Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij

Karlo Marunić

Medijska knjižnica - MediaBase

Diplomski rad

Mentor: izv. prof. dr. sc. Marina Ivašić Kos prof. mat. i inf.

Rijeka, lipanj 2019. godine

Sadržaj

Sažetak

MediaBase je medijska knjižnica programirana u C# jeziku koja služi za sortiranje, filtriranje i pregledavanje video datoteka na korisnikovom računalu. Za izradu je korišteno radno okruženje Microsoft Visual Studio. U radu su opisane već postojeće medijske knjižnice Windows Media Player, KODI i PLEX, tehnologije i paketi korišteni za izradu ove aplikacije i programski kod zaslužan za izvođenje same aplikacije.

MediaBase is a media library programmed in C# language that serves to sort, filter and browse video files on a user's computer. Integrated development environment (IDE) used is Microsoft Visual Studio. This paper describes the already existing media libraries Windows Media Player, KODI and PLEX, technologies and packages used to create this application and the software code responsible for running the application itself.

Ključne riječi/key words: Media library, Microsoft Visual Studio, C#, Windows Presentation Foundation, Entitty Framework

1. Uvod

U današnje vrijeme se susrećemo s velikim brojem aplikacija za različite namjene, bilo na mobilnim uređajima, pametnim kućanskim aparatima, računalima pa čak i u automobilima. Malo je reći da većina takvih aplikacija ima široku primjenu i velik broj ugrađenih opcija.

Uz to, količina informacija je sve veća, što dovodi do problema prikaza tih informacija korisniku na primjeren način koji će mu omogućiti jednostavno korištenje.

Aplikacijama, odnosno tvorcima istih, je u interesu da korisnik što lakše pronađe informacije koje traži jer budućnost nam sigurno nosi sve složenije aplikacije i široki raspon usluga. Uz razvoj tehnologije, veoma je bitno i prikazivanje informacija na pravilan, intuitivan i jednostavan način, kako bi se korisnici što brže i lakše snalazili.

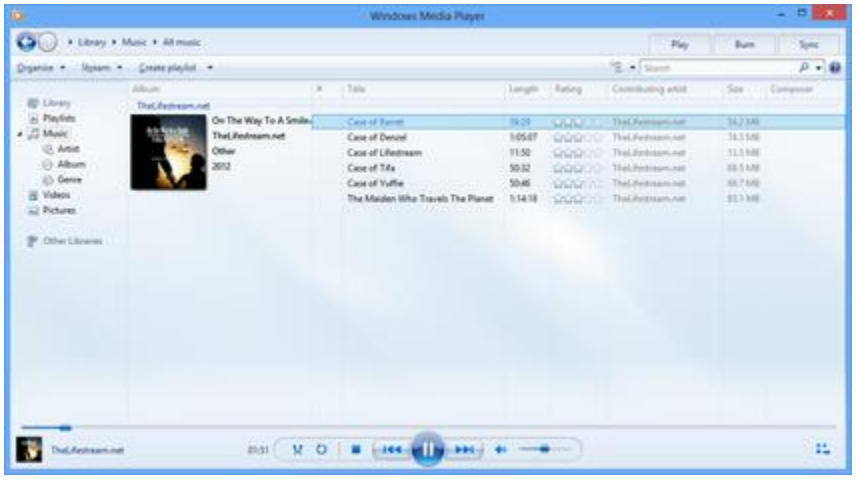
Iz tog razloga ovaj diplomski rad će se baviti izradom medijske knjižnice koja detektira video sadržaje na korisnikovom računalu, te ih sprema u bazu podataka zajedno sa njihovim metapodacima. Odnosno, postupak započinje pretragaom za video datotekama filmova/serija sa određene putanje na računalu, nakon čega se dohvaćaju odgovarajući meta podaci sa Interneta za svaku i spremaju u bazu podataka kako bi materijali bili dobro organizirani i spremni za sortiranja, filtriranja unutar aplikacije.

2. Medijska knjižnica

Medijska knjižnica (eng. media library) je računalni program koji pomaže korisniku u upravljanju sa slikovnim, video i audio datotekama na vlastitom računalu. Pod upravljanje spadaju operacije dodavanja, uređivanja, pregledavanja i brisanja datoteka.

Dakle na medijsku knjižnicu možemo gledati kao na pravu knjižnicu u kojoj su knjige sortirane prema autorima, žanru ili nekom drugom parametru. Uz razliku da korisnik može pretraživati medijsku knjižnicu pomoću više parametara od jednom te pronaći upravo onu datoteku koju traži, bila audio, video ili slikovna.

Prvi oblik medijske knjižnice se pojavio na Windows platformi sa nazivom Windows Media Player (WMP) čija se iteracija može i danas pronaći na Windows operativnom sustavu. WMP nudi mogućnosti reprodukcije sadržaja sa putanje koju odredimo (ukoliko na toj putanji postoje sadržaji za reprodukciju) te njihovo sortiranje i pretraživanje (Slika 1.). Također omogućuje kreiranje playlist-a koje se spremaju u obliku XML datoteke.



Slika 1 – Windows Mesia Player medijska knjižnica

Danas su mogućnosti medijskih knjižnica velike te ne služe samo za gore navedene operacije. Jedna od najpopularnijih opcija je takozvano dohvaćanje sadržaja na zahtjev (eng. streaming) sa računala na ostale uređaje koji su povezani na Internet. U tom slučaju računalo preuzima ulogu osobnog servera na koji se korisnik može spojiti i pregledavati sadržaj bez da je fizički za računalom.

Trenutno najpopularnije medijske knjižnice su:

- Kodi[1]
- Plex[2]

2.1. Kodi

Kodi je besplatan softver otvorenog tipa (eng. open-source) koji je razvio XBMC Foundation. Dostupan je za većinu operativnih sustava, odnosno Windows, MacOS, Android, iOS, tvOS, Raspbian, Linux i FreeBSD. Omogućuje korisnicima da reproduciraju i gledaju većinu sadržaja kao što su videozapisi, glazba i radio emisije (eng. podcast) sa Interneta, kao i sve u multimedijske datoteka sa lokalnih i mrežnih medija za pohranu. Sadržaj na zahtjev sa Interneta se provodi pomoću usluga kao što su Amazon Prime, Crackle, radio Pandora, Spotify, Youtube, Netflix i mnoge druge.

Nastao je kao samostalno razvijena aplikacija za prvu generaciju Xbox konzola pod nazivom Xbox Media Center, što se kasnije pretvorilo u aplikaciju za gore navedene sustave uz promjenu imena u XBMC.

Zbog korištenja koda otvorenog tipa (eng. open source) i mogućnosti pokretanja na više vrsta uređaja (eng. cross-platform), sa svojim jezgrenim kodom pisanim u C++ programskom jeziku, izmijenjene inačice Kodi-ja su korištene kao softeverski okviri (eng. software framework) u različitim uređajima uključujući pametne televizore, hotelske televizijske sustave i druge. No baš zbog svoje „otvorene prirode“ dobio je negativnu pozornost zbog dostupnosti dodataka sa treće strane (eng. third party plug-ins) koji olakšavaju neovlašten pristup medijima sa zaštićenim autorskim pravima.

[1] KODI. 2018. *Kodi Wiki.* 10. 10. Pokušaj pristupa 10. 5 2019. https://kodi.wiki/view/Main_Page
[2] PLEX. n.d. *Support Articles.* Pokušaj pristupa 10. 5 2019. https://support.plex.tv/articles/

Glavne značajke su:

- Live TV – gledanje TV kanala koje mogu prenositi DTT, ADSL, kabel ili Internet streaming, ovisno o odabranom dodatku
- Video knjižnica – ključna značajka Kodi-ja koja omogućuje organizaciju vlastitog video sadržaja po kategorijama žanr, naslov, godina, glumci i redatelj
- Reprodukcija video sadržaja – pregled video sadržaja, takozvani „video player“
- Glazbena knjižnica – kao i video knjižnica, ali za audio datoteke
- Reprodukcija audio sadržaja – pregled audio sadržaja, takozvani „music player“
- Digitalni prikaz slike – pregledavanje vlastitih slika u digitalnom obliku
- Igrača knjižnica – kao i ostale knjižnice, no za računalne igre

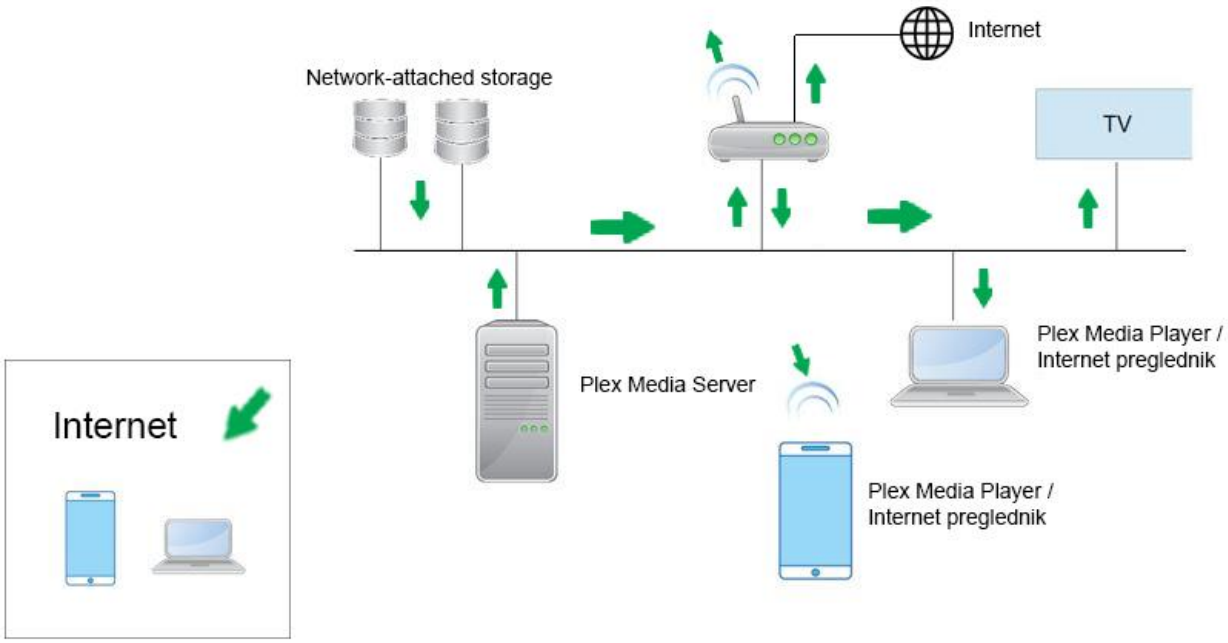
2.2. Plex

Plex je klijentstko-poslužiteljski (eng. client-server) sustav za reprodukciju medija. Njegov programski paket se sastoji od dvije glavne komponente:

- Plex Media Server – poslužitelj za Windows, MacOS i Linux operativne sustave pa čak i Network-attached storage (NAS) uređaje. Organizira video, audio i fotografije iz korisnikovih zbirki, omogućujući pristup i prijenos sadržaja na druge uređaje.
- Plex Media Player – klijent strana aplikacije na računalima, mobilnim uređajima, pametnim televizorima itd. koja pristupa sadržaju poslužitalja.

Nakon preuzimanja Plex Media Server-a na željeni uređaj, korisnik konfigurira putanje na kojima se nalazi medijski sadržaj. Tada će sadržaj biti rasčlanjen na video, audio i slikovne zapise te indeksiran unutar aplikacije za korištenje na drugim uređajima koji su spojeni na mrežu bilo lokalno ili putem Interneta. Za gledanje sadržaja uređaji mogu koristiti Plex Media Player

aplikaciju, no pošto Plex koristi i WebRTC[3] projekt, koji omogućuje komunikaciju između Internetskih preglednika i aplikacija, sadržaj je dostupan i putem bilo kojeg preglednika (Slika 2.).



Slika 2 – Klijent-server veza Plex aplikacije

Besplatan je za korištenje uz dodatnu opciju zvanu Plex Pass koja se naplaćuje, no nudi nove mogućnosti: sinkroznizacija s mobilnim uređajima, integraciju pohrane podataka u oblaku, podršku za više korisnika, Live TV i mogućnosti roditeljske kontrole.

Nastao je kao hobi razvojnog programera Elana Feringolda koji je želio aplikaciju medijskog centra za svoj Apple Mac 2007. godine. Zanimljivo je da su prve verzije bile bazirane na kodu za ranije spomenuti Kodi.

Glavne značajke su:

- Biblioteka videozapisa – omogućuje automatsku organizaciju video sadržaja pomoću informacija povezanih sa video datotekama
- Glazbena knjižnica – automatska organizacija glazbene zbirke putem informacija pohranjenih u ID3 ili M4A oznakama

[3] WebRTC. n.d. *WebRTC.* Pokušaj pristupa 10. 5 2019. https://webrtc.org/faq/

- Dodatni sadržaj knjižnice – prepoznavanje dodatni sadržaj uz video ili audio datoteku, uz napomenu da dodatni sadržaj moraj imati isti naziv audio/video datoteke koju nadopunjuje (npr. titlovi)
- Označavanje – automatsko prikupljanje metapodataka za audio/video datoteke pohranjene u Plex knjižnici

3. MediaBase

MediaBase je aplikacija za Windows operativni sustav, razvijena u okviru diplomskog rada na Odjelu za informatiku Sveučilišta u Rijeci.

Osnovna ideja aplikacije je skeniranje željene putanju na računalu te pronalazak svih video zapise na zadanoj putanji, pomoću rekurzivne pretrage koja obuhvaća sve direktorije i poddirektorije.

Nakon uspješnog pronalaska svih putanja na kojima se nalaze video datoteke, dohvaća meta podatke za te video datoteke sa themoviedb.org web stranice pomoću API-a i sprema ih u lokalnu bazu podataka. Tada korisnik ima pristup svim video sadržajima na jednome mjestu unutar aplikacije neovisno na kojim se putanjama nalaze na računalu.

3.1. Korištene tehnologije

Za izradu aplikacije korišten je Microsoft Visual Studio u kojemu je aplikacija programirana u C# programskom jeziku. Korisničko sučelje je rješeno pomoću Windows Presentation Foundation (WPF) sustava[4] za koji bi se moglo reći da je nasljednik klasičnih Windows Forms sučelja. Korišteni su sljedeći paketi, knjižnice (eng. libraries) i tehnologije:

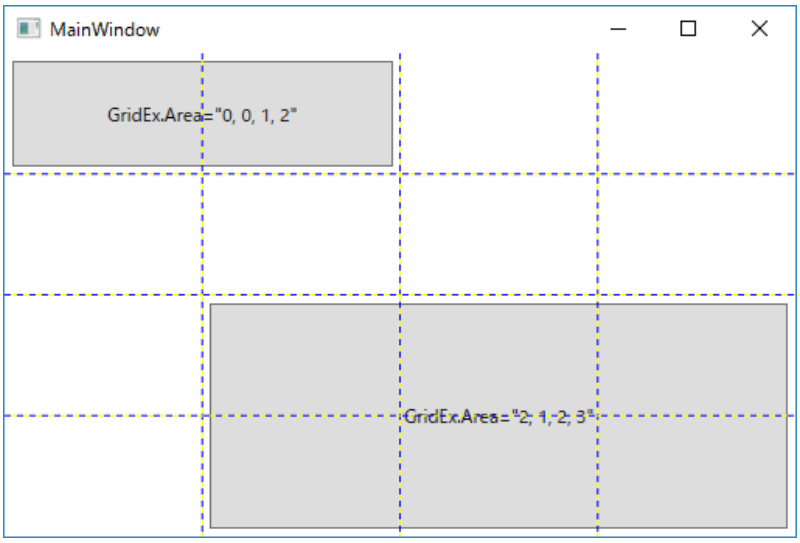
- GridExtra
- Entity Framework
- TheMovieDb API
- Torrent-name-parser
- Edge.js
- Newtonsoft.Json
- Microsoft SQL server
- OpenSubtitles API

[4] Jones, Mike, Petr Kulikov, i Maira Wenzel. 2018. Windows Presentation Foundation. 25. 1. Pokušaj pristupa 26. 3 2019. https://docs.microsoft.com/en-us/dotnet/framework/wpf/.

3.1.1. GridExtra

GridExtra je paket otvorenog koda (eng. open source) dostupan na GitHub-u[5] koji dostavlja funkcionalnost matričnog (eng. grid) oblikovanja korisničkog sučelja. Radi na vrlo sličan način kao i Bootstrap[6] paket namjenjen za HTML dizajn.

Dakle definira se koliko će korisničko sučelje imati redaka i stupaca, te se potom određuje lokacija za svaki element korisničkog sučelja, odnosno koliko i koje retke i stupce će zauzimati (Slika 3.).



Slika 3 – Primjer postavljanja 2 elementa u korisničkom sučelju sa 4 stupca i 4 retka

Time se postiže responzivnost korisničkog sučelja kojom se može kontrolirati ponašanje istog na različitim rezolucijama. Rezolucije su raspoređene na 4 dijela:

- XS – rezolucije manje od 768px širine
- SM – rezolucije manje od 992px širine
- MD – rezolucije manje od 1200px širine
- LG – rezolucije veće ili jednake 1200px širine

[5] minami_SC. 2017. *sourcechord/GridExtra: Custom panel controls for WPF/UWP*. 29. 10. Pokušaj pristupa 26. 3 2019. https://github.com/sourcechord/GridExtra

[6] Bootstrap. n.d. Documentation. Pokušaj pristupa 26. 3 2019. https://getbootstrap.com/docs/4.3/getting-started/introduction/.

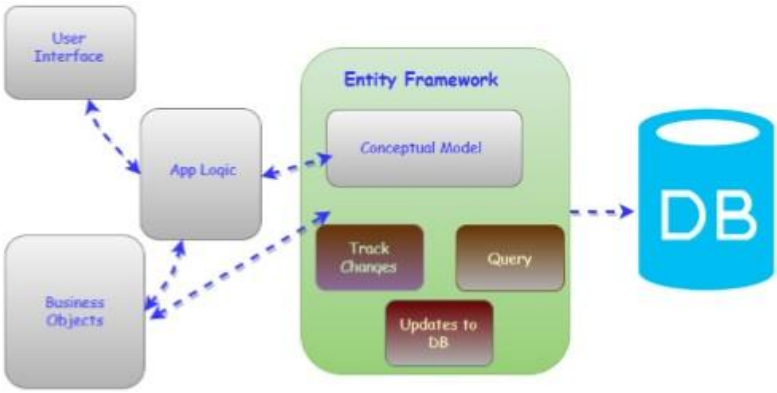
3.1.2. Entity Framework

Entity Framework[7] je objavljen 2008. godine kao Microsoft-ov glavni način interakcije između .NET aplikacija i relacijskih baza podataka. On je sam po sebi objektni relacijski maper (eng. object relational mapper; ORM u daljnjem tekstu) koji pojednostavljuje mapiranje između objekata u modelu podataka sa tablicama i redovima relacijske baze podataka.

Povećava produktivnosti programera smanjenjem redundantnog posla kodiranja funkcija za unos, čitanje, uređivanje i brisanje (eng. create, read, update, delete; CRUD u daljnjem tekstu) elemenata iz baze podataka na način da sam generirate potrebne funkcije. Umjesto SQL upita, koristi se LINQ (eng. language integrated query) .NET komponenta koja omogućuje formiranje SQL upita u jednom od .NET jezika (C#, F#, VB.NET). Upit kao rezultat prosljeđuje instance zatraženih objekata u obliku liste ili pojedinačnog objekta ovisno o samom upitu.

Može se koristiti na dva različita načina. Prvi način je da program u .NET jeziku izradi konceptualni model podataka kojeg će Entity Framework pretvoriti u relacijsku bazu podataka. Dok u drugom načinu Entity Framework sam stvara model podataka na temelju već postojeće relacijske baze podataka.

Oba načina rješavaju problem prelaska iz konceptualnog modela u bazu podataka tako da se upiti rade nad konceptualnim modelom, a Entity Framework komunicira direktno sa bazom (Slika 4.[8]).



Slika 4 – Prikaz načina rada aplikacije sa EF komponentom i relacijskom bazom podataka

[7] Microsoft. n.d. *Entity Framework Documentation.* Pokušaj pristupa 26. 3 2019. https://docs.microsoft.com/en-us/ef/#pivot=entityfmwk&panel=entityfmwk1

[8] TutorialsPoint. n.d. *Entity Framework - Overview.* Pokušaj pristupa 26. 3 2019. https://www.tutorialspoint.com/entity_framework/index.htm

3.1.3. TheMovieDb API

TheMovieDb[9] je web stranica na kojoj je moguće pretraživati serije i filmove uz pomoć raznih parametara pretrage (npr. po naslovu, žanru, glumcu/glumici, redatelju itd.). Pomoću API-a te stranice moguće je poslati upit sa željenim parametrima nakon čega se kao rezultat vraća lista u JSON formatu.

Za potrebe ovog diplomskog rada je stvoren novi omotač koji pomaže u komunikaciji između aplikacije i TheMovieDb API-a na način da su definirane C# funkcije za pretraživanje filmova i serija po njihovom naslovu, za dohvaćanje dodatnih detalja za filmove i serije po njihovom TheMovieDB ID-u i dohvaćanje glumačke postave filma ili serije. Rezultat upita u JSON formatu je deserijaliziran pomoću Newtonsoft.Json paketa, koji je opisan u nastavku, i mapiran u odgovarajuću instancu klase.

Upit se upućuje API-u pomoću REST ((REpresentational State Transfer) web servisa u obliku: „https://api.themoviedb.org/3/kategorija/parametar_pretrage?api_key=KEY?ostali_parametri=..“ gdje kategorija označava tip pretrage koji može biti search, movie, tv i drugi. Prvi parametar označava znakovni niz ili brojčanu vrijednost pretrage, u slučaju da je kategorija „search“, tada je parametar znakovni niz koji predstavlja traži li se film (movie) ili serija (tv). Dok „movie“ i „tv“ kategorije zahtjevaju brojčanu vrijednost koja predstavlja TheMovieDb ID filma ili serije. Api ključ se dobiva besplatno registracijom na web stranicu. Ostali parametri pretrage ovise o kategoriji pretrage, a uglavnom su query (u slučaju kategorije „search“), jezik, broj stranica, regija, godina izdanja i drugi.

Za primjer je uzeta općenita pretraga po nazivu filma „Marvel Avengers“:

https://api.themoviedb.org/3/search/movie?api_key=API_KEY&query=Marvel%20Avengers

Rezultat je u JSON obliku na način da su u korijenu broj stranice, broj rezultata, ukupni broj rezultata i polje sa rezultatima gdje je prvi element polja uglavnom najtočniji željenim rezultatima. U polju rezultati su sadržane osnovne informacije o filmu (Slika 5.).

[9] TheMovieDatabase. n.d. Getting Started. Pokušaj pristupa 26. 3 2019. https://developers.themoviedb.org/3/getting-started/introduction.


```
{
  "page": 1,
  "total_results": 6,
  "total_pages": 1,
  "results": [
    {
      "vote_count": 19657,
      "id": 24428,
      "video": false,
      "vote_average": 7.6,
      "title": "The Avengers",
      "popularity": 48.358,
      "poster_path": "/cezWGskPY5x7GaglTTRN4Fugfb8.jpg",
      "original_language": "en",
      "original_title": "The Avengers",
      "genre_ids": [
        878,
        28,
        12
      ],
      "backdrop_path": "/hbn46fQaRmlpBuUrEiFqv0GDL6Y.jpg",
      "adult": false,
      "overview": "When an unexpected enemy emerges and threatens global safety and security, Ni
      "release_date": "2012-04-25"
    },
    {
      "vote_count": 26,
      "id": 368304,
      "video": false,
      "vote_average": 6.4,
      "title": "LEGO Marvel Super Heroes: Avengers Reassembled!",
      "popularity": 2.525,
      "poster_path": "/pPdB6N9iJjn9GwrNkhMS1FKgx22.jpg",
      "original_language": "en",
      "original_title": "LEGO Marvel Super Heroes: Avengers Reassembled!",
      "genre_ids": [
        16
```


Slika 5 – Prikaz rezultata pretrage TMDB API-a u JSON formatu

3.1.4. Torrent-name-parser

Torrent-name-parser[10] je Node.JS paket koji pomoću regularnih izraza normalizira nazive filmova i serija koji su preuzeti putem torrent-a. Ubačen je u aplikaciju iz razloga što većina ciljanih korisnika aplikacije MediaBase upravo na taj način dolaze do medijskih sadržaja.

Princip rada je najbolje objašnjiv primjerom. Dakle recimo da postoji video datoteka sa nazivom brooklyn.nine-nine.s05e01.web.x264-tbs.mkv, ljudskom oku je jasno da se radi o prvoj epizodi pete sezone serije „Brooklyn Nine-Nine“ sa x264 codec-om i ekstenzijom .mkv. Kada bi se takav naziv datoteke proslijedio TheMovieDbWrapper-u, rezultat vrlo vjerojatno nebi bio odgovarajuć

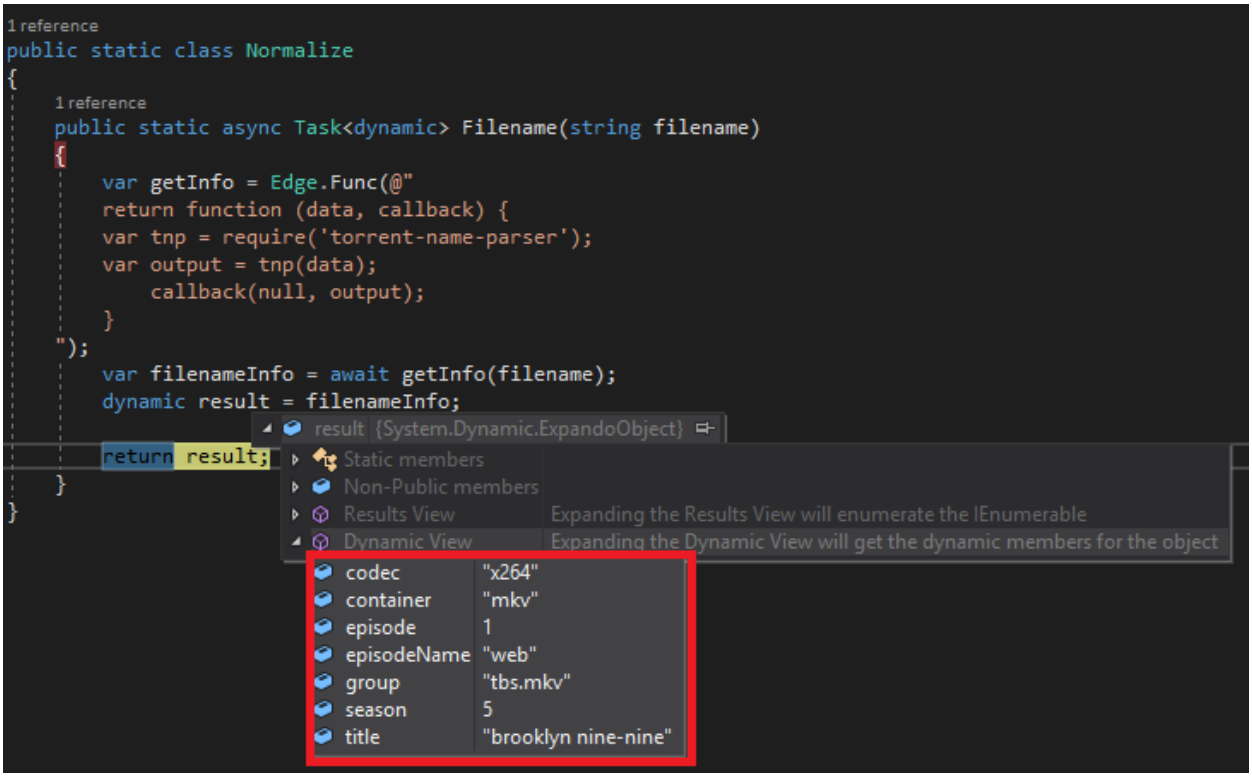
[10] clem6ever. 2016. *torrent-name-parser*. Pokušaj pristupa 26. 3 2019. https://www.npmjs.com/package/torrent-name-parser

pošto API ne razumije radi li se o filmu ili o seriji te koji dio naziva datoteke uopće upućuje na naslov.

Prosljeđivanjem ovakvog naziva u Torrent-name-parser vraća se rezultat u obliku dinamičkog objekta:

- season: 5
- episode: 1
- container: 'mkv'
- group: 'tbs.mkv'
- codec: 'x264'
- title: 'Brooklyn Nine-Nine'

Objekt je dinamičkog oblika iz razloga što nema svaki film/serije iste atribute pa se zbog toga sprema u objekt tipa rječnika sa parom ključa i vrijednosti (npr. film nema broj sezone niti epizode) (Slika 6.). Nakon ovog procesa, video je spreman na daljnju obradu podataka.



Slika 6 – Primjer normaliziranja naziva datoteke brooklyn.nine-nine.s05e01.web.x264-tbs.mkv

Edge klasa je dio paketa Edge.JS koji je je opisan u nastavku rada, a njegova funkcija Func prosljeđuje znakovni niz Node.JS zahtjeva koji se emulira u .NET za normaliziranje naziva.

3.1.5. Edge.JS

Edge.JS[11] je paket otvorenog koda koji se koristi isključivo iz razloga što ne postoji Torrent-name-parser paket za C# okruženje. Edge.JS je paket koji omogućuje izvođenje Node.JS skripti unutar .NET aplikacije.

3.1.6. Newtonsoft.Json

Newtonsoft.Json[12] je paket otvorenog koda koji omogućuje serijalizaciju bilo kojeg .NET objekta u JSON (Slika 7.[13]) format te deserijalizaciju JSON objekta u .NET model (Slika 8.[14]).

```
private void JSONSerilaize()
{
    // Serializaion
    Employee empObj = new Employee();
    empObj.ID = 1;
    empObj.Name = "Manas";
    empObj.Address = "India";

    // Convert Employee object to JOSN string format
    string jsonData = JsonConvert.SerializeObject(empObj);
                      jsonData "{\"ID\":1,\"Name\":\"Manas\",\"Address\":\"India\"}"
    Response.Write(jsonData);
}
```

Slika 7 – Primjer serijalizacije objekta Employee u JSON objekt

```
private void JSONDeserilaize()
{
    string json = @"{
        'ID': '1',
        'Name': 'Manas',
        'Address': 'India'
    }";

    Employee empObj = JsonConvert.DeserializeObject<Employee>(json);
                      empObj {WebApplication3.Employee}
                        Address "India"
                        ID      1
                        Name    "Manas"
    Response.Write(empObj.Name);
```

Slika 8 – Primjer deserijalizacije JSON objekta u objekt Employee

[11] Janczuk, Tomas. 2018. Edge.js: .NET and Node.js in-process. 3. 5. Pokušaj pristupa 26. 3 2019. https://github.com/tjanczuk/edge.
[12] Newtonsoft. n.d. *Json.NET.* Pokušaj pristupa 26. 3 2019. https://www.newtonsoft.com/json
[13] Mohapatra, Manas. 2018. JSON Serialization And Deserialization Using JSON.NET Library In C#. 20. 11. Pokušaj pristupa 26. 3 2019. https://www.c-sharpcorner.com/UploadFile/manas1/json-serialization-and-deserialization-using-json-net-librar/
[14] Mohapatra, Manas. 2018. JSON Serialization And Deserialization Using JSON.NET Library In C#. 20. 11. Pokušaj pristupa 26. 3 2019. https://www.c-sharpcorner.com/UploadFile/manas1/json-serialization-and-deserialization-using-json-net-librar/

Koristi se u paru sa TheMovieDb API-em i vlastitim izvornim repozitorijem, za deserijalizaciju rezultata kojeg TheMovieDb API vraća kao rezultat upita.

3.1.7. Microsoft SQL server

Microsoft SQL server[15] je sustav upravljanja relacijskim bazama podataka razvijen od strane Microsoft-a 1989. godine. Podržava širok raspon obrade transakcija, aplikacije poslovne inteligencije i analitike u korporativnim IT okruženjima. To je jedna od tri vodeće tehnologije baza podataka zajedno s Oracle Database i IBM-ovim DB2[16].

Kao i druga programska rješenja za upravljanje relacijskim bazama podataka, MSSQL je izgrađen na SQL-u, standardiziranom programskom jeziku koji se koristi za upravljanje i pretraživanje baza podataka. Temelji se na strukturi tablica zasnovanoj na retku koja spaja povezane elemente podataka u različite tablice, izbjegavajući potrebu pohranjivanja istih podataka na više lokacija unutar baze podataka. Relacijski model također osigurava održavanje točnosti podataka.

3.1.8. OpenSubtitles API

OpenSubtitels API[17] je u početku koristio XML-RPC[18] protokol kojeg je teže spremati u predmemoriju, stoga su se nedavno prebacili na REST zahtjeve kao i TheMovieDb čime su uspjeli ubrzati rad API-a i povećati broj mogućih zahtjeva u isto vrijeme.

Funkcionira na sličan način kao i TheMovieDb API, upućivanjem REST upita u obliku https://rest.opensubtitles.org/search/episode-[int]/imdbid-[int]/moviebytesize-[int]/moviehash-[string]/query-[string]/season-[int]/sublanguageid-[string]/tag-[string]. Svi parametri su neobavezni, no što ih je više, to će rezultat biti točniji:

- episode – brojčana vrijednost epizode ukoliko se radi o seriji

[15] —. n.d. SQL Server 2017 on Windows and Linux. Pokušaj pristupa 29. 3 2019. https://www.microsoft.com/en-us/sql-server/sql-server-2017.

[16] Rouse, Margaret. 2006. Microsoft SQL Server . 24. 1. Pokušaj pristupa 29. 3 2019. https://searchsqlserver.techtarget.com/definition/SQL-Server..

[17] OpenSubtitles. n.d. Subtitles API. Pokušaj pristupa 29. 3 2019. https://trac.opensubtitles.org/projects/opensubtitles/wiki/XMLRPC

[18] UserLand Software. 1999. *Home.* 14. 6. Pokušaj pristupa 29. 3 2019. http://xmlrpc.scripting.com/

- imdbid – ID filma ili serije sa IMDB stranice bez prva dva znaka (https://www.imdb.com/title/tt**0848228**/)
- moviebytesize – veličina datoteke u byte-ovima
- moviehash – 16 znakova hash oznake datoteke, mora se koristiti uz moviebytesize parametar
- query – naziv filma i serije uz napomenu da razmaci moraju biti u URL formatu, odnosno zamjenjeni sa „%20“
- season – brojčana vrijednost sezone ukoliko se radi o seriji
- sublanguageid – kratica od 3 slova za jezik
- tag – znakovni niz kojima se opisuje film/serija u URL formatu

Valja napomenuti da je URL uvijek potrebno formatirati redom kojim su nabrojani parametri. Također je potrebno u zaglavlju (eng. header) upita upisati valjajući „User-Agent“ koji za potrebe testiranja može glasiti „TemporaryUserAgent“, no za produkciju je potrebno registrirati računa na OpenSubtitles web stranici i potpuno besplatno zatražiti vlastiti „User-Agent“.

Rezultati su u JSON formatu sa kodom odgovora (eng. response code) 200 ukoliko postoje i nije došlo do greško prilikom traženja (Slika 9.). Gdje je prvi rezultat onaj koji bi trebao biti najtočniji. Daljnjom obradom je potrebno dohvatiti SubDownLoadLink atribut, preuzeti raspakirati preuzetu datoteku u direktorij u kojemu se nalazi film/serija za koji su potrebni podnaslovi, što će biti objašnjeno u nastavku diplomskog rada.

Slika 9 – Primjer rezultata OpenSubtitles API-a u JSON formatu

3.2. Baza i model podataka

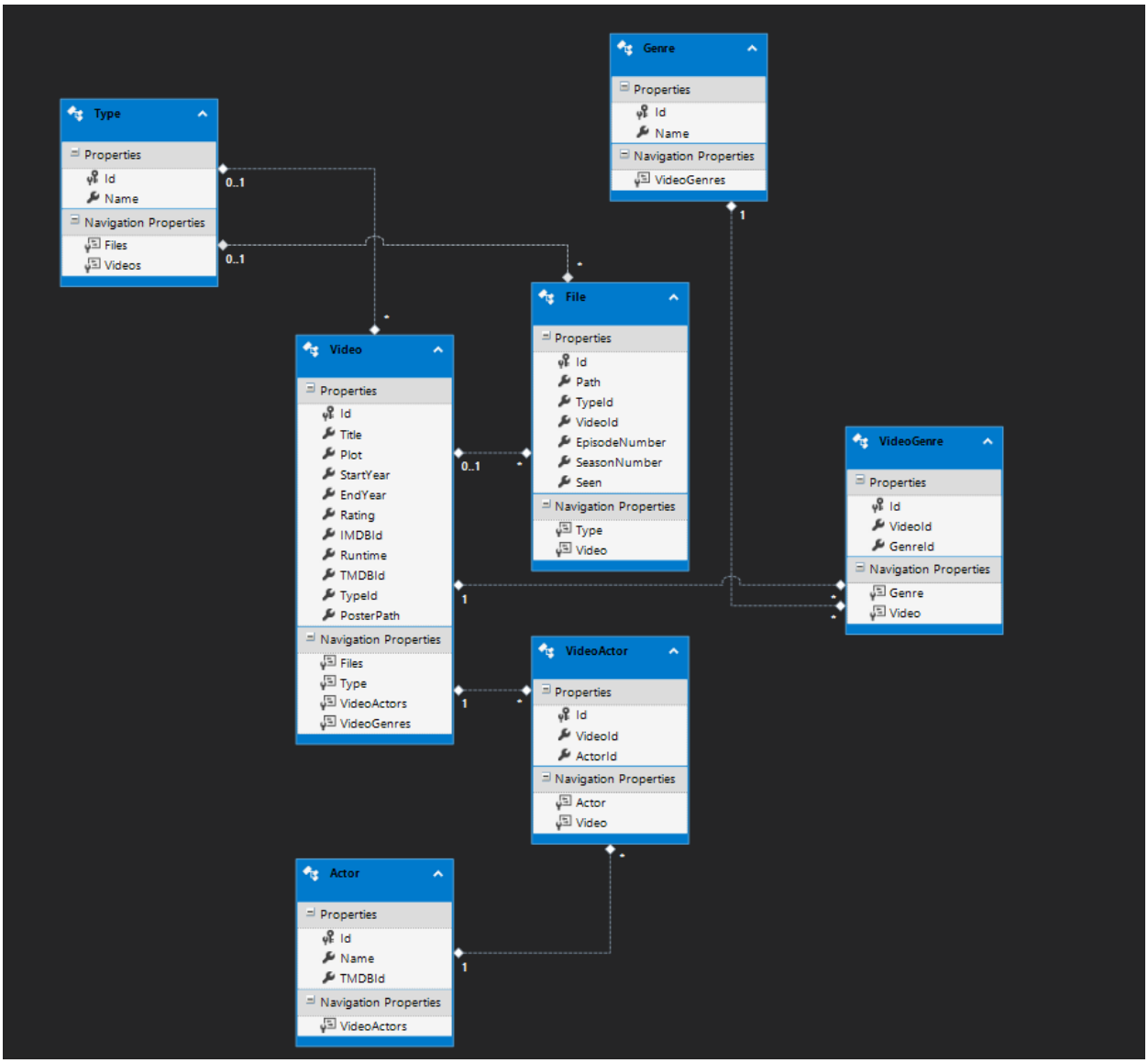
Baza podataka u MediaBase aplikaciji služi za pohranu informacija o datotekama, filmovima/serijama, glumcima te uz koji film/seriju su povezani, kao i o žanrovima. Stvorena je pomoću MSSQL-a te povezana sa projektom putem Entity Framework-a. Nalazi se unutar glavnog direktorija u .mdf formatu.

Tablice u bazi podataka:

- Files (sadrži putanje do datoteka)
    - **Id (int) (PK)**
    - Path (varchar) – putanja do video datoteke
    - EpisodeNumber (smallint) [nullable] – ukoliko se radi o datoteci serije, broj epizode
    - SeasonNumber (smallint) [nullable] – ukoliko se radi o datoteci serije, broj sezone
    - Seen (bool) – označuje je li korisnik pogledao taj video zapis
    - VideoId (int) (FK) – vanjski ključ na Videos entitet koji sadržava meta podatke za taj film/seriju
    - TypeId (smallint) (FK) – vanjski ključ koji označuje tip datoteke
- Types (tip datoteka (video), videa (film/serija))
    - **Id (smallint) (PK)**
    - Name (varchar) – naziv tipa (Video/Movie/TV Show)
- Genres (žanrovi filmova/serija)
    - **Id (smallint) (PK)**
    - Name (varchar) – naziv žanra
- Videos (metapodaci za film/seriju)
    - **Id (int)**
    - Title (varchar) – naslov filma/serije
    - Plot (varchar) – opis radnje
    - StartYear (smallint) – godina kada je film snimljen / godina početka snimanja serije
    - EndYear (smallint) [nullable] – null u slučaju filma / godina završetka snimanja serije ukoliko je završila
    - Rating (decimal) – ocjena od 1 do 10 od strane korisnika TheMovieDb web stranice

    - IMDBId (varchar) – ID tog filma/te serije na IMDB web stranici
    - TMDBId (int) – ID tog filma/te serije na TheMovieDb web stranici
    - Runtime (int) – vrijeme trajanja filma/jedne epizode u minutama
    - PosterPath (varchar) – URL do putanje naslovne slike za film/seriju
    - TypeId (smallint) FK – vanjski ključ koji označuje radi li se o filmu (movie) ili seriji (tv show)
- Actors (lista glumaca)
    - **Id (int) PK**
    - Name (varchar) – ime i prezime glumca
    - TMDBId (int) – ID tog glumca na TheMovieDb stranici
- VideoActors (agregacija glumaca i videa – gdje je tko glumio)
    - **Id (int) PK**
    - VideoId (int) FK – vanjski ključ entiteta Videos
    - ActorId (int) FK – vanjski ključ entiteta Actors
- VideoGenres (agregacija žanrova i videa – koji video je kojeg žanra)
    - **Id (int) PK**
    - VideoId (int) FK – vanjski ključ entiteta Videos
    - GenreId (smallint) FK – vanjski ključ entiteta Genres

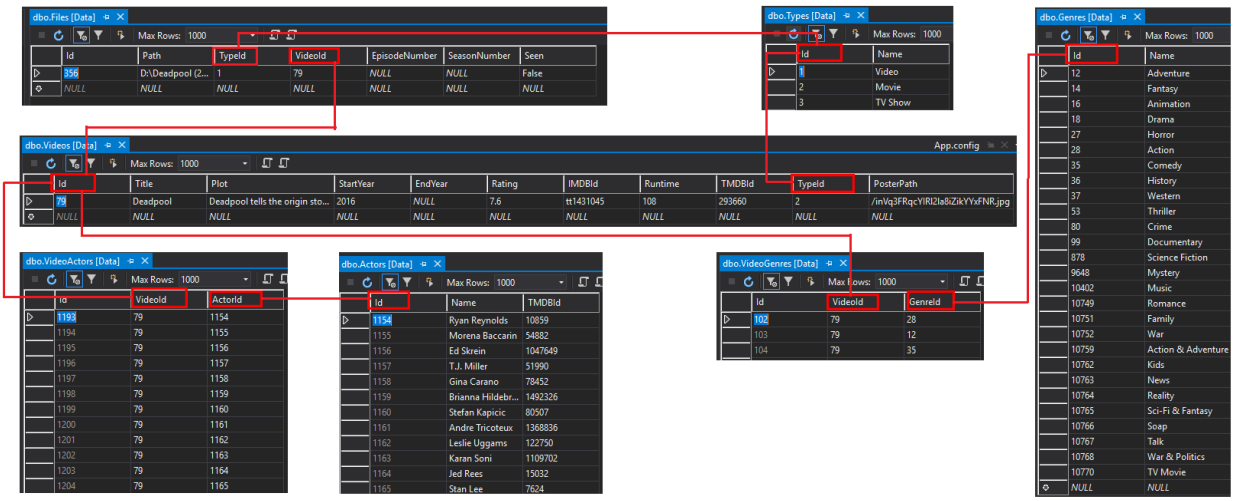
Model je stvoren pomoću Entity Framwork-a koji automatski na temelju tablica u bazi podataka generira odgovarajuće klase/entitete koje su spremne za korištenje u programskom djelu (Slika 10.).



Slika 10 – Model podataka generiran pomoću EF na temelju baze podataka

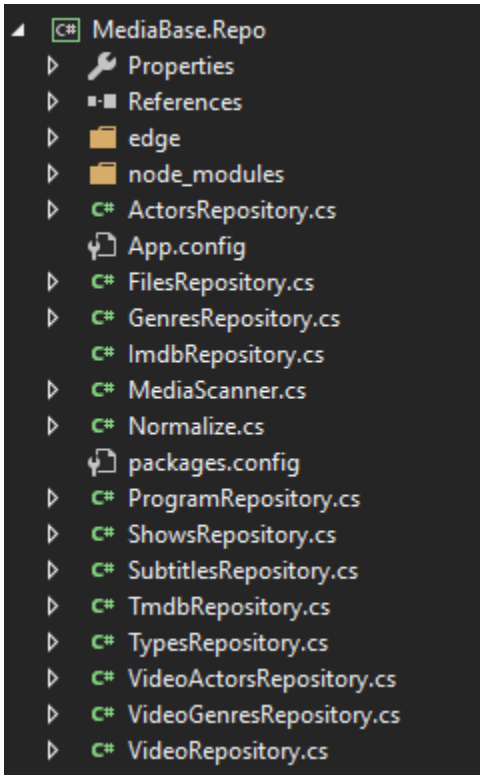
Za upis, čitanje, uređivanje i brisanje u/iz baze podataka Entity Framework koristi funkcije kontekstualne klase (DbContext) koja je roditelj svim stvorenim modelima.

Nakon pronalaska video datoteke/a na putanji i dohvaćanja odgovarajućih meta podataka za svaku, sve je spremljeno u bazu podataka (Slika 11.).



Slika 11 – Prikaz baze podataka sa jednim filmom

3.3 Repozitorij



Slika 12 - Repozitorij

Repozitorij (Slika 12.) je dio projekta koji sadrži svu logiku upravljanja aplikacijomi rada sa bazom podataka. Najvećim djelom se sastoji od kreiraj, pročitaj, uredi i izbriši (eng. CREATE, READ, UPDATE, DELETE – CRUD) funkcija za svaku klasu modela.

Sve klase unutar repozitorija su zaslužne za funkcioniranje programa:

- MediaScanner
- Normalize
- TmdbRepository
- ProgramRepository
- SubtitlesRepository
- CRUD klase

3.3.1. MediaScanner

Klasa čije funkcije pretražuju sve direktorije (i poddirektorije) na određenoj putanji te kao rezultat vraća listu tipa Files koja sadrži samo one datoteke koje su u video formatu, odnosno pregledava ekstenzije svih datoteka na zadanoj putanji te na listu sprema one sa odgovarajućom ekstenzijom (npr. .avi, .mp4, .mkv...) (Slika 13.).

```
2 references
public static List<Model.File> GetMedia(string targetPath)
{
    if (!IsValidPath(targetPath))
        throw new Exception("Please input the correct path!");
    List<Model.File> Files = new List<Model.File>();
    FilesRepository fileRepo = new FilesRepository();

    List<string> mediaPaths = GetDirectoryFiles(targetPath, "*.*", SearchOption.AllDirectories).ToList();

    foreach (string mediaPath in mediaPaths)
    {
        if (!fileRepo.Exists(mediaPath))
        {
            Model.File file = new Model.File
            {
                Path = mediaPath,
                TypeId = 1
            };
            Files.Add(file);
        }
    }
    return Files;
}
```

Slika 13 – Prikaz funkcije GetMedia – unutar klase MediaScanner

Funkcija prima parametar putanje koji je korisnik zadao te prvo provjerava je li ispravna, jer ukoliko korisnik unese znakovni niz koji ne odgovara klasičnom zapisu putanje, program nema smisla pokretati. Točnost putanje određuje funkcija IsValidPath(string path) koja na temelju regularnih izraza određuje započinje li znakovni niz sa slovom nakon kojeg slijedi dvotočka i obrnuta kosa crta, te postoje li nedozvoljeni znakovi unutar putanje (Slika 14.).

```
1 reference
public static bool IsValidPath(string path)
{
    Regex driveCheck = new Regex(@"^[a-zA-Z]:\\$");
    if (path.Count() < 3) return false;
    if (!driveCheck.IsMatch(path.Substring(0, 3))) return false;
    string strTheseAreInvalidFileNameChars = new string(Path.GetInvalidPathChars());
    strTheseAreInvalidFileNameChars += @":/?*" + "\"";
    Regex containsABadCharacter = new Regex("[" + Regex.Escape(strTheseAreInvalidFileNameChars) + "]");
    if (containsABadCharacter.IsMatch(path.Substring(3, path.Length - 3)))
        return false;
    return true;
}
```

Slika 14 – Prikaz funkcije IsValidPath – unutar klase MediaScanner

Ako je putanja ispravna, funkcija GetDirectoryFiles rekurzivno pretražuje sve direktorije i poddirektorije tražeći datoteke sa odgovarajućom ekstenzijom pritom pazeći na moguće izuzetke (eng. exceptions) koji se mogu pojaviti. Kao rezultat vraća IEnumerable listu znakovnih nizova koja sadrži putanje do svih datoteka sa odgovarajućom ekstenzijom (Slika 15.).

```
2 references
public static IEnumerable<string> GetDirectoryFiles(string rootPath, string patternMatch, SearchOption searchOption)
{
    var foundFiles = Enumerable.Empty<string>();
    if (searchOption == SearchOption.AllDirectories)
    {
        try
        {
            IEnumerable<string> subDirs = Directory.EnumerateDirectories(rootPath);
            foreach (string dir in subDirs)
            {
                foundFiles = foundFiles.Concat(GetDirectoryFiles(dir, patternMatch, searchOption));
            }
        }
        catch (UnauthorizedAccessException) { }
        catch (PathTooLongException) { }
    }
    try
    {
        foundFiles = foundFiles.Concat(Directory.EnumerateFiles(rootPath, patternMatch).Where(f => extensions.Contains(System.IO.Path.GetExtension(f))));
    }
    catch (UnauthorizedAccessException) { }
    return foundFiles;
}
```

Slika 15 – Prikaz funkcije GetDirectoryFiles – unutar klase MediaScanner

Za kraj algoritam provjerava postoji li ta putanja u bazi podataka pomoću funkcije iz FilesRepository klase, ako ne postoji, dodaje ju na listu datoteka za daljnju obradu (Slika 13.).

3.3.2. Normalize

Klasa izvodi Node.js paket zvan torrent-name-parser koji iz naziva pronađene video datoteke izvlači najbitnije informacije (radi li se o filmu ili seriji, naziv,broj sezone, broj epizode), što je opisano u 3.1.4. poglavlju rada (Slika 6.).

3.3.3. TmdbRepository

Repozitorij koja služi za povezivanje sa TheMovieDb API-em. Zaslužan je za dohvaćanje svih meta podataka filmova/serija koji se kasnije spremaju u bazu podataka.

Svaka funkcija služi za dohvaćanje određene JSON liste i pretvaranje iste u odgovarajuće aplikacijske modele. Za komunikaciju sa API-em koristi se klasa RestClient koja je dio RestSharp paketa. Kod inicijalizacije instance RestClient-a konstruktoru se prosljeđuje URL sa API zahtjevom, te nakon izvođenja zahtjeva, vraća se JSON rezultat koji se deserijalizira u dinamički objekt.

Kod dohvaćanja meta podataka o samom filmu potrebno je poslati još jedan zahtjev koji na temelju upravo dohvaćenog TheMovieDbID-a pronalazi dodatne detalje za film/seriju. Iz razloga što kod prvog zahtjeva nisu zapisani podaci o vremenu trajanja i ImdbId-u koji je naknadno potreban za dohvaćanje podnaslova (eng. subtitles) (Slika 16.).

```
public async Task<Video> GetMovieInfoAsync(string title)
{
    var client = new RestClient("https://api.themoviedb.org/3/search/movie?api_key=" + _apiKey +
        "&language=en-US&page=1&include_adult=false&query=" + title);
    var request = new RestRequest(Method.GET);
    request.AddParameter("undefined", "{}", ParameterType.RequestBody);
    IRestResponse response = client.Execute(request);
    var result = JsonConvert.DeserializeObject<dynamic>(response.Content);

    if (result.results[0].id != null)
    {
        int tmdbId = result.results[0].id;
        client = new RestClient("https://api.themoviedb.org/3/movie/" + tmdbId + "?language=en-US&api_key=" + _apiKey);
        response = client.Execute(request);
        var details = JsonConvert.DeserializeObject<dynamic>(response.Content);

        Video movie = new Video();

        movie.Plot = details.overview;
        movie.Rating = details.vote_average;
        movie.StartYear = (short)DateTime.Parse(details.release_date.ToString()).Year;
        movie.Title = details.original_title;
        movie.TMDBId = details.id;
        movie.PosterPath = details.poster_path;
        movie.TypeId = 2;
        movie.Runtime = details.runtime;
        movie.IMDBId = details.imdb_id;
        movie.OnCreated(); //Initialize List<Actor> and List<Genre>
        movie.Actors = GetMovieActors(movie.TMDBId.Value);
        foreach (var genre in details.genres)
        {
            Genre newGenre = new Genre();
            newGenre.Id = (short)genre.id;
            newGenre.Name = genre.name;
            movie.Genres.Add(newGenre);
        }
        return movie;
    }
    return null;
}
```

Slika 16 – Prikaz funkcije GetMovieInfoAsync unutar klase TmdbRepository

Analogno tome na isti način funkcionira funkcija za dohvaćanja meta podataka serije, uz iznimku da se prosljeđuje API poziv koji označuje da treba pretraživati serije, a ne filmove.

Funkcija GetMovieActors je također dio TmdbRepository klase koja na temelju TMDBId-a šalje API zahtjev za ispis svih glumaca u filmu, na isti način radi i funkcija GetShowActors za serije (Slika 17.).

```
1 reference
public List<Actor> GetMovieActors(int movieId)
{
    List<Actor> actors = new List<Actor>();
    var client = new RestClient("https://api.themoviedb.org/3/movie/" + movieId + "/credits?api_key=" + _apiKey);
    var request = new RestRequest(Method.GET);
    request.AddParameter("undefined", "{}", ParameterType.RequestBody);
    IRestResponse response = client.Execute(request);
    var result = JsonConvert.DeserializeObject<dynamic>(response.Content);

    foreach (var responseActor in result.cast)
    {
        Actor actor = new Actor();
        actor.Name = responseActor.name;
        actor.TMDBId = responseActor.id;
        actors.Add(actor);
    }
    return actors;
}

1 reference
public List<Actor> GetTVShowActors(int tvShowId)
{
    List<Actor> actors = new List<Actor>();
    var client = new RestClient("https://api.themoviedb.org/3/tv/" + tvShowId + "/credits?language=en-US&api_key=" + _apiKey);
    var request = new RestRequest(Method.GET);
    request.AddParameter("undefined", "{}", ParameterType.RequestBody);
    IRestResponse response = client.Execute(request);
    var result = JsonConvert.DeserializeObject<dynamic>(response.Content);

    foreach (var responseActor in result.cast)
    {
        Actor actor = new Actor();
        actor.Name = responseActor.name;
        actor.TMDBId = responseActor.id;
        actors.Add(actor);
    }
    return actors;
}
```

Slika 17 – Prikaz funkcija GetMovieActors i GetTVShowActors unutar klase TmdbRepository

3.3.4. CRUD klase

Za upravljanje nad bazom podataka se koriste klase koje sadržavaju funkcije unosa (eng. create), ispisa (eng. read), uređivanja (eng. edit) i brisanja (eng. delete) (CRUD). Za svaku tablicu u bazi podataka postoji odgovarajuća CRUD klasa za obavljanje tih operacija.

- ActorsRepository
- FilesRepository
- GenresRepository

- ShowsRepository
- TypesRepository
- VideoActorsRepository
- VideoGenresRepository
- VideoRepository

Pošto su sve klase ovog popisa većinom generične, kao primjer će biti prikazana klasa VideoRepository (Slika 18.).

```
public int Create(Video video)
{
    using (MediaDatabaseEntities db = new MediaDatabaseEntities())
    {
        if (video == null)
        {
            throw new Exception();
        }

        if (db.Videos.Where(v => v.TMDBId == video.TMDBId).Count() == 0)
        {
            db.Videos.Add(video);
            db.SaveChanges();
            return video.Id;
        }
        else
        {
            return db.Videos.Where(v => v.TMDBId == video.TMDBId).FirstOrDefault().Id;
        }
    }

}
```

Slika 18 – Prikaz funkcije Create unutar klase VideoRepository

Instanca Video klase je prosljeđena u ovu funkciju unosa sa svim potrebnim unesenim meta podacima. Dolazi do greške jedino u slučaju da je ta instanca prazna (eng. null). Ukoliko nije prazna, provjerava se postoji li već takav zapis u bazi pomoću jedinstvenog TMDB (TheMovieDatabaseId ključa), u slučaju da ne postoji novi zapis sa meta podacima je dodan u bazu i novostvoreni primarni ključ je vraćen kao rezultat, a ako postoji onda funkcija vraća postojeći primarni ključ.

Za ispis podataka postoji više funkcija gdje svaka služi svojoj svrsi. Te funkcije se odnose na sortiranja, filtriranja i pretraživanja koje korisnik može zatražiti. Glavna funkcija ispisa podataka je ReadBy koja vraća listu svih video zapisi određenog tipa (filma ili serije), uz opcionalne parametre sortiranja, filtriranja prema glumcu i/ili žanru i pretraživanja prema naslovu (Slika 19.).

```
public List<Video> ReadBy(short typeId, SortOption sortOption, short? genreId, int? actorId, string title)
{
    using(MediaDatabaseEntities db = new MediaDatabaseEntities())
    {
        List<Video> videos;
        videos = db.Videos.Where(v => v.TypeId == typeId).ToList();
        if (genreId.HasValue)
            videos = FilterByGenre(videos, genreId.Value);
        if (actorId.HasValue)
            videos = FilterByActor(videos, actorId.Value);
        if (String.IsNullOrEmpty(title))
            videos = ReadByTitleMulti(title);
        if(videos.Count > 1)
            videos = Sort(videos, sortOption);
        return videos;
    }
}
```

Slika 19 – Prikaz funckije ReadByType unutar klase VideoRepository

Sortiranje može biti uzlazno ili silazno abecedno po naslovu filma ili serije, uzlazno ili silazno po godini izdavanja i isto tako uzlazno ili silazno po ocjeni. Sortiranje je komplementarno sa bilo kojim načinom pretraživanja, dakle ispis video zapisa određenog tipa (ReadByType), sa filtriranjem i sa naprednim pretraživanjem, što znači da korisnik može sortirati bilo koje rezultate koje je prethodno zatražio (Slika 20.).

```
public List<Video> Sort(List<Video> videos,SortOption sortOption)
{
    List<Video> sortedVideos = new List<Video>();
    switch (sortOption)
    {
        case SortOption.TitleAscending:
            sortedVideos = videos.OrderBy(s => s.Title).ToList();
            break;
        case SortOption.TitleDescending:
            sortedVideos = videos.OrderByDescending(s => s.Title).ToList();
            break;
        case SortOption.YearAscending:
            sortedVideos = videos.OrderBy(s => s.StartYear).ToList();
            break;
        case SortOption.YearDescending:
            sortedVideos = videos.OrderByDescending(s => s.StartYear).ToList();
            break;
        case SortOption.RatingAscending:
            sortedVideos = videos.OrderBy(s => s.Rating).ToList();
            break;
        case SortOption.RatingDescending:
            sortedVideos = videos.OrderByDescending(s => s.Rating).ToList();
            break;
        default:
            sortedVideos = videos.OrderBy(s => s.Title).ToList();
            break;

    }
    return sortedVideos;
}
```

Slika 20 – Prikaz funkcije Sort unutar klase VideoRepository

Filtriranje je moguće prema žanru gdje korisnik traži sve filmove/serije iz baze određenog žanra i/ili prema glumcu gdje korisnik traži sve filmove/serije iz baze u kojima glumi određeni glumac (Slika 21.).

```
public List<Video> FilterByGenre(List<Video> videos, short genreId)
{
    List<Video> filteredVideos = new List<Video>();
    VideoGenresRepository vgRepository = new VideoGenresRepository();
    foreach(Video video in videos)
    {
        if (vgRepository.VideoHasGenre(video.Id, genreId))
            filteredVideos.Add(video);
    }
    return filteredVideos;
}

1 reference
public List<Video> FilterByActor(List<Video> videos, int actorId)
{
    List<Video> filteredVideos = new List<Video>();
    VideoActorsRepository vaRepository = new VideoActorsRepository();
    foreach (Video video in videos)
    {
        if (vaRepository.VideoHasActor(video.Id, actorId))
            filteredVideos.Add(video);
    }
    return filteredVideos;
}
```

Slika 21 – Prikaz funkcija FilterByGenre i FilterByActor unutar klase VideoRepository

Za dohvaćanje isključivo jednog video zapisa također se koristi funkcija ReadBy, no ova ReadBy funkcija prima samo jedan parametar, a to je ID filma kojei se traži (Slika 22.).

```
public Video ReadBy(int id)
{
    using (MediaDatabaseEntities db = new MediaDatabaseEntities())
    {
        Video video = db.Videos.Find(id);
        if (video == null)
            throw new Exception();
        ActorsRepository actorsRepository = new ActorsRepository();
        video.OnCreated();
        List<VideoActor> videoActors = db.VideoActors.Where(va => va.VideoId == video.Id).ToList();
        foreach (VideoActor videoActor in videoActors)
        {
            Actor actor = actorsRepository.ReadBy(videoActor.ActorId);
            video.Actors.Add(actor);
        }
        GenresRepository genresRepository = new GenresRepository();
        List<VideoGenre> videoGenres = db.VideoGenres.Where(vg => vg.VideoId == video.Id).ToList();
        foreach (VideoGenre videoGenre in videoGenres)
        {
            Genre genre = genresRepository.ReadBy(videoGenre.GenreId);
            video.Genres.Add(genre);
        }
        video.FilesList = db.Files.Where(f => f.VideoId == id).ToList();
        return video;
    }
}
```

Slika 22 – Prikaz funkcije ReadBy unutar klase VideoRepository

Uz to koristi ActorsRepository, GenresRepository i FilesRepository klase za dohvaćanje svih žanrova, glumaca i datoteka povezanih sa filmom/serijom kako bi mogli biti prikazani u korisničkom sučelju.

U slučaju da je program dohvatio pogrešne meta podatke, korisnik ih može ručno promijeniti unutar korisničkog sučelja, promjene će program spremiti u instancu klase Video te jednostavnim pozivom funkcije napraviti promjenu u bazi podataka (Slika 23.)

```
public void Edit(Video video)
{
    using (MediaDatabaseEntities db = new MediaDatabaseEntities())
    {
        db.Entry(video).State = EntityState.Modified;
        db.SaveChanges();
    }
}
```

Slika 23 – Prikaz funkcije Edit unutar klase VideoRepository

Korisnik ima mogućnost i uklanjanja video zapisa iz medijske knjižnice, uz to će ga program pitati želi li potpuno ukloniti video što će ga izbrisati i sa putanje na kojoj se nalazi ili želi samo izbrisati

meta podatke tog videa i ukloniti ga iz programa. Kod brisanja je potrebno ukloniti i sve reference na taj video, odnosno sva pojavljivanja u drugim tablicama u bazi podataka. Dakle potrebno je izbrisati svaki zapis gdje je video povezan sa žanrovima, glumcima i datotekama (Slika 24.)

```
public void Delete(int videoId, bool deleteFiles)
{
    using (MediaDatabaseEntities db = new MediaDatabaseEntities())
    {
        Video video = ReadBy(videoId);
        foreach(Genre genre in video.Genres)
        {
            db.Genres.Remove(genre);
        }
        foreach(Actor actor in video.Actors)
        {
            db.Actors.Remove(actor);
        }
        foreach(File file in video.FilesList)
        {
            if(deleteFiles)
                System.IO.File.Delete(file.Path);
            db.Files.Remove(file);
        }
        db.Videos.Remove(video);
        db.SaveChanges();
    }
}
```

Slika 24 – Prikaz funkcije Delete unutar klase VideoRepository

3.3.5. ProgramRepository

Glavna klasa cijelog programa koja povezuje većinu ostalih klasa u repozitoriju, dohvaća odgovarajuće datoteke sa funkcijama iz MediaScanner-a, normalizira nazive datoteka sa Normalize klasom, dohvaća meta podatke za njih uz TmdbRepository i sprema ih u bazu podataka sa CRUD klasama (Slika 25.).

```
public async Task CreateVideoMetadata(File file)
{
    VideoRepository vidRepo = new VideoRepository();
    FilesRepository fileRepo = new FilesRepository();
    Video video = null;
    dynamic fileInfo = await Normalize.Filename(file.Filename);

    if (((IDictionary<String, object>)fileInfo).ContainsKey("title"))
    {
        file.Seen = false;
        if (((IDictionary<String, object>)fileInfo).ContainsKey("episode") && ((IDictionary<String, object>)fileInfo).ContainsKey("season"))
        {
            video = vidRepo.ReadBy(fileInfo.title);
            if (video == null)
            {
                video = await _tmdbRepository.GetTTVShowInfoAsync(fileInfo.title);
            }
            if (video != null)
            {
                file.VideoId = vidRepo.Create(video);
                file.EpisodeNumber = (short)fileInfo.episode;
                file.SeasonNumber = (short)fileInfo.season;
                fileRepo.Create(file);
                AddVideoActors(file.VideoId.Value, video.Actors);
                AddVideoGenres(file.VideoId.Value, video.Genres);
            }
        }
        else
        {
            video = vidRepo.ReadBy(fileInfo.title);
            if (video == null)
            {
                video = await _tmdbRepository.GetMovieInfoAsync(fileInfo.title);
            }
            if (video != null)
            {
                file.VideoId = vidRepo.Create(video);
                fileRepo.Create(file);
                AddVideoActors(file.VideoId.Value, video.Actors);
                AddVideoGenres(file.VideoId.Value, video.Genres);
            }
        }
    }
}
```

Slika 25 – Prikaz funkcije CreateVideoMetadata unutar klase ProgramRepository

Funkciji se prosljeđuje instanca klase File koja u sebi sadrži putanju do video datoteke čiji su meta podaci potrebni. Prvo se normalizira sam naziv datoteke kako bi se odredio sam naslov filma ili serije, ako se uistinu radi o valjanom naslovu postavlja se atribut „seen“ na neistinu pod pretpostavkom da korisnik video koji dodaje nije pogledao (kasnije se unutar UI-a ta vrijednost može promijeniti. Slijedi provjera radi li se o seriji ili filmu, ukoliko datoteka u sebi sadrži indikacije o broju sezone i broju epizode tada se radi o seriji, inače je riječ o filmu.

Neovisno radi li se o filmu ili seriji, prvo se pomoću CRUD klase VideoRepository provjerava postoje li meta podaci za taj film ili seriju zapisani u bazu, pošto korisnik može imati više verzija kvalitete istog filma ili ako se radi o seriji svaka epizoda nema zasebne meta podatke već ju opisuju meta podaci cijele serije. Kada je to ustanovljeno dohvaćaju se meta podaci iz TmdbRepository-ja.

U instancu klase File se nadodaje brojčana vrijednost koja je vanjski ključ i referira na primarni ključ upravo stvorenog zapisa u tablici Videos baze podataka. Za seriju se također u instancu

sprema broj epizode i broj sezone, atributi koji mogu biti i jesu prazni kod filmova. Tada se i ta instanca sprema u tablicu Files u bazi podataka.

Funkcije AddVideoActors i AddVideoGenres služe za punjenje agregacijskih tablica VideoActors i VideoGenres gdje je svaki redak ispunjen primarnim ključem videa i primarnim ključem glumca ili žanra ovisno o kojoj se tablici radi. Za isti video zapis, primarni ključ je uvijek identičan, dok se u svakom sljedećem zapisu mjenjaju primarni ključevi žanrova i glumaca (Slika 26.).

```
2 references
public static void AddVideoActors(int videoId, List<Actor> actors)
{
    VideoActorsRepository vaRepository = new VideoActorsRepository();
    ActorsRepository actorsRepository = new ActorsRepository();
    foreach (Actor actor in actors)
    {
        int actorId = actorsRepository.Create(actor);
        vaRepository.Create(videoId, actorId);
    }
}

2 references
public static void AddVideoGenres(int videoId, List<Genre> genres)
{
    VideoGenresRepository vgRepo = new VideoGenresRepository();
    foreach (Genre genre in genres)
    {
        vgRepo.Create(videoId, genre.Id);
    }
}
```

Slika 26 – Prikaz funkcija AddVideoActors i AddVideoGenres unutar klase Program Repository

Žanrovi su unaprijed postavljeni pri prvom paljenju aplikacije, dok se glumci dodaju u bazu pomoću ActorsRepository-ja, funkcija Create stvara novi zapis i vraća njezin primarni ključ, no prije toga provjerava postoji li već zapis o tom glumcu u bazi, ukoliko postoji vraća njegov primarni ključ.

3.3.6. SubtitlesRepository

Klasa služi za komunikaciju sa OpenSubtitles API-em na način da izvlači podatke o videu iz baze podataka i o samoj datoteci na koju se video odnosi te ih prosljeđuje funkciji SearchSubtitles unutar ove klase. Tada se na temelju tih parametara generira odgovarajući REST URL koji se izvršava pomoću .NET klase HttpClient i kao povratnu informaciju vraća JSON sa listom svih odgovarajućih titlova. Uz pomoć Newtonsoft.Json taj se sadržaj deserijalizira i sprema u listu klase Subtitle (Slika 27.)

```
public async Task<List<Subtitle>> SearchSubtitles(int? episodeNumber, string imdbId, int? movieByteSize,
    string movieHash, string movieName, int? seasonNumber, string languageCode)
{
    HttpClient client = new HttpClient();
    client.DefaultRequestHeaders.Add("User-Agent", "TemporaryUserAgent");
    client.DefaultRequestHeaders.Add("Accept", "application/json");
    string baseUrl = "https://rest.opensubtitles.org/search";

    //episode (number)
    if (episodeNumber.HasValue)
        baseUrl += "/episode-" + episodeNumber.Value.ToString();
    //imdbid (always format it as sprintf("%07d", $imdb)
    if (!String.IsNullOrEmpty(imdbId))
    {
        imdbId = imdbId.Substring(2);
        baseUrl += "/imdbid-" + imdbId;
    }
    //moviebytesize (number)
    if (movieByteSize.HasValue)
        baseUrl += "/moviebytesize-" + movieByteSize.Value.ToString();
    //moviehash (should be always 16 character, must be together with moviebytesize)
    if (!String.IsNullOrEmpty(movieHash))
        baseUrl += "/moviehash-" + movieHash;
    //query (use url_encode, make sure " " is converted to "%20")
    if (!String.IsNullOrEmpty(movieName))
    {
        movieName = HttpUtility.UrlEncode(movieName);
        baseUrl += "/query-" + movieName;
    }
    //season (number)
    if (seasonNumber.HasValue)
        baseUrl += "/season-" + seasonNumber.Value.ToString();
    //sublanguageid (if ommited, all languages are returned)
    if (!String.IsNullOrEmpty(languageCode))
        baseUrl += "/sublanguageid-" + languageCode;

    string response = await client.GetStringAsync(baseUrl);
    var results = JsonConvert.DeserializeObject<dynamic>(response);
    List<Subtitle> subtitles = new List<Subtitle>();
    foreach (var result in results)
    {
        Subtitle subtitle = new Subtitle();
        subtitle.SubFileName = result.SubFileName;
        subtitle.MovieReleaseName = result.MovieReleaseName;
        subtitle.DownloadLink = result.SubDownloadLink;
        subtitle.ManualLink = result.SubtitlesLink;
        subtitle.Language = result.LanguageName;
        subtitle.LanguageCode = result.SubLanguageID;
        subtitle.SubFormat = result.SubFormat;
        subtitles.Add(subtitle);
    }
    return subtitles;
```

Slika 27 – Prikaz funkcije SearchSubtitles unutar klase SubtitlesRepository

Lista mogućih podnaslova se prikazuje korisniku koji izabire željeni podnaslov. Odabrani podnaslov biva spremljen na putanju na kojoj se nalazi video datoteka (Slika 28.).

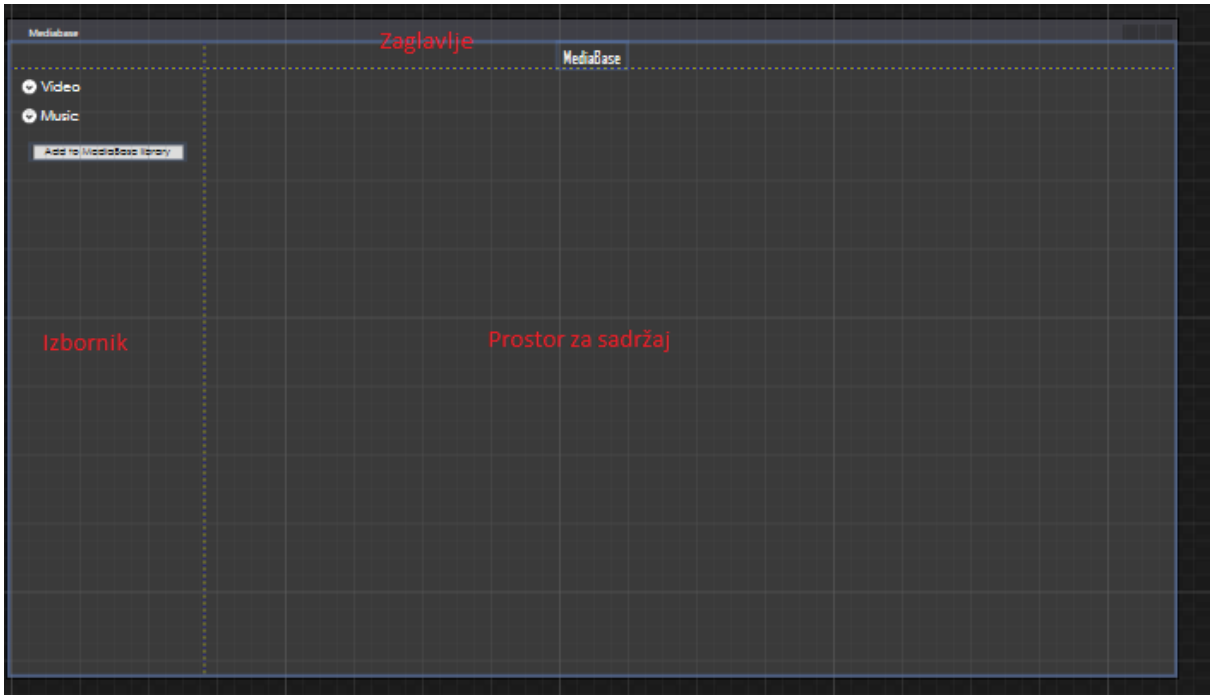
```
public async void DownloadSubtitle(string link, string filename, string extension)
{
    using (var client = new WebClient())
    {
        byte[] subtitleGzip = await client.DownloadDataTaskAsync(link);
        using (MemoryStream compressedStream = new MemoryStream(subtitleGzip))
        {
            using (GZipStream gZipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
            {
                using (MemoryStream uncompressedStream = new MemoryStream())
                {
                    await gZipStream.CopyToAsync(uncompressedStream);
                    uncompressedStream.Position = 0;
                    using (FileStream fileStream = new FileStream(filename + extension, FileMode.OpenOrCreate))
                    {
                        await uncompressedStream.CopyToAsync(fileStream);
                        await fileStream.FlushAsync();
                    }
                }
            }
        }
    }

}
```

Slika 28 – Prikaz funkcije DownloadSubtitle unutar klase SubtitlesRepository

3.4 Korisničko sučelje (UI)

Korisničko sučelje je WPF projekt unutar aplikacije i sastoji se od jednog glavnog prozora. Taj glavni prozor je podjeljen na sekcije (Slika 29.):

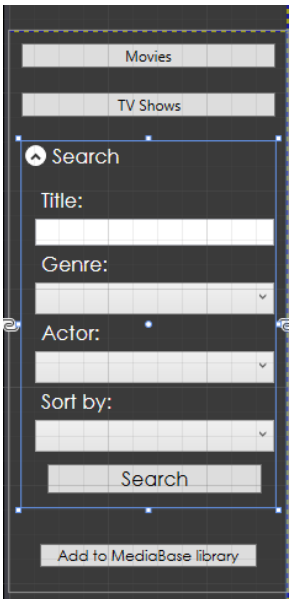
- Zaglavlje
- Izbornik
- Prostor za sadržaj



Slika 29

Navedeni djelovi koristi XAML i GridExtra paket za prikaz, te uz to svaka ima svoju C# klasu u kojoj se pozivaju funkcije iz repozitorija. Uz to u njima je sadržan kod koji se koristi za promjene tokom izvođenja (eng. runtime).

3.4.1 Izbornik



Slika 30 – Prikaz izbornika

Izbornik se sastoji od dva gumba (eng. button) „Movies“ i „Shows“ koji služe za izbor prikaza sadržaja, filmovi ili serije. Ovisno o izboru korisnika sučelje prikazuje traženi sadržaj u istoimeni prostor za sadržaj označen na slici 29.

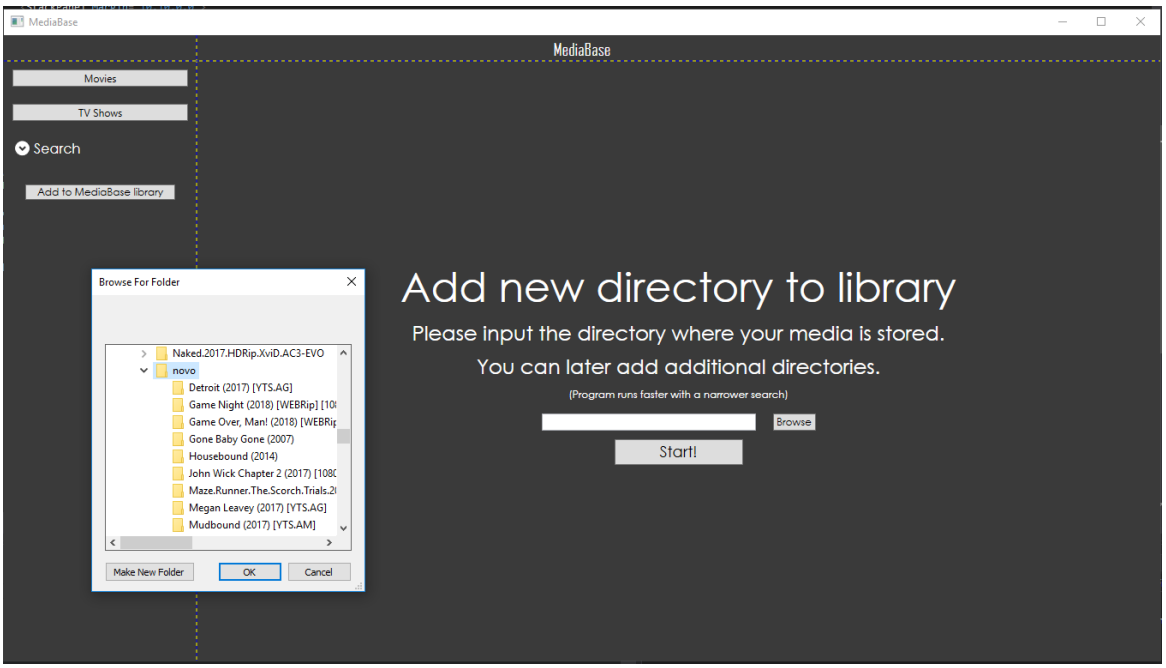
Ispod njih se nalazi produživač (eng. expander) nazvan „Search“ koji klikom prikazuje polja na temelju kojih korisnik može filtrirati, sortirati i pretraživati sadržaj. Padajuće liste „Genre“, „Actor“ i „Sort by“ se ispune pri paljenju aplikacije.

Na dnu izbornika je gumb „Add to MediaBase library“ čijom se aktivacijom se pojavljuje izbornik u prostoru za sadržaj za unos nove putanje koja sadržava video datoteke.

3.4.2. Prostor za sadržaj

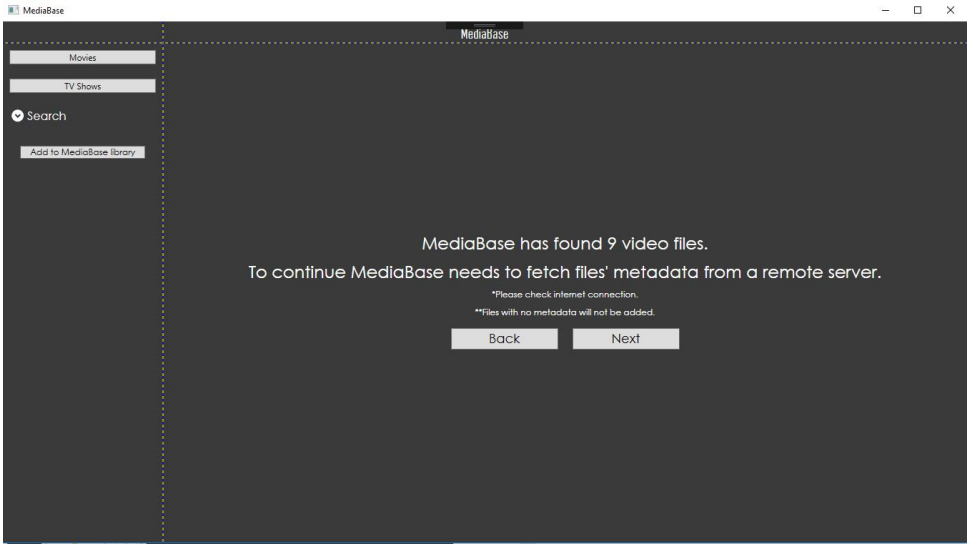
Prostor za sadržaj je prazan prostor u koji se učitavaju korisničke kontrole (eng. User Control) ovisno o zahtjevima korisnika. Korisničke kontrole su prikazi sadržaja, te tako prikazuju listu filmova/serija, pojedinačan film/seriju i izbornik za dodavanje nove putanje.

Kada se aplikacija upali po prvi puta i u bazi još nema sadržaja, prva kontrola koja se pojavljuje je ona za dodavanje nove putanje. Upućuje korisnika da unese putanju za pretraživanje video datoteka (Slika 31.).



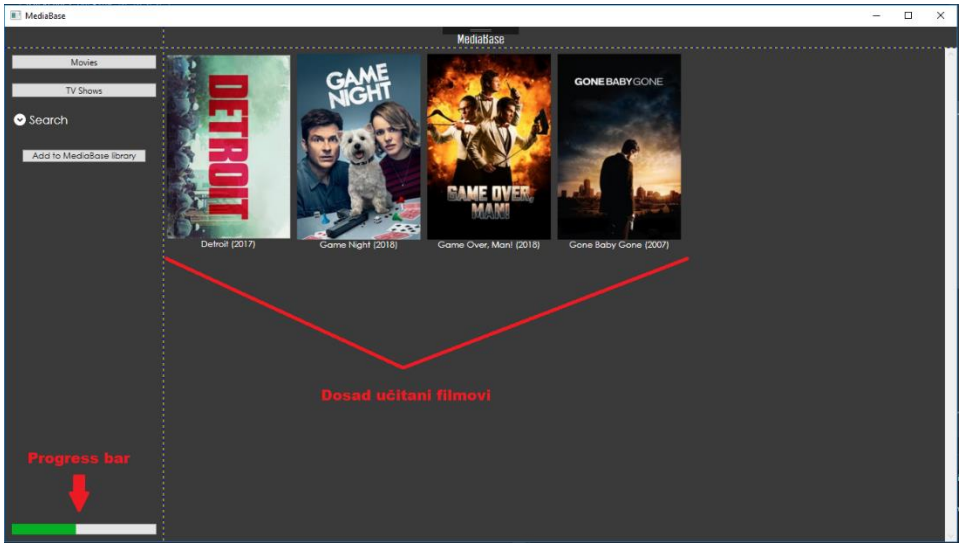
Slika 31 – Odabir putanje

Klikom na „Start!“ aplikacija javlja koliko je video datoteka pronađeno na toj putanji i upozorava da je potrebna Internet veza za dohvaćanje meta podataka (Slika 32.).



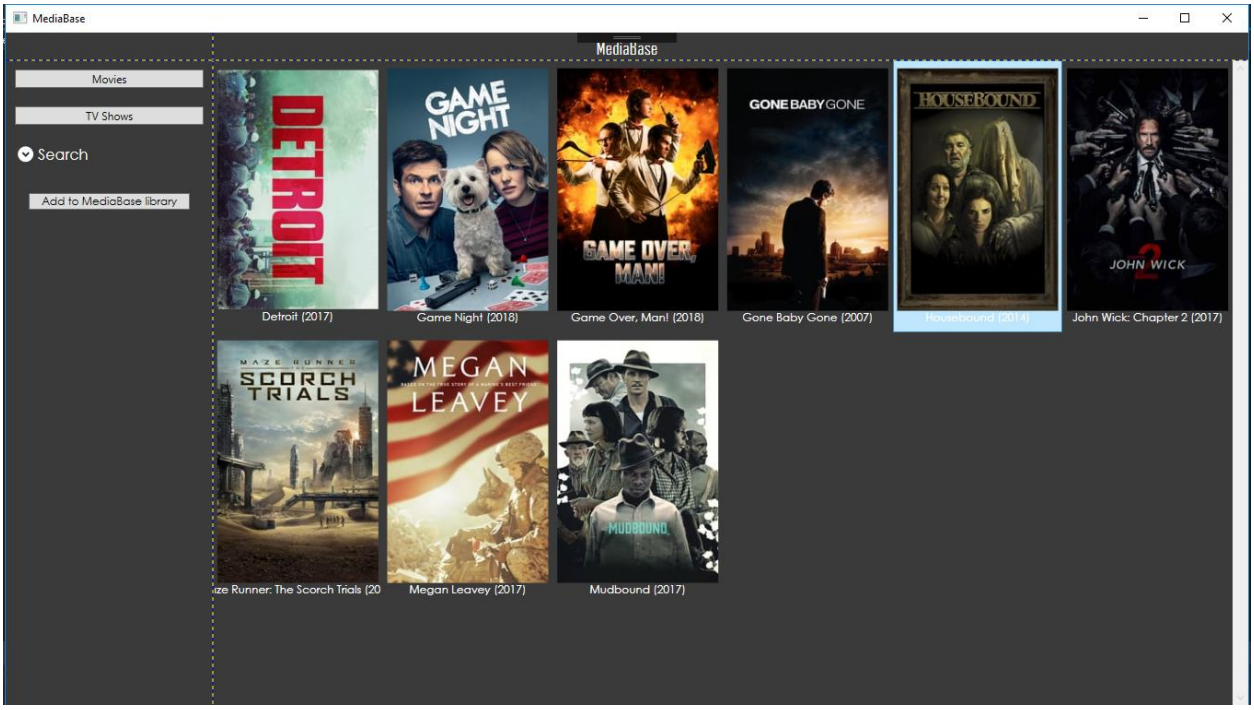
Slika 32 – Početak učitavanja meta podataka

Zahvaljući asinkronim (eng. asynchronous) funkcijama unutar repozitorija, aplikacija je funkcionalna i dok se dohvaćaju meta podaci sa Interneta i spremaju u bazu podataka. Što znači da svakim novim dodavanjem, datoteka je automatski vidljiva i upotrebljiva korisniku dok se ostatak učitava. Za vrijeme učitavanja je vidljiva traka za napredak (eng. progress bar) na dnu izbornika (Slika 33).



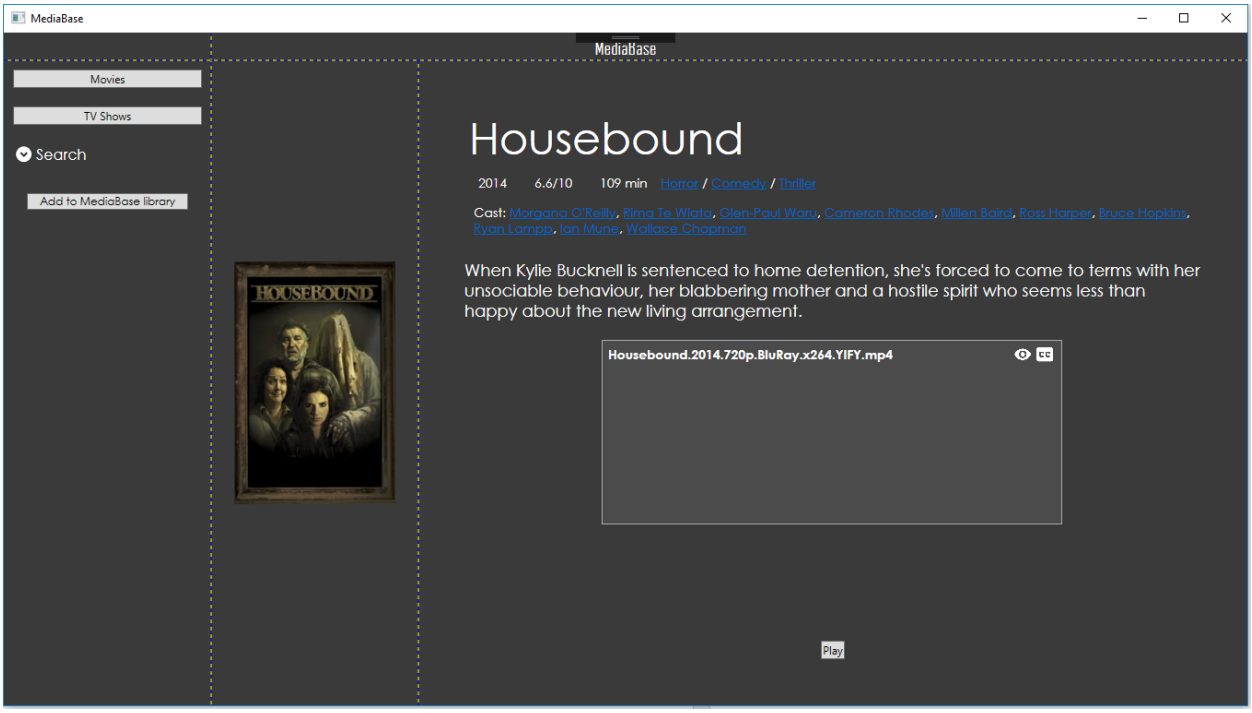
Slika 33 – Učitavanje meta podataka na temelju datoteka sa putanje

Lista filmova/serija se učitava ovisno o korisničkom odabiru u izborniku, dakle lista filmova, serija ili lista sa rezultatima ciljane pretrage. Rezultat odabira prikazuje poster filma/serije, naziv i godinu izdavanja. Te tri komponente zajednički tvore gumb (Slika 34.).



Slika 34 – Prikaz liste filmova

Klikom na određeni film/seriju zatvara se korisnička kontrola za prikaz liste, te se pokreće kontrola za prikaz pojedinačnog filma ili serije. Kontrola prikazuje naziv filma/serije, poster, godinu izdavanja, relativnu ocjenu, vrijeme trajanja, popis žanrova, popis glumaca, kratki opis radnje i popis datoteka povezanih sa tim filmom/serijom. Popis datoteka kod serija je grupiran po sezonama, dok se kod filmova prikazuje samo jedan popis sa svim datotekama koje su povezane sa filmom (ukoliko korisnik ima više verzija filma ili ako je film podijeljen na više datoteka – CD1, CD2..). Uz to se može kliknuti na poveznicu (eng. hyperlink) jednog od žanrova ili glumaca kako bi se u prostoru za sadržaju prikazali svi filmovi/serije u kojima glumi određeni glumac ili koji su određenog žanra. Također klikom na ikonu oka korisnik može označiti datoteku kao pogledanu i klikom na ikonu „CC“ se otvara izbornik za preuzimanje podnaslova sa funkcijama SubtitlesRepository-ja (Slika 35.).



Slika 35 – Prikaz pojedinog filma

4. Zaključak

Mediabase je Windows računalna aplikacijom koja svojom funkcionalnošću i jednostavnošću omogućava korištenje i gledanje filmova i serija na računalu. Aplikacija se sastoji od nekoliko programskih knjižnica, baze podataka, grafičkog sučelja. Sama primjena aplikacije ukazuje na potrebu za razvojem korisnih i kvalitetnih računalnih aplikacija koje će korisnicima olakšati svakodnevni život.

Razvoj računalne aplikacije zahtjeva poznavanje određenih programskih jezika, programskih paketa i modula, struktura podataka, relacijskih baza podataka, ekstrahiranja informacija iz podataka, osmišljavanja sučelja i logike funkcioniranja istoga. Korisnost i funkcionalnost same aplikacije ograničena je samo inicijalnom idejom kao rješenjem za problem koji se isplati riješiti. Ukoliko je aplikacija dovoljno dobra te se može na neki način monetizirati, otvara se prilika za razvoj proizvoda. Kada više ljudi radi na različitim komponentama samih aplikacija, razvoj bi mogao ići brže te bi se brže mogli rješavati realni problemi i na taj način potencijalno otvarale veće mogućnosti za zaradu u različitim industrijama.

Informatika je kao grana zahvatila apsolutno svaku industriju te predstavlja jednu izuzetno interdisciplinarnu disciplinu u kojoj nudi rješenja za svakakve probleme. Informacijski sustavi koriste se u financijskom sektoru, poljoprivredi, trgovini, uslužnom sektoru, a svaki informacijski sustav zapravo predstavlja konkretan skup računalnih aplikacija koje olakšavaju poslovanje i omogućavaju lakše baratanje informacijama i pristup istima.

U današnjem svijetu, razvoj računalnih aplikacija konceptualno je vrlo sličan razvoju mobilnih aplikacija, što omogućava puno prostora za razvoj različitih proizvoda i izvoz na Europsko ili čak globalno tržište. Može se primijetiti kako hrvatska IT industrija te neke njena najpoznatijim poduzećima rastu prihodi kao i broj zaposlenika posljednjih godina, dok su pritome izuzetno poželjni kao poslodavci mladim ljudima. Razvoj računalnih i mobilinih aplikacija jedna kao od grana kojima se IT industrija bavi, predstavlja priliku i potencijal koji treba iskoristiti kako bi svojim znanjem i sposobnošću učinili život u neposrednom okruženju boljim.

5. Popis literature

Bootstrap. n.d. *Documentation.* Pokušaj pristupa 26. 3 2019. https://getbootstrap.com/docs/4.3/getting-started/introduction/.

clem6ever. 2016. *torrent-name-parser*. Pokušaj pristupa 26. 3 2019. https://www.npmjs.com/package/torrent-name-parser.

Janczuk, Tomas. 2018. *Edge.js: .NET and Node.js in-process.* 3. 5. Pokušaj pristupa 26. 3 2019. https://github.com/tjanczuk/edge.

Jones, Mike, Petr Kulikov, i Maira Wenzel. 2018. *Windows Presentation Foundation.* 25. 1. Pokušaj pristupa 26. 3 2019. https://docs.microsoft.com/en-us/dotnet/framework/wpf/.

KODI. 2018. *Kodi Wiki.* 10. 10. Pokušaj pristupa 10. 5 2019. https://kodi.wiki/view/Main_Page.

Microsoft. n.d. *Entity Framework Documentation.* Pokušaj pristupa 26. 3 2019. https://docs.microsoft.com/en-us/ef/#pivot=entityfmwk&panel=entityfmwk1.

—. n.d. *SQL Server 2017 on Windows and Linux.* Pokušaj pristupa 29. 3 2019. https://www.microsoft.com/en-us/sql-server/sql-server-2017.

minami_SC. 2017. *sourcechord/GridExtra: Custom panel controls for WPF/UWP*. 29. 10. Pokušaj pristupa 26. 3 2019. https://github.com/sourcechord/GridExtra.

Mohapatra, Manas. 2018. *JSON Serialization And Deserialization Using JSON.NET Library In C#.* 20. 11. Pokušaj pristupa 26. 3 2019. https://www.c-sharpcorner.com/UploadFile/manas1/json-serialization-and-deserialization-using-json-net-librar/.

Newtonsoft. n.d. *Json.NET.* Pokušaj pristupa 26. 3 2019. https://www.newtonsoft.com/json.

OpenSubtitles. n.d. *Subtitles API.* Pokušaj pristupa 29. 3 2019. https://trac.opensubtitles.org/projects/opensubtitles/wiki/XMLRPC.

PLEX. n.d. *Support Articles.* Pokušaj pristupa 10. 5 2019. https://support.plex.tv/articles/.

Rouse, Margaret. 2006. *Microsoft SQL Server* . 24. 1. Pokušaj pristupa 29. 3 2019. https://searchsqlserver.techtarget.com/definition/SQL-Server.

TheMovieDatabase. n.d. *Getting Started.* Pokušaj pristupa 26. 3 2019. https://developers.themoviedb.org/3/getting-started/introduction.

TutorialsPoint. n.d. *Entity Framework - Overview.* Pokušaj pristupa 26. 3 2019. https://www.tutorialspoint.com/entity_framework/index.htm.

UserLand Software. 1999. *Home.* 14. 6. Pokušaj pristupa 29. 3 2019. http://xmlrpc.scripting.com/.

WebRTC. n.d. *WebRTC*. Pokušaj pristupa 10. 5 2019. https://webrtc.org/faq/.