. Korzystanie z tego mechanizmu w aplikacji webowej wymaga zarejestrowania w przeglądarce odpowiedniego skryptu określanego jako *service worker* (korzystanie z *service workerów* wymaga z kolei zarejestrowania domeny i obsługi strony przy użyciu zaszyfrowanego protokołu HTTPS).

Firebase Cloud Messaging jest jedną z funkcjonalności dostarczonych przez Google w ramach usługi Firebase. Poza Cloud Messaging, Firebase dostarcza wiele możliwości takich jak zarządzana baza danych, uwierzytelnianie użytkowników i hosting. Przekazywanie powiadomień działa tak samo jak przy wykorzystaniu Web Push API.

Ze względu na problemy z komunikacją offline w przypadku Web Socketów, oraz silną integrację z innymi usługami Google w przypadku Firebase Cloud Messaging, w aplikacji *Looking Glass* zastosowano mechanizm Web Push API.

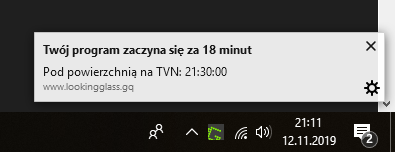
Do skutecznego przekazania użytkownikowi powiadomienia przy użyciu mechanizmu Web Push API potrzebne są dwie czynności:

* Rejestracja subskrypcji dla danego klienta w serwisie Push
* Przekazanie powiadomienia przez serwer do serwisu Push

Rejestracja subskrypcji odbywa się z poziomu klienta przez wywołanie odpowiedniej metody. Po rejestracji dane subskrypcji (adres pod który należy przekazać żądanie oraz klucze szyfrujące i identyfikujące aplikację w serwisie) muszą zostać zapisane na serwerze.

Przekazanie powiadomienia do serwisu polega na przesłaniu żądania HTTP z serwera aplikacji pod zapisany w momencie rejestracji subskrypcji adres, wewnątrz żądania zawarte są klucze identyfikujące aplikację oraz właściwa treść wiadomości.

Efektem jest powiadomienie, którego wygląd jest różny w zależności od przeglądarki i systemu operacyjnego, ale pojawia się bez dalszej interakcji ze strony użytkownika. W przypadku Firefoxa działającego pod Windowsem 10 wygląda następująco:

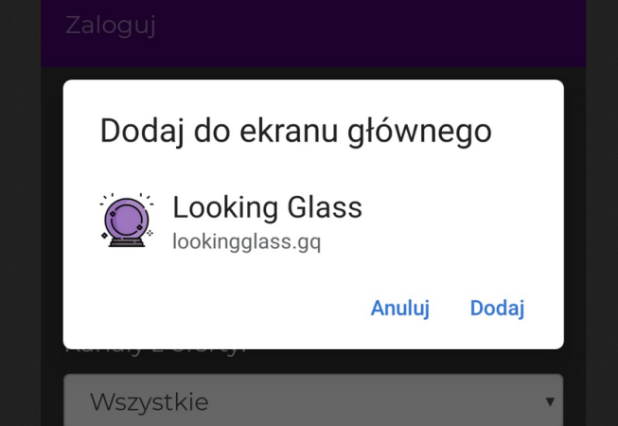


* + 1. **Powiadomienia offline – PWA** (20)

Technologia PWA (*Progressive Web App*) pozwala na zainstalowanie aplikacji na takich samych zasadach jak natywne aplikacje mobilne. Zainstalowana w ten sposób aplikacja ma swój skrót na głównym ekranie użytkownika, jest częściowo dostępna offline, a ponadto pozwala przekazywać powiadomienia nawet przy wyłączonej przeglądarce. Żeby można było zainstalować aplikację w ten sposób musi ona spełniać kilka warunków:

* Używać szyfrowanego połączenia HTTPS
* Rejestrować skrypt *service worker* kontrolujący stronę główną i adres startowy
* Zawierać plik *manifest.json*

Spełnienie pierwszego wymagania gwarantuje odpowiednia konfiguracja serwera NGINX, dwóch ostatnich – odpowiednia konfiguracja pakietu *@angular/pwa*. Jeżeli aplikacja spełnia te warunki, po otwarciu na urządzeniach mobilnych pojawia się opcja „Dodaj do ekranu głównego”, której wybranie skutkuje instalacją aplikacji.



* + 1. **Scrapper – WebGrabPlus+** (21)

Drugim z klientów korzystających z API jest WebGrabPlus, odpowiedzialny za ekstrakcję danych z Internetu jest darmowym programem o zamkniętym kodzie źródłowym. Efektem jego działania jest plik XML o rozmiarze ok. 25 MB (przy obecnej konfiguracji – 156 kanałów na 4 dni). Po zakończeniu działania, plik jest przesyłany w ramach wewnętrznej sieci dockera do interfejsu API, dalej przetwarzany przez serwer aplikacji i docelowo ładowany do bazy danych.

Pobieranie danych jest uruchamiane cyklicznie, codziennie wieczorem, przez unixową usługę *cron*.

## Projekt bazy danych

Wykorzystanie biblioteki EntityFramework Core sprowadza projekt bazy danych do określenia klas głównych obiektów domenowych. Przejście od klas do tabel SQL odbywa się automatycznie.

* + 1. **Analiza wymaganych danych**
       1. **Przedstawienie listy nadawanych programów użytkownikowi**

Różne programy nadawane są w różnych godzinach i na różnych kanałach. Aplikacja powinna przechowywać dane dotyczące programu (co najmniej jego tytuł, ale inne informacje takie jak opis, czy występujący aktorzy również są pomocne), oraz tego na którym kanale będzie nadawany, o której się zacznie i skończy.

* + - 1. **Zbieranie preferencji użytkownika**

Do określenia preferencji potrzebna jest informacja identyfikująca użytkownika, oraz to jak oceniał poszczególne programy (pozytywnie, negatywnie lub neutralnie).

* + - 1. **Wyznaczenie rekomendacji**

Zgodnie z analizą przeprowadzoną w rozdziale 3.2, do wyznaczenia wartości funkcji rekomendacji dla danego programu, potrzebne są jego cechy charakterystyczne (do tych zaliczamy gatunek, występujących aktorów, reżysera, kraj produkcji oraz rok produkcji). W zależności od typu cechy, może być inaczej traktowana w ramach funkcji rekomendacji, zatem należy również przechowywać dane o typach cech. Ponieważ wagi w systemie rekomendacji są dostosowane do użytkownika, również powinny być przechowywane w bazie.

* + - 1. **Przekazywanie powiadomień**

Mechanizm działania Push API przedstawiony w rozdziale 4.2.7 wymaga przechowywania adresu i kluczy dla każdej zarejestrowanej subskrypcji. Użytkownik może otrzymywać powiadomienia na kilka urządzeń.

* + - 1. **Filtrowanie kanałów**

Kanały można pogrupować w zależności od tematu lub dostawcy, który je udostępnia. Filtrowanie powinno uwzględniać obie te informacje. Kanał może należeć do kilku ofert lub grup tematycznych.

* + 1. **Wyodrębnione klasy**
* *Programme* – reprezentująca pojedynczy program telewizyjny (punkt 1, 2, 3)
* *Emission* – reprezentuje jedną emisję określonego programu (początek, koniec i kanał; punkt 1, 4)
* *Channel* – reprezentuje jeden kanał telewizyjny (punkt 1, 5)
* *ChannelGroup* – reprezentuje grupę kanałów (tematyczną lub ofertową, punkt 5)
* *GroupedChannel* – klasa wyodrębniona ze względu na związek N:N pomiędzy *Channel*, a *ChannelGroup*, reprezentuje przynależność kanału do grupy (punkt 5).
* *Feature* – reprezentuje pojedynczą cechę programu (np. nazwisko aktora lub rok produkcji, punkt 3)
* *FeatureType* – reprezentuje typ cechy (aktor, reżyser itp.; punkt 3)
* *ProgrammesFeature –* klasa wyodrębniona ze względu na związek N:N pomiędzy *Programme*, a *Feature* – reprezentuje powiązanie między programem a cechą   
  (punkt 3)
* *User* – reprezentuje pojedynczego użytkownika aplikacji (punkt 2, 3, 4)
* *Notification* – reprezentuje zażądane przez użytkownika powiadomienie (*Przypomnij mi* o danej emisji programu, punkt 4)
* *Rating* – reprezentuje pojedynczą ocenę wystawioną przez użytkownika programowi (punkt 2 i 3)
* *Subscribtion* – reprezentuje subskrypcję dla danego użytkownika i danego urządzenia, zawiera adres dla serwisu Push API i klucze identyfikujące aplikację (punkt 4)

Dodatkowo ze względów porządkowych dołączona została klasa *GuideUpdate*, zawierająca godzinę rozpoczęcia i zakończenia kolejnych aktualizacji listy dostępnych programów. Ze względów wydajnościowych, ponieważ opis programu nie zawsze jest dostępny (i nie zawsze jest potrzebny), a może zawierać duże ilości tekstu, będzie przechowywany w osobnej tabeli – odpowiada jej klasa *Description*.

## Komunikacja między warstwami

Dostępne listy programów – widoki i powiązane z nimi endpointy API

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nr | Nazwa | | Zalogowany | Filtr | Sortowanie | Dodatkowa funkcjonalność | |
| V-01 | Rekomendacje | | TAK | Funkcja rekomendacji  > próg | Funkcja rekomendacji |  | |
| V-02 | Ocenione | | TAK | Isnieje ocena | Wartość oceny | Wartość oceny | |
| V-03 | Kanał | | NIE | Emisja na danym kanale | Godzina emisji |  | |
| V-04 | Kanał | | TAK | Emisja na danym kanale | Godzina emisji |  | |
| V-05 | Cecha | | NIE | Występowanie danej cechy | Godzina emisji | - | |
| V-06 | Cecha | | TAK | Występowanie danej cechy | Godzina emisji | - | |
| V-07 | Podobne | | NIE | Podobieństwo do wskazanego programu  > próg | Podobieństwo do wskazanego programu | Szczegóły wskazanego programu | |
| V-08 | Podobne | | TAK | Podobieństwo do wskazanego programu  > próg | Podobieństwo do wskazanego programu | Szczegóły wskazanego programu | |
| V-09 | Szukaj | | NIE | Tytuł lub cechy zawierają szukane słowa | Ilość dopasowanych słów | - | |
| V-10 | Szukaj | | TAK | Tytuł lub cechy zawierają szukane słowa | Ilość dopasowanych słów | - | |
| Nr widoku | | Adres kontrolera API | | | | |
| V-01 | | /api/Users/*login*/Recommended | | | | |
| V-02 | | /api /Users/*login*/Ratings | | | | |
| V-03 | | /api /Channel/*id*/Programmes | | | | |
| V-04 | | /api /Channel/*id*/Programmes | | | | |
| V-05 | | /api/Feature/*id*/Programmes | | | | |
| V-06 | | /api/Feature/*id*/Programmes?username=*login* | | | | |
| V-07 | | /api/Programmes/*id*/Similar | | | | |
| V-08 | | /api/Programmes/*id*/Similar?username=*login* | | | | |
| V-09 | | /api/Programmes?search=*search terms* | | | | |
| V-10 | | /api/Programmes?search=*search terms* | | | | |

# Implementacja

## Serwer

* + 1. **Obiekty domenowe i komunikacja z bazą danych**

Klasy obiektów domenowych zostały wymienione w rozdziale 4.3.2. Każda z wymienionych tam klas została opisana w języku C# z wykorzystaniem anotacji z biblioteki *EntityFramework Core* identyfikujących nazwy tabeli i kolumn, oraz parametry kolumn. Składają się one z publicznych właściwości i anotacji, na podstawie których *EntityFramework* określa typy poszczególnych kolumn w bazie danych.



Przykładowa implementacja klasy reprezentującej kanał telewizyjny.

* + 1. **Funkcjonalność**
       1. **Wczytywanie danych**

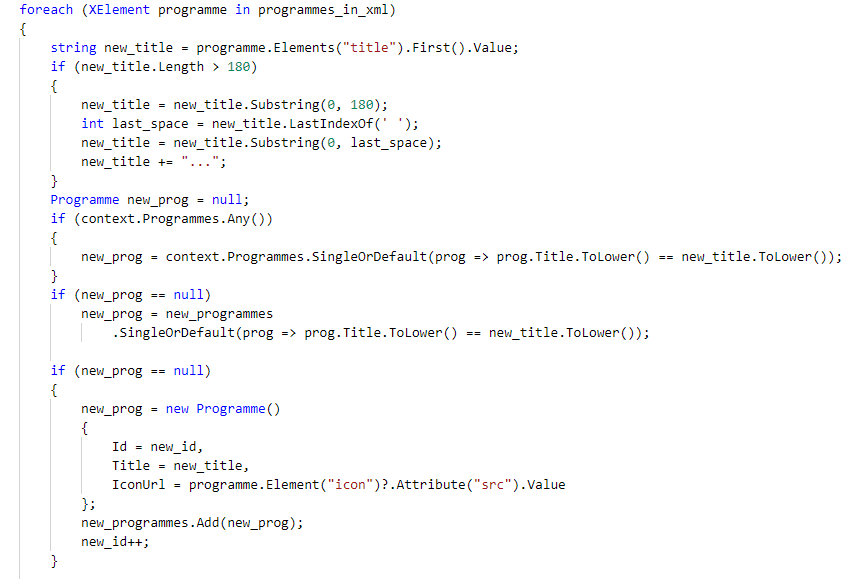
Wczytywanie danych jest procesem dwuetapowym:

* Etap 1 – pobranie programu z Internetu
* Etap 2 – wczytanie programu do bazy

Etap 1 jest realizowany przez zewnętrzny program Webgrabplus, działający w osobnym kontenerze dockerowym i uruchamiany cyklicznie, codziennie o 1. w nocy. Efektem działania jest plik XML zawierający listę kanałów i nadawanych programów na ustaloną liczbę dni (w tym przypadku 4).

Etap 2 jest realizowany przez właściwy serwer aplikacji, do tego celu jest wyznaczony osobny endpoint API – pod adresem */api/GuideUpdate*. Zaciągnięty w etapie 1 program jest wysyłany do serwera aplikacji pod tym adresem.

Serwer po odebraniu pliku przekształca zapis XML na odpowiednie obiekty domenowe (sprawdzając, czy nie istnieją one wcześniej w bazie), po czym zapisuje przekształcone obiekty w bazie danych. Za samo przetwarzanie pobranych danych odpowiada klasa *GuideUpdateService*. Korzysta ona z wbudowanego w język C# narzędzia   
LINQ-to-XML, które znacznie ułatwia przetwarzanie plików w tym formacie.

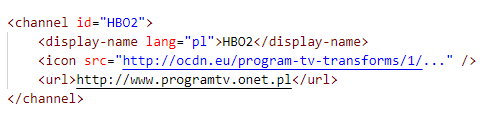


W podobny sposób przetwarzane są dane dotyczące godzin emisji i cech. Dla przykładowego programu – filmu *Ant-Man i Osa*, nadawanego na HBO2 – we wczytanym pliku XML znajduje się następujący wpis:

Aplikacja wykona następujące kroki:

1. Sprawdzenie czy program znajduje się już w bazie lub na liście programów do dodania (jeżeli tak – dalsza analiza jest prowadzona na istniejącej instancji programu, jeżeli nie – tworzona jest nowa instancja)
2. Przypisanie do ustalonej instancji programu godzin emisji – atrybuty *start*, *stop* i *channel* znacznika *programme*. Kanał jest wyszukiwany w bazie danych na podstawie nazwy.
3. Jeżeli program w XMLu zawiera opis, a odpowiadający mu wpis w bazie danych go nie zawiera – utworzenie nowej instancji klasy *Description* i przypisanie jej do ustalonego programu.
4. Analiza cech:
   1. z XMLa wyciągane są znaczniki o nazwach *date*, *category*, *country*, oraz wszystkie elementy znajdujące się pod znacznikiem *credits* – tutaj *actor* i *director*.
   2. Dla każdej z cech na przygotowanej liście aplikacja sprawdza, czy istnieje już w bazie cecha o tej samej wartości i typie (np. typ *aktor* i wartość *Paul Rudd*), jeżeli nie – tworzy nową cechę i dodaje ją do listy cech programu, jeżeli tak – dodawana jest istniejąca.
5. Efektem jest obiekt domenowy klasy *Programme* z dołączonymi obiektami towarzyszącymi klas *Emission, Description, ProgrammesFeature* i *Feature*.

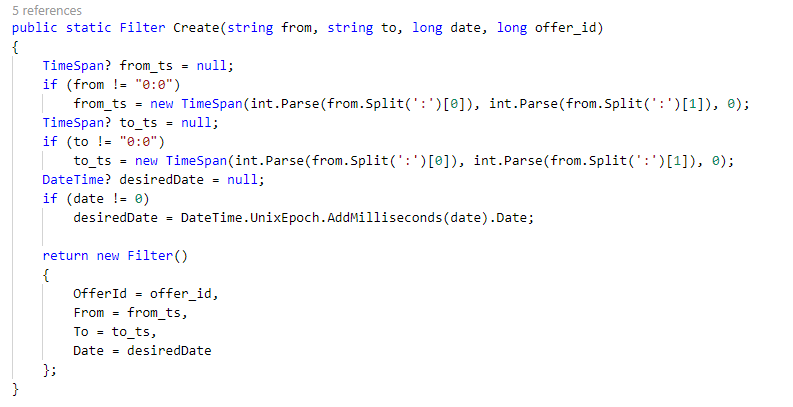
Po przejściu przez wszystkie programy w XMLu, powstanie lista obiektów klasy *Programme* zawierająca wszystkie nadawane programy. Lista jest potem zapisywana w bazie danych.

Przetwarzanie kanałów jest wykonywane w podobny, choć nieco prostszy sposób. 

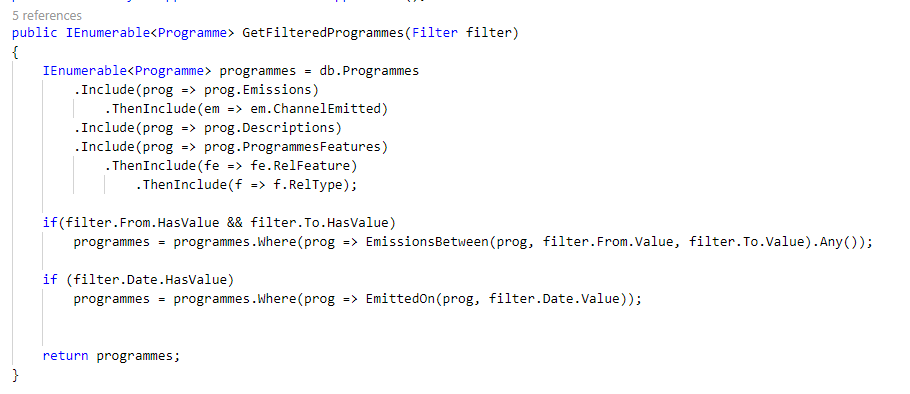
Dla wpisu zawierającego kanał, aplikacja sprawdza czy już istnieje w bazie danych, jeżeli nie to dodaje go do listy kanałów i na koniec zapisuje listę w bazie.

* + - 1. **Filtrowanie**

Filtrowanie na podstawie daty, godziny i oferty kanałów jest realizowane w klasie *ProgrammeService*. Za utworzenie filtra z przekazanych parametrów odpowiada metoda *Create*



Przygotowany filtr wykorzystuje metoda *GetFilteredProgrammes* z klasy odpowiedzialna za pobieranie listy programów z bazy danych

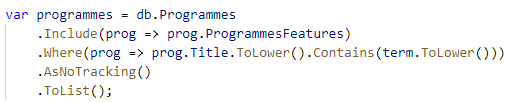


* + - 1. **Wyszukiwanie**

Za wyszukiwanie odpowiada endpoint *api/Programmes* wywoływany z metodą GET i parametrem żądania *search*. Po jego uruchomieniu, aplikacja wywołuje trzy metody – *GetWithSearchTerm* (odpowiedzialną za wyszukiwanie programów na podstawie tytułu), oraz *GetFeatures* i *GetWithFeature* (odpowiedzialne za wyszukiwanie na podstawie nazw cech). Wynik wyszukiwania jest składany, w pierwszej kolejności uwzględniane jest wyszukiwanie na podstawie tytułu.

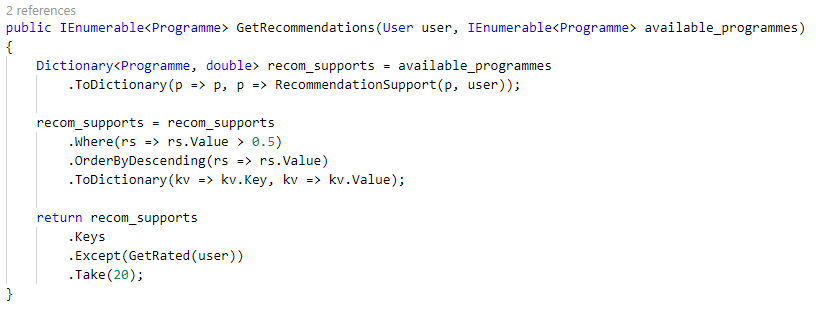


Podstawą dopasowania tytułu lub nazwy jest zawartość dokładnej treści szukanego hasła w tytule/nazwie.

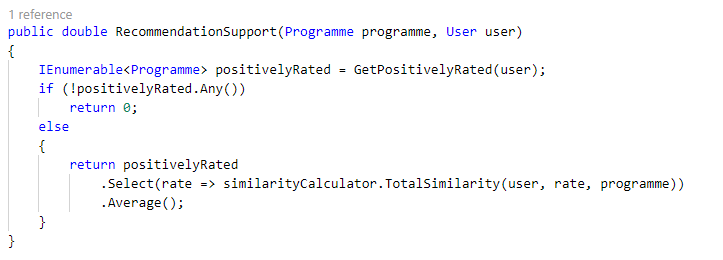


* + - 1. **Wyznaczanie rekomendacji**

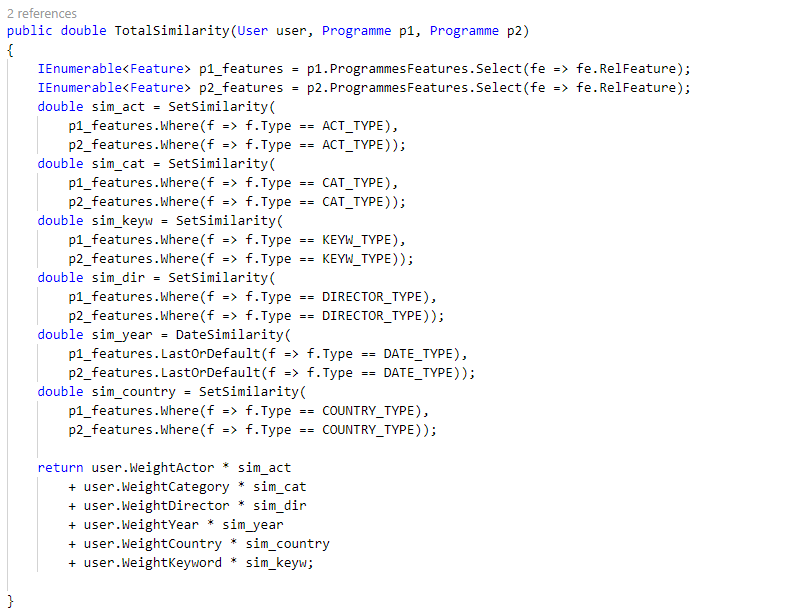
Za wyznaczanie rekomendacji użytkownikom odpowiada klasa *RecommendationService* oraz pomocnicza klasa wyznaczająca podobieństwa ­– *SimilarityCalculator*. Bezpośrednio wywoływaną metodą jest *GetRecommendations*, przyjmująca jako parametr dane użytkownika i listę dostępnych programów



Efektem działania jest lista co najwyżej 20 programów posortowanych wg średniego podobieństwa do programów, które użytkownik wcześniej oznaczył jako będące w jego guście. Średnie podobieństwo określa metoda *RecommendationSupport*:



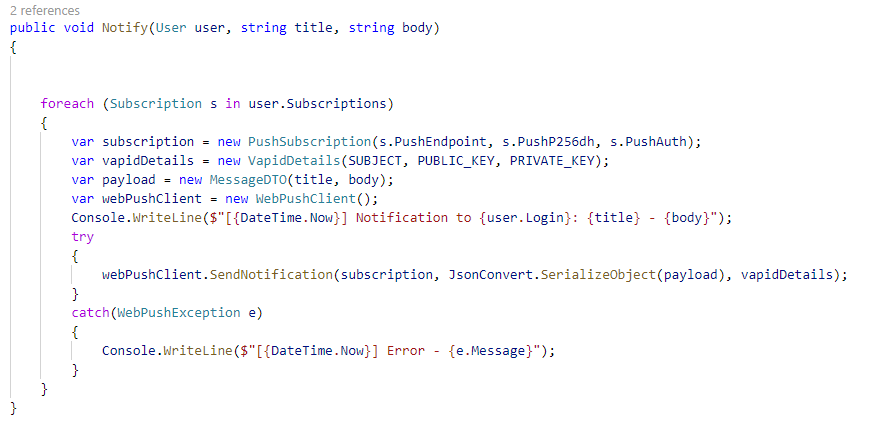
*RecommendationSupport* z kolei korzysta z metod określających podobieństwo programów zdefiniowanych w *SimilarityCalculator*, pierwszą z nich jest *TotalSimilarity*, zwracająca średnią ważoną wg wag charakterystycznych dla użytkownika.



Podstawy matematyczne przedstawionej tu logiki zostały omówione w rozdziale 3.

* + - 1. **Przekazywanie powiadomień**

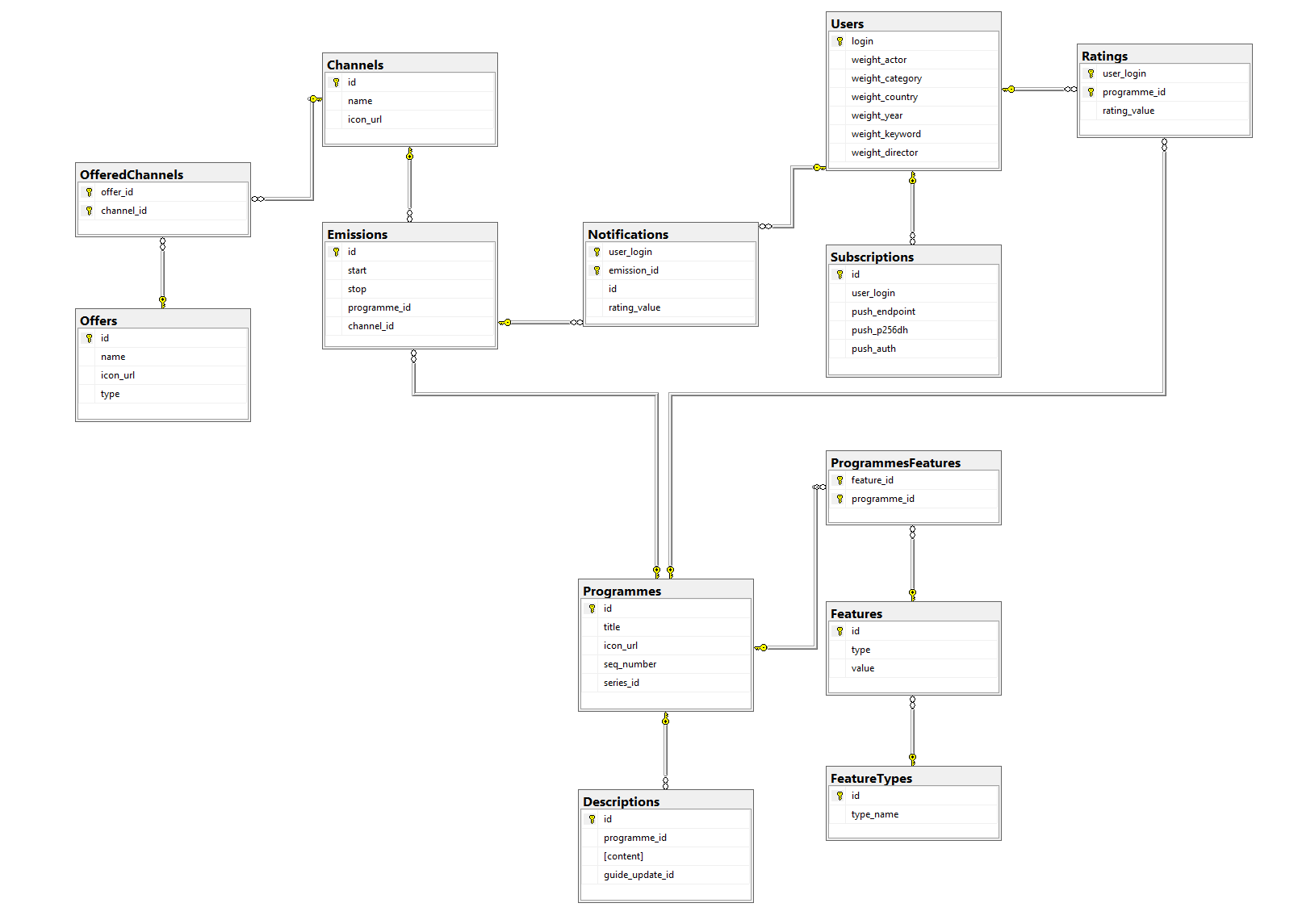
Do przekazywania powiadomień wykorzystano bibliotekę kliencką *Push API* – *WebPush*. Pozwala ona zamknąć mechanizm interakcji z zewnętrznym API w celu wysyłania powiadomień w jednej metodzie nazwanej *Notify*. Przyjmuje ona użytkownika i dla każdej z jego subskrypcji wysyła odpowiednie żądanie HTTP pod wskazany adres.



Logika dotycząca sprawdzania jakie powiadomienia należy wysłać w danym momencie jest wystawiona jako endpoint API i uruchamiana cyklicznie co 15 minut. Pozwala to również na ręczne uruchomienie sprawdzania do celów testowych. Raz przekazane powiadomienia są usuwane z bazy (pozytywna ocena zostaje na potrzeby rekomendacji)

## Baza danych

* + 1. Przechowywane dane



1. Wygenerowany przez SQL Server Management Studio schemat bazy danych

Baza danych aplikacji przechowuje informacje dotyczące poszczególnych programów telewizyjnych w tabelach *Programmes, Emissions, Descriptions* i *ProgrammesFeatures*. Tabela *Features* zawiera poszczególne cechy – nazwiska aktorów, reżyserów i scenarzystów. Tabela *Ratings* zawiera oceny wystawione przez użytkowników programom. Dostępne wartości ocen:

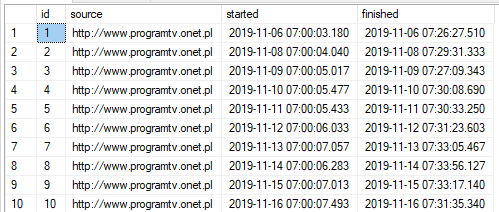
* -1 – *„nie podobał mi się”* – podobne programy będą rekomendowane z mniejszą częstotliwością
* 0 – *„nie interesuje mnie”* – program będzie ignorowany w procesie wyznaczania rekomendacji
* +1 – *„podobał mi się” ­–* aplikacja będzie rekomendowała podobne programy, ale pomijała ten konkretny (użytkownik już go oglądał, więc spodziewamy się że nie chce oglądać go po raz kolejny)
* Dodatkową możliwością jest ocena *„Chcę obejrzeć”*, która dla celów rekomendacji traktowana jest jako pozytywna, a dodatkowo użytkownik otrzyma powiadomienie przed planowaną emisją

Tabele odzwierciedlają klasy obiektów domenowych opisane wcześniej w rozdziałach 4.3.2 oraz 5.1.1

Baza jest aktualizowana codziennie i przechowuje program dla 156 kanałów na 4 najbliższe dni. Z każdą aktualizacją usuwane są dane dotyczące przeszłych emisji oraz dane programów bez określonych godzin emisji i ocen użytkowników. Programy wcześniej ocenione przechowywane są bezterminowo i wykorzystywane do wyznaczania rekomendacji dla danego użytkownika. Łącznie daje to ok.:

* 25 tys. rekordów dotyczących programów
* 15 tys. rekordów dotyczących godzin emisji
* 24 tys. rekordów dotyczących cech

Tabela *GuideUpdates* pozwala zweryfikować poprawne zakończenie procesów aktualizacji programu, oraz oszacować czas jego trwania. W ciągu 11 dni (5 – 11 listopada 2019) aktualizacja przebiegała następująco:



Jak wynika z tabeli, przetwarzanie pliku z programem o rozmiarze ok. 27MB trwa średnio ok. 30 minut.

## Klient

* + 1. **Organizacja kodu**

Kod aplikacji klienckiej jest podzielony na 2 główne foldery: *containers* i *shared*. Folder *containers* zawiera główne ekrany aplikacji, dla każdego z nich pliki określające odpowienio:

* .component.html – układ elementów na ekranie
* .component.scss – specyficzne dla danego widoku style
* .component.ts – używane przez widok dane i wewnętrzna logikę
* .component.spec.ts – testy jednostkowe dla widoku
* .service.ts – logika odpowiadająca za komunikację z API
* .service.spec.ts – testy jednostkowe dla komunikacji z API



Elementy współdzielone są zdefiniowane w folderze *shared*, należą do nich serwisy odpowiedzialne za funkcjonalność taką jak:

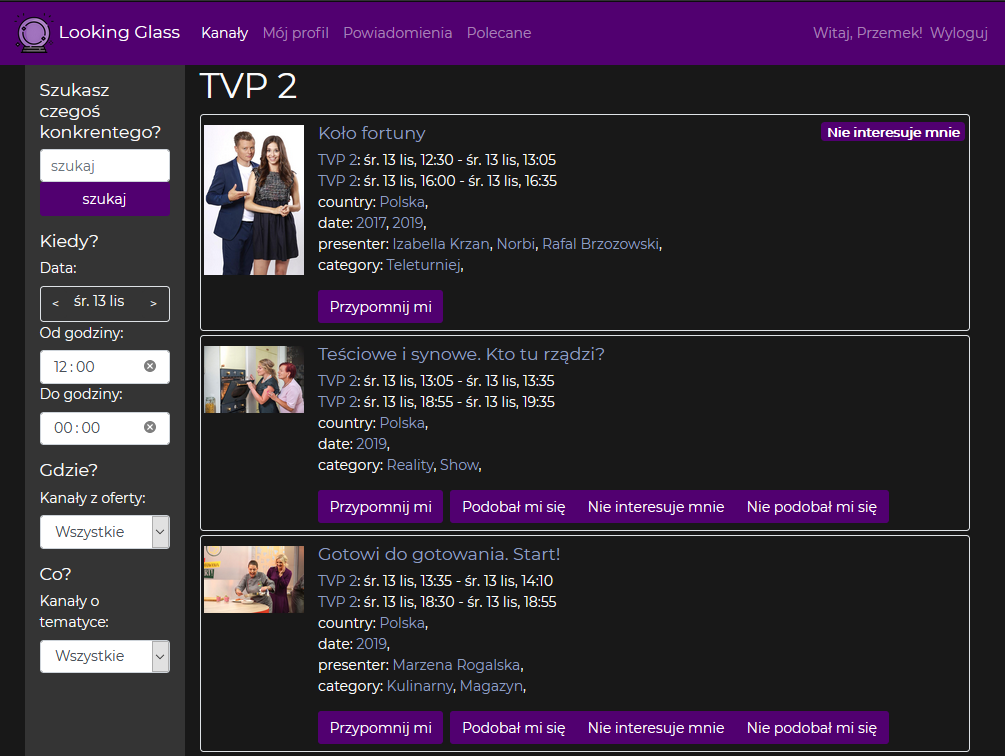
* Wyświetlanie powiadomień push
* Przechowywanie obecnego użytkownika

Poza tym, do współdzielonych elementów zaliczają się szablony elementów widoku, czyli:

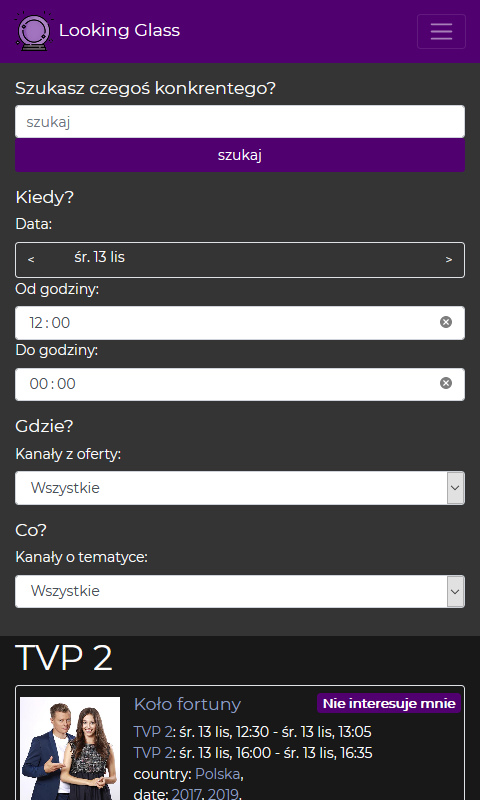
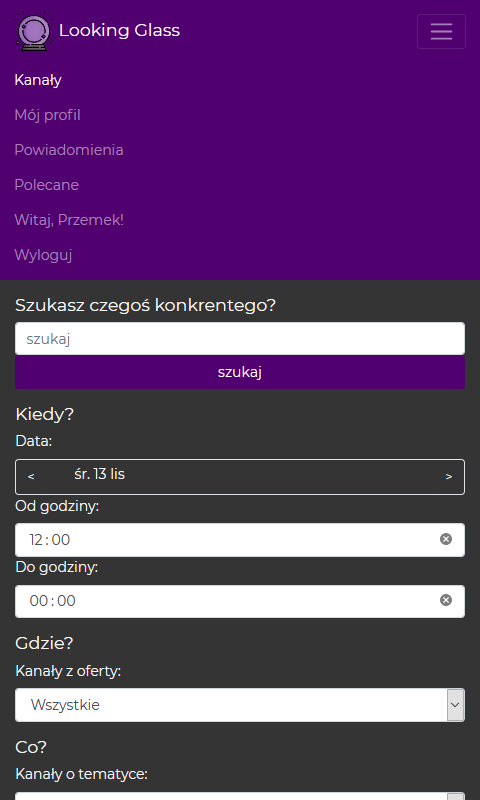
* Szablon pozycji na liście programów
* Boczne menu pozwalające na określenie filtrów
* Pasek nawigacyjny
  + 1. **Główne widoki**
       1. **Układ widoku i responsywność**

Jednym z wymagań jest dostępność aplikacji również na platformach mobilnych, zatem istotna jest responsywność układu elementów oraz styli CSS. Za reakcję na rozmiar ekranu odpowiada biblioteka styli *Bootstrap*.

Na urządzeniach typu desktop dostępna do wykorzystania przestrzeń pozioma jest stosunkowo duża, w związku z czym elementy kontrolne dla filtrów oraz pole wyszukiwarki umieszczone są po lewej stronie, równolegle z zawartością

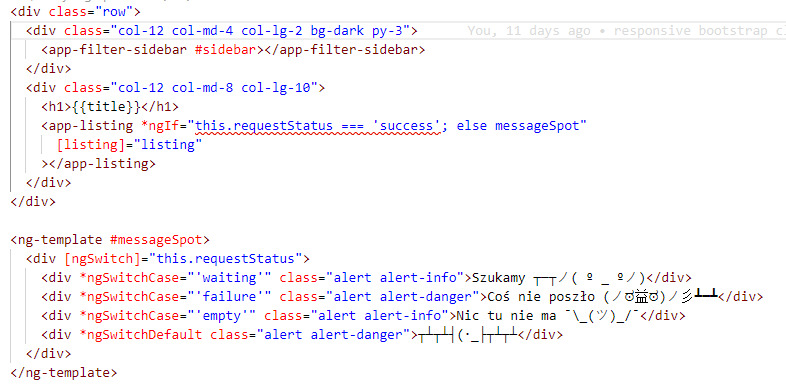


Na urządzeniach mobilnych przestrzeń pozioma jest mocno ograniczona, przez co, boczne menu z filtrami znajduje się na górze ekranu, a górny pasek można zwinąć – wszystko w celu pozostawienia jak największej przestrzeni na właściwą zawartość. Efektem jest strona, która zachowuje tą samą funkcjonalność, a jednocześnie jest wyświetlana w sposób, który wymaga przewijania tylko w kierunku pionowym, ponieważ cała szerokość strony mieści się na węższym ekranie.



* + - 1. **Implementacja**

Opisany wcześniej ekran zawierający listę programów nadawanych na określonym kanale – zawarty w folderze *channel-listing* został zaimplementowany następująco. Układ elementów zdefiniowano w pliku channel-listing.component.html

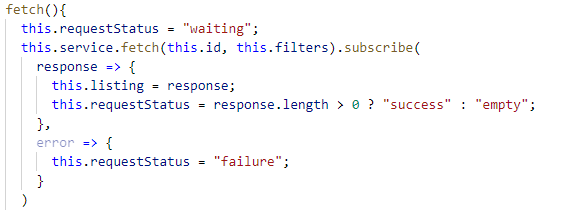


Układ jest podzielony na dwa bloki:

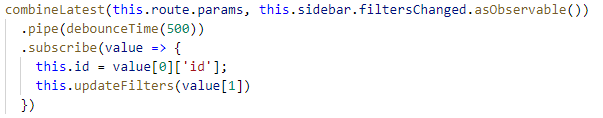
* Filtry
* Lista programów jeżeli są dostępne lub odpowiednia informacja o błędzie w przeciwnym wypadku.

Logika związana z pobieraniem i wyświetlaniem informacji znajduje się w pliku *channel-listing.component.ts*.

W tym przypadku jest to wywołanie żądania do API w reakcji na zmianę filtrów lub wyświetlanego kanału (za samo żądanie odpowiada serwis *channel-listing.service.ts*), oraz ustawienie odpowiedniej flagi w zależności od statusu żądania (czy się powiodło, czy oczekujemy na informację itd.). Wywołanie żądania następuje z użyciem metody fetch()



Reakcja na zmiany filtrów lub numeru kanału jest zaimplementowana z użyciem często używanej w aplikacjach Angulara biblioteki RxJS implementującej wzorzec obserwatora.



RxJS pozwala w krótkim fragmencie kodu obserwować zarówno zmiany filtrów jak i ścieżki adresu URL w przeglądarce i reagować na te zmiany w czasie rzeczywistym. Użyty operator *debounceTime* pozwala uniknąć odpytywania API po każdej zmianie, zamiast tego żądanie jest wysyłane dopiero w momencie kiedy filtry przestaną się zmieniać (pozostaną bez zmian przez 500 milisekund).

Implementacja serwisu korzysta z angularowego klienta HTTP. Żądanie zawiera 3 główne elementy, w tym przypadku:

* Numer kanału
* Ustawione przez użytkownika filtry
* Nazwa użytkownika, jeżeli jest dostępna – pochodzi ze współdzielonego między widokami serwisu przechowującego dane użytkownika



# Testy

## Jednostkowe serwera – xUnit

* + 1. **Wstęp**

Testy serwera obejmują głównie klasy serwisów – są to jedyne klasy wykonujące jakąkolwiek logikę, którą można testować. Klasy obiektów domenowych i DTO składają się głównie z właściwości, funkcjonalności wbudowanej w język *C#*, podczas gdy klasy migracji zostały wygenerowane automatycznie przez *EntityFramework Core*. Pokrycie testami obiektów domenowych jest zagwarantowane z automatu, ponieważ są one używane w testach serwisów.

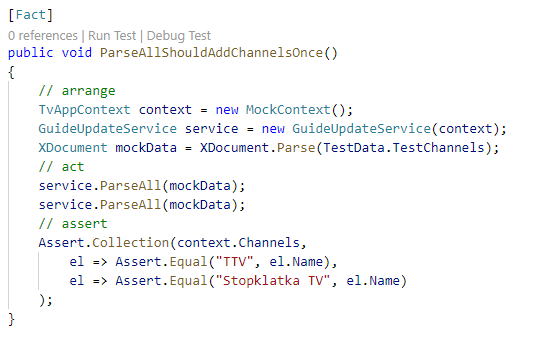
W przypadku gdy test potrzebuje sprawdzić, czy dane są zapisywane do bazy danych, wykorzystano bazę typu *In Memory*, również dostarczoną w ramach *EntityFramework Core*. Dane zawarte w takiej bazie są przechowywane w pamięci i tracone po zakończeniu działania programu (co w oczywisty sposób jest niepraktyczne w typowym zastosowaniu produkcyjnym, ale bardzo wygodne do celu testów).

* + 1. **Przykładowe testy**
       1. Wartości zmiennych



Standardowy test, sprawdzający czy kalkulator podobieństwa zwróci wartość **1** dla dwóch identycznych programów.

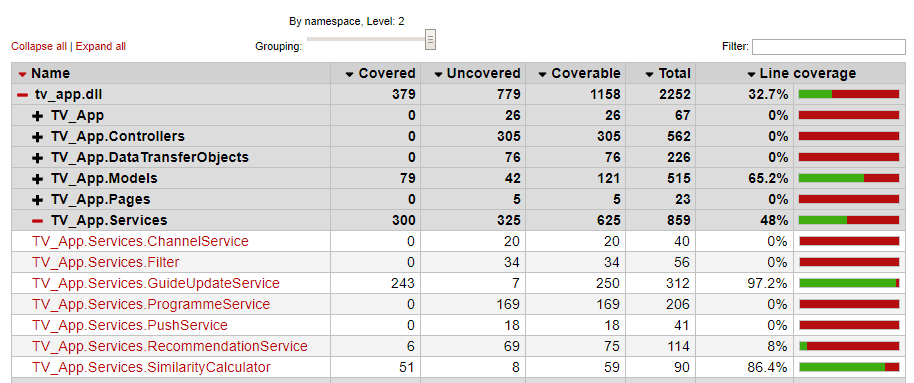
* + - 1. Zapis do bazy danych



Test parsowania kanałów w trakcie procesu ładowania do bazy korzysta z bazy *In Memory* (za pośrednictwem klasy *MockContext*). Instrukcja *Assert* dotyczy sprawdzenia, czy zapisane w bazie kanały to te znajdujące się w przykładowym XMLu.

* + 1. **Pokrycie kodu**

Analizę pokrycia kodu zapewniają dwa pakiety Nuget – *CodeCoverage* i *ReportGenerator*. Efektem jest raport w formie pliku HTML. Ponieważ testowanie narzędzi zewnętrznych mija się z celem, z raportu wyłączono klasy migracji generowane przez *EntityFramework Core* – testowanie ich byłoby testowaniem biblioteki, a nie kodu napisanego na potrzeby aplikacji.

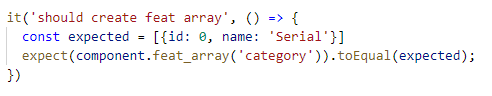


## Jednostkowe klienta – Jasmine i Karma

* + 1. **Wstęp**

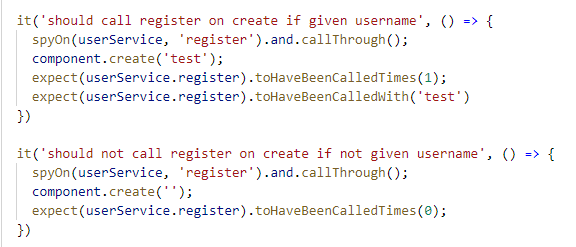
Testy jednostkowe po stronie klienta obejmują logikę wykonywaną po stronie klienta. Narzędzia Jasmine i Karma pozwalają testować wartości zmiennych, zliczać ilość wywołań poszczególnych metod, a także pozwalają na wgląd w renderowany kod HTML, co pozwala sprawdzać jaka wartość pojawi się na ekranie.

* + 1. **Przykładowe testy**
       1. Wartości zmiennych



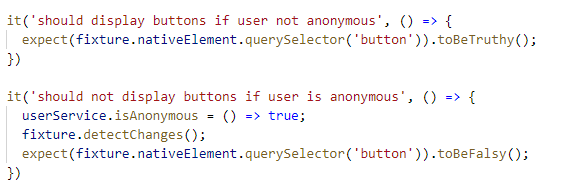
Test sprawdza, czy tworzona przez metodę *feat\_array* lista cech programu ma właściwy format

* + - 1. Wywołania metod



Pokazane testy sprawdzają, czy metoda *create* wywołuje odpowiadającą jej metodę z serwisu przy poprawnych danych.

* + - 1. Renderowany kod HTML



Testy sprawdzają, czy w elementach listy programów pojawiają się przyciski. Powinny się pojawiać, jeżeli użytkownik jest zalogowany.

* + 1. **Pokrycie kodu**

Analizę pokrycia kodu po stronie klienta zapewnia narzędzie Istanbul, generujące odpowiedni raport po uruchomieniu testów. Wykonano łącznie 53 testy jednostkowe, zapewniające pokrycie ok. 80% kodu aplikacji.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Folder** | **Instrukcje** | | **Linie** | |
| [src](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\index.html) | 100% | 3/3 | 100% | 3/3 |
| [src/app](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\index.html) | 100% | 5/5 | 100% | 4/4 |
| [src/app/containers/all-channels](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\all-channels\index.html) | 86.36% | 19/22 | 84.21% | 16/19 |
| [src/app/containers/channel-listing](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\channel-listing\index.html) | 88.24% | 30/34 | 85.19% | 23/27 |
| [src/app/containers/details](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\details\index.html) | 88.24% | 30/34 | 85.19% | 23/27 |
| [src/app/containers/feature](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\feature\index.html) | 88.24% | 30/34 | 85.19% | 23/27 |
| [src/app/containers/notifications](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\notifications\index.html) | 89.66% | 26/29 | 86.36% | 19/22 |
| [src/app/containers/profile](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\profile\index.html) | 89.66% | 26/29 | 86.36% | 19/22 |
| [src/app/containers/recommendations](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\recommendations\index.html) | 88.89% | 24/27 | 86.36% | 19/22 |
| [src/app/containers/search](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\containers\search\index.html) | 90% | 27/30 | 87.5% | 21/24 |
| [src/app/shared/components/filter-sidebar](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\filter-sidebar\index.html) | 70% | 35/50 | 68.89% | 31/45 |
| [src/app/shared/components/footer](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\footer\index.html) | 100% | 7/7 | 100% | 5/5 |
| [src/app/shared/components/listing](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\listing\index.html) | 81.82% | 9/11 | 80% | 8/10 |
| [src/app/shared/components/listing-element](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\listing-element\index.html) | 88.37% | 38/43 | 86.84% | 33/38 |
| [src/app/shared/components/login](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\login\index.html) | 71.05% | 27/38 | 71.88% | 23/32 |
| [src/app/shared/components/navbar](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\navbar\index.html) | 87.5% | 14/16 | 86.67% | 13/15 |
| [src/app/shared/components/offer-picker](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\offer-picker\index.html) | 85.71% | 12/14 | 81.82% | 9/11 |
| [src/app/shared/components/programmedetails](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\components\programmedetails\index.html) | 93.55% | 29/31 | 92% | 23/25 |
| [src/app/shared/services](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\app\shared\services\index.html) | 60.87% | 42/69 | 59.65% | 34/57 |
| [src/environments](file:///C:\Users\Przemek\Documents\repos\imanengineer\client\coverage\src\environments\index.html) | 100% | 1/1 | 100% | 1/1 |
| **Razem** | **82.35%** | **434/527** | **80.28%** | **350/436** |

## Automatyczne end-to-end – Protractor

# Instalacja

## Założenia

Gotowe obrazy są przechowywane w rejestrze obrazów dockerowych, w tym przypadku publicznie dostępnym [*https://hub.docker.com/*](https://hub.docker.com/) (istnieje możliwość uruchamiania rejestru na prywatnym serwerze, ale w tym przypadku nie widzę takiej potrzeby). Kompletna procedura wdrożenia budowanej w ten sposób aplikacji składa się z czterech kroków:

1. Budowa obrazów – wywołanie na stacji deweloperskiej komendy   
   *docker-compose build*
2. Publikacja obrazów do rejestru – również ze stacji deweloperskiej, komendą   
   *docker-compose push*
3. Pobranie pliku konfiguracyjnego *docker-compose.yml* na serwer produkcyjny (przechowywany razem z kodem źródłowym aplikacji w repozytorium Git)
4. Pobranie obrazów i uruchomienie na serwerze produkcyjnym – wywołaniem komendy  
   *docker-compose up*

## Zastosowanie

* + 1. Serwer

Aplikacja została stworzona w technologii .NET Core, ponieważ pozwala to, w przeciwieństwie do poprzedniej platformy Microsoftu, .NET Framework, na uruchamianie jej w systemie Linux, a zatem również wewnątrz obrazów platformy Docker.

Kontener zawierający część serwerową jest opisany w następujący sposób:



Dockerfile opisujący część serwerową jest dwuetapowy

* pierwszy etap korzysta z obrazu *mcr.microsoft.com/dotnet/core/sdk:3.0-bionic* - opartego na Ubuntu i zawierającego wszystkie zależności, potrzebne do zbudowania aplikacji .NET Core
* drugi etap korzysta z innego obrazu, *mcr.microsoft.com/dotnet/core/aspnet:3.0-bionic* - również opartego na Ubuntu, ale zawierającego tylko te zależności, które potrzebne są do uruchomienia gotowej aplikacji
  + 1. Klient

Artefaktem powstałym w wyniku kompilacji kodu Angulara (podobnie jak Reacta, Vue czy innych front-endowych frameworków) jest zestaw plików html, css i js (oraz dodatkowych zasobów – ikon, dźwięków lub plików konfiguracyjnych). Te pliki muszą być obsłużone po stronie serwera i przesłane przeglądarce użytkownika na żądanie. Obecnie najpopularniejszym serwerem do obsługi statycznych plików jest NGINX, z którego korzysta kontener obsługujący część kliencką.

Kontener zawierający część kliencką jest opisany w następujący sposób:



Dockerfile opisujący aplikację kliencką również jest dwuetapowy

* pierwsza część zawiera środowisko node.js i służy do pobrania wszystkich zależności i zbudowania docelowych artefaktów
* druga część to sam serwer NGINX z odpowiednią konfiguracją i plikami do obsługi

## Komunikacja

Zgodnie z dobrymi praktykami, kontener z aplikacją powinien zawierać tylko i wyłącznie tą jedną aplikację, jeżeli wymagane są inne procesy, powinny one działać w osobnych kontenerach.

Za budowę i uruchamianie odpowiednich kontenerów odpowiada usługa docker-compose będąca częścią całej platformy Docker. Pozwala ona zdefiniować plik konfiguracyjny YAML, określający w jaki sposób mają być budowane poszczególne kontenery, w jaki sposób mają być uruchamiane i jak powinny się ze sobą komunikować. Na opisywaną aplikację składają się 4 kontenery, wszystkie zdefiniowane w pliku   
*docker-compose.yml*

Poza dwoma opisywanymi wcześniej obrazami server i client, aplikacja korzysta z obrazu zawierającego serwer bazy danych MS-SQL oraz opracowany przez grupę linuxserver.io, obraz zawierający program pobierający programy telewizyjne, WebGrab+.

# Podsumowanie

* Ilość wyprodukowanego kodu, w tym kod wygenerowany automatycznie (m. in. przez *EF Core*, na podstawie [*https://www.npmjs.com/package/sloc*](https://www.npmjs.com/package/sloc)) :
  + Serwer: 8969 linii w 74 plikach
  + Klient: 3029 linii w 110 plikach
* 2 GB pamięci RAM to teoretycznie minimalne wymagania dla SQL Servera (przy mniejszej w ogóle nie da się uruchomić kontenera), natomiast wydają się niewystarczające dla stabilnego działania. Przy obecnej konfiguracji (z 2GB RAMu) zdarzają się przypadki, gdy baza przestaje odpowiadać i jedyne prawdopodobne wyjaśnienie znajduje się po stronie sprzętowej.

# Bibliografia

1. **Wirtualna Polska.** Program TV w WP. [Online] [Zacytowano: 16 01 2018.] https://tv.wp.pl/.

2. **Onet.pl.** Program TV - Onet.pl. [Online] [Zacytowano: 26 11 2019.] https://programtv.onet.pl/.

3. **Interia.pl.** Program TV w Interia.pl. [Online] [Zacytowano: 20 01 2018.] https://programtv.interia.pl.

4. **CableTV.** Which TV Show Should I Binge Watch Next? *Silnik rekomendacji CableTV.* [Online] [Zacytowano: 19 01 2018.] https://www.cabletv.com/what-to-watch.

5. **TasteDive.** TasteDive. [Online] [Zacytowano: 19 01 2018.] https://tastedive.com.

6. **Wirtualna Polska.** WP Pilot. [Online] [Zacytowano: 20 01 2018.] https://pilot.wp.pl/tv/.

7. **ipla.tv.** ipla.tv. [Online] [Zacytowano: 21 01 2018.] https://www.ipla.tv/start.

8. **Filmweb.** Filmweb. [Online] [Zacytowano: 13 04 2019.] https://www.filmweb.pl/.

9. **DigitalOcean.** Droplets on DigitalOcean. [Online] [Zacytowano: 17 08 2019.] https://www.digitalocean.com/products/droplets/.

10. **NGINX.** *Strona internetowa projektu NGINX.* [Online] https://www.nginx.com/.

11. **Docker.** Dokumentacja platformy Docker. [Online] [Zacytowano: 20 05 2019.] https://docs.docker.com/.

12. **Microsoft.** Dokumentacja MS SQL. *MSDN.* [Online] [Zacytowano: 27 05 2019.] https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15.

13. **Docker.** Przegląd dostępnych obrazów baz danych. *Dockerhub.* [Online] [Zacytowano: 21 05 2019.] https://hub.docker.com/search?q=&type=image&category=database&operating\_system=linux.

14. **Microsoft.** Dokumentacja ASP.NET Core. *MSDN.* [Online] [Zacytowano: 20 09 2019.] https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.0.

15. —. Dokumentacja EntityFramework Core. *MSDN.* [Online] [Zacytowano: 29 08 2019.] https://docs.microsoft.com/en-us/ef/core/.

16. **Angular.** Dokumentacja frameworka. [Online] [Zacytowano: 2019 08 28.] https://angular.io/docs.

17. **Mozilla Developer Network.** Dokumentacja Push API. *Mozilla Developer Network.* [Online] [Zacytowano: 2019 10 16.] https://developer.mozilla.org/en-US/docs/Web/API/Push\_API.

18. **Microsoft.** Dokumentacja SignalR. *MSDN.* [Online] [Zacytowano: 20 07 2019.] https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.0.

19. **Google.** Dokumentacja Firebase. [Online] [Zacytowano: 2019 10 25.] https://firebase.google.com/docs/cloud-messaging.

20. —. Wprowadzenie do PWA. [Online] [Zacytowano: 29 10 2019.] https://developers.google.com/web/progressive-web-apps.

21. **Straaten, Jan van.** *Strona internetowa projektu WebGrabPlus+.* [Online] http://webgrabplus.com/documentation/quick-start.

Dostępne kanały – spis

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Kanał | Adres programu | Dostawca | | | | | |
| DVB-T | Cyfrowy Polsat | Multimedia | nc+ | UPC | Vectra |
| TVP 1 | /program-tv/tvp-1-321 | + | + | + | + | + | + |
| TVP 2 | /program-tv/tvp-2-323 | + | + | + | + | + | + |
| TVP 3 | /program-tv/tvp-3-172 | + | + | + | + | + | + |
| POLSAT | /program-tv/polsat-38 | + | + | + | + | + | + |
| TVN | /program-tv/tvn-357 | + | + | + | + | + | + |
| TV 4 | /program-tv/tv-4-360 | + | + | + | + | + | + |
| TV Puls | /program-tv/tv-puls-332 | + | + | + | + | + | + |
| TVN 7 | /program-tv/tvn-7-326 | + | + | + | + | + | + |
| PULS 2 | /program-tv/puls-2-439 | + |  | + | + | + | + |
| TV 6 | /program-tv/tv-6-429 | + | + | + |  |  |  |
| Eska TV | /program-tv/eska-tv-114 | + | + | + |  | + | + |
| Polo TV | /program-tv/polo-tv-135 | + | + | + | + | + | + |
| ATM Rozrywka | /program-tv/atm-rozrywka-287 | + | + | + | + | + | + |
| TV Trwam | /program-tv/tv-trwam-108 | + |  | + |  | + | + |
| Stopklatka TV | /program-tv/stopklatka-tv-185 | + | + | + | + | + | + |
| Fokus TV | /program-tv/fokus-tv-46 | + | + | + | + | + | + |
| TVP ABC | /program-tv/tvp-abc-182 | + | + | + | + | + | + |
| TVP Kultura | /program-tv/tvp-kultura-477 | + | + | + | + | + | + |
| TVP Historia | /program-tv/tvp-historia-74 | + | + | + | + | + | + |
| TVP Sport | /program-tv/tvp-sport-40 | + | + | + | + | + | + |
| TVP Info | /program-tv/tvp-info-462 | + | + | + | + |  |  |
| TVP Rozrywka | /program-tv/tvp-rozrywka-159 | + | + | + | + | + | + |
| TTV | /program-tv/ttv-33 |  | + |  |  | + | + |
| STARS.TV | /program-tv/stars-tv-149 |  | + | + | + | + | + |
| ZOOM TV | /program-tv/zoom-tv-526 | + | + | + | + | + | + |
| POLSAT News | /program-tv/polsat-news-100 |  | + | + | + | + | + |
| POLSAT 2 | /program-tv/polsat-2-327 |  | + | + | + | + | + |
| POLSAT Play | /program-tv/polsat-play-21 |  | + | + | + | + |  |
| POLSAT Café | /program-tv/polsat-caf-110 |  | + | + | + | + | + |
| POLSAT Sport | /program-tv/polsat-sport-334 |  | + | + |  | + | + |
| POLSAT Sport Extra | /program-tv/polsat-sport-extra-485 |  | + | + |  | + | + |
| Eurosport 1 | /program-tv/eurosport-1-93 |  | + | + | + | + | + |
| Eurosport 2 | /program-tv/eurosport-2-76 |  | + | + | + | + | + |
| Extreme Sports Channel | /program-tv/extreme-sports-channel-231 |  | + |  | + | + | + |
| Paramount Channel HD | /program-tv/paramount-channel-hd-65 |  | + | + | + | + | + |
| Disco Polo Music | /program-tv/disco-polo-music-191 |  | + | + | + | + |  |
| Food Network HD | /program-tv/food-network-hd-240 |  | + | + | + | + | + |
| POLSAT Film | /program-tv/polsat-film-123 |  | + | + | + | + | + |
| HBO | /program-tv/hbo-23 |  | + | + | + | + | + |
| HBO2 | /program-tv/hbo2-24 |  | + | + | + | + | + |
| HBO3 | /program-tv/hbo-3-25 |  | + | + | + | + | + |
| Cinemax | /program-tv/cinemax-59 |  | + | + | + | + | + |
| Cinemax2 | /program-tv/cinemax2-58 |  | + | + | + | + | + |
| FilmBox | /program-tv/filmbox-84 |  | + | + | + | + | + |
| FilmBox Family | /program-tv/filmbox-family-103 |  | + | + | + | + | + |
| AXN | /program-tv/axn-249 |  | + | + | + | + | + |
| AXN White | /program-tv/axn-white-272 |  | + | + | + | + | + |
| AXN Black | /program-tv/axn-black-271 |  | + | + | + | + | + |
| FOX Comedy | /program-tv/fox-comedy-75 |  | + |  | + | + | + |
| Comedy Central | /program-tv/comedy-central-63 |  | + | + | + | + | + |
| Comedy Central Family | /program-tv/comedy-central-family-61 |  | + | + |  | + |  |
| POLSAT Romans | /program-tv/polsat-romans-173 |  | + | + | + |  |  |
| CBS Europa | /program-tv/cbs-europa-317 |  | + | + | + | + |  |
| Kino Polska | /program-tv/kino-polska-324 |  | + |  |  |  |  |
| TNT | /program-tv/tnt-72 |  | + | + | + | + | + |
| ID | /program-tv/id-117 |  | + | + | + | + | + |
| FOX | /program-tv/fox-127 |  | + |  | + | + | + |
| TVP Seriale | /program-tv/tvp-seriale-130 |  | + | + | + | + | + |
| 13 Ulica | /program-tv/13-ulica-316 |  | + | + | + |  | + |
| FilmBox Action | /program-tv/filmbox-action-451 |  | + | + | + | + | + |
| FilmBox Arthouse | /program-tv/filmbox-arthouse-183 |  | + | + | + | + | + |
| AMC | /program-tv/mgm-333 |  | + | + | + |  |  |
| Lifetime | /program-tv/lifetime-50 |  | + |  | + | + |  |
| TVN 24 | /program-tv/tvn-24-347 |  | + | + | + | + | + |
| POLSAT News 2 | /program-tv/polsat-news-2-471 |  | + | + | + | + | + |
| TVN 24 Biznes i Świat | /program-tv/tvn-24-biznes-i-swiat-6 |  | + |  |  |  |  |
| Superstacja | /program-tv/superstacja-69 |  | + | + | + |  | + |
| TV Republika | /program-tv/tv-republika-18 |  | + | + | + | + | + |
| POLSAT Viasat Explore HD | /program-tv/polsat-viasat-explore-hd-82 |  | + | + | + | + |  |
| POLSAT Viasat History HD | /program-tv/polsat-viasat-history-hd-71 |  | + | + | + | + | + |
| BBC Earth | /program-tv/bbc-earth-274 |  | + |  | + | + | + |
| National Geographic Channel | /program-tv/national-geographic-channel-32 |  | + |  | + | + | + |
| Discovery Channel | /program-tv/discovery-channel-202 |  | + | + | + | + | + |
| Nat Geo Wild | /program-tv/nat-geo-wild-77 |  | + |  | + | + | + |
| Discovery Science | /program-tv/discovery-science-53 |  | + | + | + | + | + |
| Discovery Historia | /program-tv/discovery-historia-54 |  | + | + | + | + | + |
| Travel Channel | /program-tv/travel-channel-201 |  | + | + | + | + | + |
| HISTORY | /program-tv/history-91 |  | + |  | + | + | + |
| CBS Reality | /program-tv/cbs-reality-318 |  | + |  | + | + |  |
| TVN Turbo | /program-tv/tvn-turbo-346 |  | + | + | + | + | + |
| TVS | /program-tv/tvs-90 |  | + | + | + | + | + |
| Disney Channel | /program-tv/disney-channel-478 |  | + | + | + |  | + |
| POLSAT JimJam | /program-tv/polsat-jimjam-89 |  | + | + | + | + |  |
| Disney Junior | /program-tv/disney-junior-469 |  | + | + | + | + | + |
| Cartoon Network | /program-tv/cartoon-network-273 |  | + | + | + | + | + |
| Disney XD | /program-tv/disney-xd-235 |  | + | + | + | + | + |
| Nickelodeon | /program-tv/nickelodeon-42 |  | + | + | + | + | + |
| Nick Jr. | /program-tv/nick-jr-45 |  | + | + | + | + | + |
| BBC CBeebies | /program-tv/bbc-cbeebies-2 |  | + |  | + | + | + |
| TVN Style | /program-tv/tvn-style-472 |  | + | + | + | + | + |
| E! Entertainment | /program-tv/e-entertainment-73 |  | + | + | + | + |  |
| TLC | /program-tv/tlc-238 |  | + | + | + |  | + |
| BBC Brit | /program-tv/bbc-brit-275 |  | + |  | + | + | + |
| TVP Polonia | /program-tv/tvp-polonia-325 |  | + | + | + |  | + |
| AXN Spin | /program-tv/axn-spin-294 |  | + | + | + | + | + |
| FilmBox Extra HD | /program-tv/filmbox-extra-hd-86 |  | + |  | + |  |  |
| BBC HD | /program-tv/bbc-hd-261 |  | + |  | + | + | + |
| Sundance Channel | /program-tv/sundance-channel-237 |  | + | + | + | + |  |
| Romance TV | /program-tv/romance-tv-129 |  | + |  | + |  | + |
| H2 | /program-tv/h2-203 |  | + |  | + |  |  |
| TVN Fabuła | /program-tv/tvn-fabula-4 |  | + | + | + | + | + |
| Nat Geo People HD | /program-tv/nat-geo-people-hd-211 |  | + |  | + | + | + |
| Animal Planet HD | /program-tv/animal-planet-hd-284 |  | + | + | + | + | + |
| POLSAT Viasat Nature | /program-tv/polsat-viasat-nature-413 |  | + |  |  | + |  |
| Da Vinci Learning | /program-tv/da-vinci-learning-83 |  | + | + | + |  | + |
| MTV Rocks | /program-tv/mtv-rocks-344 |  | + | + | + | + |  |
| VH1 | /program-tv/vh1-9 |  | + | + | + | + |  |
| Kino Polska Muzyka | /program-tv/kino-polska-muzyka-426 |  | + | + | + |  | + |
| 4FUN.TV | /program-tv/4fun-tv-269 |  | + | + | + | + | + |
| 4FUN HITS | /program-tv/4fun-hits-283 |  | + | + | + | + | + |
| VOX | /program-tv/vox-418 |  | + |  |  | + | + |
| MTV Live HD | /program-tv/mtv-live-hd-105 |  | + | + | + | + | + |
| Arte | /program-tv/arte-253 |  | + |  |  |  |  |
| Tele 5 | /program-tv/tele-5-352 |  | + | + | + | + | + |
| Novela tv | /program-tv/novela-tv-461 |  | + | + |  |  |  |
| Water Planet | /program-tv/water-planet-415 |  | + | + |  |  |  |
| CBS Action | /program-tv/cbs-action-315 |  | + | + | + | + |  |
| Mango 24 | /program-tv/mango-24-113 |  | + | + | + | + | + |
| Eleven | /program-tv/eleven-208 |  | + | + | + | + | + |
| Eleven Sports | /program-tv/eleven-sports-212 |  | + | + | + | + | + |
| FightBox | /program-tv/fightbox-436 |  | + | + | + | + | + |
| Kuchnia+ | /program-tv/kuchnia-489 |  |  | + | + | + | + |
| Domo+ | /program-tv/domo-106 |  |  | + | + | + | + |
| CANAL+ | /program-tv/canal-246 |  |  | + | + | + | + |
| CANAL+ Film | /program-tv/canal-film-320 |  |  | + | + | + | + |
| CANAL+ Seriale | /program-tv/canal-seriale-293 |  |  | + | + | + | + |
| CANAL+ Family | /program-tv/canal-family-296 |  |  | + | + | + | + |
| CANAL+ Sport | /program-tv/canal-sport-14 |  |  | + | + | + | + |
| CANAL+ Sport 2 | /program-tv/canal-sport-2-15 |  |  | + | + | + | + |
| CANAL+ Discovery | /program-tv/canal-discovery-307 |  |  | + | + | + | + |
| Discovery Life | /program-tv/discovery-life-187 |  |  | + |  | + | + |
| Discovery Turbo Xtra | /program-tv/discovery-turbo-xtra-239 |  | + | + | + | + | + |
| Planete+ | /program-tv/planete-349 |  |  | + | + | + | + |
| Viasat History HD/Viasat Nature HD | /program-tv/viasat-history-hd-viasat-nature-hd-424 |  |  | + |  |  |  |
| Adventure | /program-tv/adventure-303 |  |  | + | + | + | + |
| MiniMini+ | /program-tv/minimini-236 |  |  | + | + | + | + |
| teleTOON+ | /program-tv/teletoon-232 |  |  | + | + | + | + |
| Top Kids | /program-tv/top-kids-225 |  |  | + |  | + | + |
| Boomerang | /program-tv/boomerang-270 |  |  | + |  |  | + |
| ducktv | /program-tv/ducktv-94 |  |  | + |  |  |  |
| nSport+ | /program-tv/nsport-19 |  |  | + | + | + | + |
| Motowizja | /program-tv/motowizja-178 |  |  | + | + | + | + |
| Fightklub | /program-tv/fightklub-78 |  |  | + | + | + | + |
| Sportklub | /program-tv/sportklub-29 |  |  | + |  |  |  |
| Ale kino+ | /program-tv/ale-kino-319 |  |  | + | + | + |  |
| Scifi Universal | /program-tv/scifi-universal-20 |  |  | + |  |  |  |
| Active Family | /program-tv/active-family-300 |  |  | + |  |  | + |
| Power TV | /program-tv/power-tv-176 |  |  | + | + | + |  |
| Nuta.TV | /program-tv/nuta-tv-214 |  |  | + | + | + |  |
| MTV Music | /program-tv/viva-polska-10 |  | + | + | + | + | + |
| Baby TV | /program-tv/baby-tv-285 |  |  |  | + |  |  |
| TVR | /program-tv/tvr-132 |  |  | + | + | + | + |