VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DATABÁZE FILMŮ A SERIÁLŮ

DATABASE OF MOVIES AND SERIALS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ VELECKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19957/2016/xvelec05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Velecký Lukáš**
Obor: Informační technologie
Téma: **Databáze filmů a seriálů**
Database of Movies and Serials
Kategorie: Databáze

Pokyny:

1. Seznamte se s platformou .NET, databází Microsoft SQL Server a jazykem Transact SQL.
2. Prostudujte API, které poskytují služby Trakt.tv, OMDb, CSFD.cz a dalšími.
3. Navrhněte databázi. která bude evidovat získaná data o filmech a seriálech, včetně titulků a písní, které zazněly v těchto filmech/seriálech, popř. knih, ze kterých daný film vychází. Dále navrhněte aplikaci, která bude s touto databází pracovat a vhodným způsobem zobrazovat data.
4. Navrženou databázi i aplikaci implementujte.
5. Ověřte funkčnost řešení na vhodném vzorku dat.
6. Zhodnoťte dosažené výsledky a další možná pokračování v tomto projektu.

Literatura:

- Welling, L., Thomsonová, L.: PHP a MySQL: rozvoj webových aplikací. Vyd. 1. Praha: SoftPress, 2003, 910 s. ISBN 80-86497-60-7.
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015. ISBN: 978-80-251-4573-9

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem bakalářské práce je návrh a implementace aplikace pro správu filmů a seriálů s využitím aplikačního programového rozhraní webových serverů. V práci jsou popsány použité technologie a možnosti získávání dat z API. Dále je v práci probrán samotný návrh databáze. Výsledkem je aplikace implementovaná pod technologií .NET v jazyce `C#` s využitím návrhového vzoru Model-View-ViewModel a databázového serveru Microsoft SQL Server.

Abstract

The goal of the bachelor thesis is to design and implement the application for movie and serial management using the application program interface of web servers. The thesis describes the used technologies and possibilities of data acquisition from the API. Furthermore, the database design itself is discussed. The result is an application implemented under .NET technology in `C#` using the Model-View-ViewModel pattern and the Microsoft SQL Server as database server.

Klíčová slova

.NET, C#, WPF, Transact SQL, API, databáze, Model-View-ViewModel

Keywords

.NET, C#, WPF, Transact SQL, API, database, Model-View-ViewModel

Citace

VELECKÝ, Lukáš. *Databáze filmů a seriálů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bartík Vladimír.

Databáze filmů a seriálů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D. Další informace mi poskytl Ing. Roman Jašek. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Velecký
14. května 2017

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce Ing. Vladimíru Bartíkovi, Ph.D. za jeho vedení a cenné rady při vypracování, a za možnost umožnění práce na vlastním tématu.

Obsah

1 Úvod **3**

2 Použité technologie **4**
2.1 Alternativní technologie pro tvorbu aplikací 4
2.2 Platforma .NET 4
2.3 Jazyk C# 5
2.4 Windows Presentation Foundation 7
2.5 Extensible Apllication Markup Language 7
2.6 Model-View-ViewModel 8
2.7 Entity Framework 9
2.8 Microsoft SQL Server a služba LocalDB 9

3 Možnosti získávání dat za pomoci API **10**
3.1 Trakt API 11
3.2 The Movie Database API 11
3.3 Další možnosti API 12
3.4 Formáty dat používané v komunikaci API 12
3.4.1 XML 12
3.4.2 JSON 13

4 Návrh aplikace **15**
4.1 Webové portály 15
4.2 Offline aplikace 15
4.3 Specifikace požadavků 16

5 Implementace **20**
5.1 Databáze 20
5.1.1 Migrace databáze 20
5.2 Projekt Model 21
5.2.1 Popis entit 21
5.3 Projekt ViewModel 23
5.3.1 ViewModely 23
5.3.2 Příkazy 23
5.4 Projekt Desktop 23
5.4.1 Prvky pro zobrazení 24
5.4.2 Konvertory 24
5.4.3 Soubor App.xaml a Resources 24
5.5 Projekt Repository 25

5.6 Projekt TraktAPI 25
5.7 TmdbAPI 27

6 Testování **28**

7 Závěr **29**

Literatura **30**

Přílohy **33**

A ER diagramy **34**

Kapitola 1

Úvod

Hlavním cílem této práce je navrhnout a implementovat aplikaci pro správu seriálů a filmů, které uživatel sleduje nebo se zajímá. Kromě toho má aplikace pomoci spravovat propojení nejen těchto seriálů a filmů, ale také souvisejících knížek (knižní předloha, navazující dílo, ...) a hudby k filmu či seriálu (soundtrack). Aplikace by měla především zjednodušit uživateli správu seriálů či filmů, které sleduje. Tedy ukládat informace o tom, které epizody již uživatel shlédnul, které má sledovat, nebo které seriály teprve chce shlédnout. Kromě toho by aplikace měla také umožnit zaznamenat příslušné vazby na další média nebo osoby.

V práci se nejprve v kapitole 2 věnuji samotným použitým technologiím, proč jsem tyto technologie použil a také, které další technologie lze použít místo zvolených. V kapitole 3 stručně popisuji webové API, dále se věnuji konkrétním službám Trakt API a TMDb API, jejich alternativám a dalším službám, poskytující API. V rámci kapitoly také popisuji formáty dat pro přenos informací přes tyto služby. Dále je v kapitole 4 popsán samotný návrh, v němž se věnuji některým podstatným částem samotného návrhu a proč došlo ke konkrétním řešením či úpravám. Také se v této kapitole věnuji alternativám tohoto programu, částečně také inspiraci pro vznik této aplikace. Samotnou implementaci popisuji v kapitole 5. V té se zabývám a postupně popisuji všechny hlavní celky této aplikace.

Kapitola 2

Použité technologie

Pro implementaci jsem se rozhodl využít platformu *.NET Framework*. Aplikaci jsem napsal v jazyce `C#`. Kromě něj jsem využil také technologie *Language Integrated Query* (LINQ) pro dotazování, knihovnu *Windows Presentation Foundation* (WPF) pro tvorbu grafického rozhraní a knihovnu *Entity Frameworku* pro objektově-relační mapování. V aplikaci využívám návrhový vzor pro WPF aplikace *Model-View-ViewModel* (MVVM).

Pro databázi jsem použil variantu databázového systému Microsoft SQL Server *LocalDB*. Jako dotazovací jazyk je zde použito rozšíření dotazovacího jazyka SQL *Transact-SQL*.

Pro tyto technologie jsem se rozhodl z několika důvodů. Prvním a nejdůležitějším důvodem byla znalost některých technologií, především jazyka `C#`. S Entity Frameworkem a návrhovým vzorem MVVM jsem se setkal již v rámci školního projektu. Nejnovější technologií pro mě byl hlavně Microsoft SQL Server. S dotazovacím jazykem SQL jsem již pracoval, proto přizpůsobení na Transact-SQL nebyl velký problém. Jako další důvod bych uvedl, že mě dané technologie zajímají a chtěl bych se jim věnovat.

2.1 Alternativní technologie pro tvorbu aplikací

Klasické desktopové aplikace tvoří velkou část celkového vývoje aplikací. Proto pro tyto účely vzniklo mnoho knihoven pro tvorbu uživatelského prostředí nad mnoha jazyky. Pro jazyk C lze zmínit například knihovnu GTK, dále třeba knihovnu MFC pro C++ od firmy Microsoft nebo knihovnu Qt. Knihovny GTK nebo Qt mají své alternativy např. i pro jazyky PHP nebo Python. Pro jazyk Java potom existují například knihovny AWT nebo Swing.

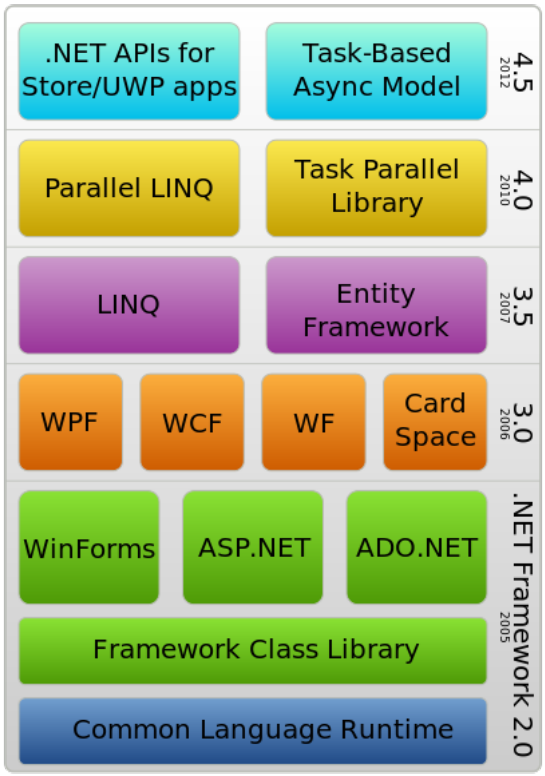
Pro databázový systém by šlo kromě LocalDB použít také SQLite. Tento systém nepracuje jako samostatný proces, ale databáze využívá k ukládání soubor na disku a samotný SQLite je šiřitelný např. jako DLL knihovna.

2.2 Platforma .NET

.NET Framework je platforma společnosti Microsoft. Platforma je určena primárně pro operační systémy Windows. Lze na ni vyvíjet jak klasické či nativní aplikace pro Windows, tak webové stránky nebo aplikace pro mobilní systémy Windows. V dnešní době ale vzniká nová verze .NET Core, která umožňuje vývoj na platformách Linux a MacOS. [30] Kromě toho také existuje Xamarin umožňující vývoj .NET aplikací pro Android nebo iOS. Platforma

umožňuje vývoj aplikací v mnoha programovacích jazycích, např. jazyky *C#*, *Visual Basic*, nebo *F#*. Aplikace jsou spouštěny nad virtuálním strojem Common Language Runtime (CLR). Díky tomu je možné v při vávoji použít více programovacích jazyků. Tato technologie překládá zdrojový kód aplikace před spuštěným do nativního kódu cílové architektury. Kromě toho CLR poskytuje služby pro správu paměti, načítání potřebných knihoven nebo zachytávání výjimek. Mezi významné knihovny platformy .NET patří ASP.NET (pro tvorbu webových stránek), ADO.NET (knihovna pro práci s databází), WPF (nahradila starší Windows Forms pro tvorbu uživatelského rozhraní) a další. [22]

V rámci .NET byl definován tzv. Common Type System. Ten definuje, jak jsou v rámci CLR deklarovány, používány a spravovány datové typy. To umožňuje sdílení datových typů nejen v rámci aplikací, ale také jazyků pod platformou. Kromě toho také zabezpečuje typovou bezpečnost mezi jazyky či vyšší rychlost vykonání příkazů. [2]



Obrázek 2.1: Struktura platformy .NET [23]

2.3 Jazyk C#

Jazyk C#, je stejně jako platforma .NET, vytvořen a spravován firmou Microsoft. Vývoj a vznik jazyku C# byl výrazně ovlivněn především jazyky C++ (případně i jazykem C) a Java. Dle standardu ECMA-334 se jedná o jednoduchý, moderní, objektově orientovaný a typově bezpečný jazyk. [20] Podporuje koncepty zapouzdření, dědičnosti a polymorfismu. Všechny proměnné a metody jsou zapouzdřeny do tříd, včetně vstupního bodu aplikace. Stejně jako jazyk Java může třída dědit maximálně z jedné nadřazené třídy. Rozhraní ovšem může

implementovat neomezeně. [26] Syntaxe jazyka obsahuje datové typy s možnou hodnotou null (nullable), výčty, delegáti nebo lambda výrazy.

Jazyk podporuje dva druhy datových typů - hodnotové a referenční. Do první kategorie patří primitivní datové typy (celočíselný datový typ, řetězec, float...), struktury a výčtové typy (enum). Struktury mají podobnost s třídami, na rozdíl od nich ale nemohou dědit ani být děděny, mohou ovšem implementovat rozhraní či konstruktory a další metody. Referenční datové typy uchovávají odkaz na místo v paměti, kde se daný objekt nachází. Při deklaraci proměnných jde použít klíčové slovo `var`. Kompilátor si sám podle operace na pravé straně určí výsledný datový typ.

Z hlediska atributů objektu se v jazyce rozlišují vlastnosti (property) a pole (field). Zatímco pole je jednoduchý atribut pro uchování stavu objektu, obvykle s modifikátorem přístupu `private` nebo `protected`. [28] Zpřístupnění polí jiným objektům probíhá buď přes metody třídy nebo právě přes vlastnost. Na rozdíl od pole je vlastnost většinou s modifikátorem public. Vlastnost může uchovávat stav objektu nebo přes ni můžou ostatní objekty přistupovat k poli. Kromě samotného přístupu nemusí docházet jen k zveřejnění, ale metoda pro zápis či čtení může tyto hodnoty upravovat, nebo upozorňovat na aktualizaci pole. Této vlastnosti využívá i Entity Framework pro aktualizaci pohledu. Od verze `C#` 3.0 lze vytvořit tzv. automaticky implementované vlastnosti (Auto-Implemented Properties), kdy tělo funkcí `get` a `set` je doplňováno až při kompilaci. [27]

```
public class SeasonModel : BaseModel
{
    private int _show;
    public int Show
    {
        get { return _show; }
        set
        {
            if (_show == value) return;
            _show = value;
            OnPropertyChanged();
        }
    }
}
```

Algoritmus 2.1: Ukázka části třídy `SeasonModel`

Na algoritmu 2.1 lze vidět vlastnost `Show`, která zpřístupňuje pole `_show`. Při vkládání nové hodnoty přes metodu `set` také dochází k upozornění na změnu voláním metody `OnPropertyChanged()`.

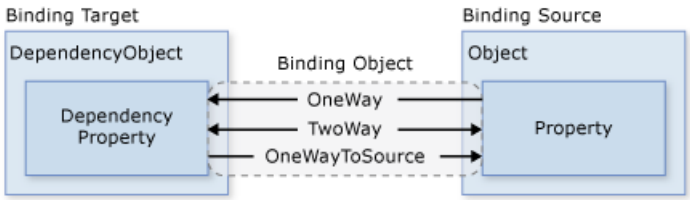
Další užitečnou vlastností jazyka jsou tzv. generika (Generics). Ty jsou součástí jazyka od verze 2.0.[29] Generika umožňují vytvářet třídy a jejich metody znovupoužitelné pro více objektů současně. Při implementaci objektu nemusíme znát typ parametru, ten poté následně určíme při vytváření instance třídy. V rámci generické třídy je poté možné vytvářet nové instance třídy, která je určena přes generický parametr T. Jedna třída může mít více generických parametrů, oddělených čárkami. Generický parametr je uváděn ve špičatých závorkách za názvem třídy. Upřesnění pro generické parametry se může nacházet za klíčovým slovem `Where`. Generiky se využívají především pro tvorbu kolekcí, např `ObservableCollection<T>`. Na první pohled se můžou generika podobat rozhraním, je zde ale rozdíl mimo jiné v tom, že nelze vytvořit instanci rozhraní.

2.4 Windows Presentation Foundation

Windows Presentation Foundation (WPF) je knihovna pro tvorbu grafického rozhraní pod platformou .NET. Technologii tvoří vektorově založené jádro nezávislé na rozlišení. Grafika WPF funguje pomocí DirectX knihovny. Pro vytváření uživatelského rozhraní technologie využívá značkovací jazyk XAML. Díky tomu je možné oddělit vzhled a funkčnost aplikace od sebe. Přestože je možné v rámci WPF implementovat i Code Behind, v rámci této aplikace jsem od tohoto postupu ustoupil a veškerá funkčnost aplikace je mimo WPF. [6]

Důležitou vlastností WPF je tzv. vázaní (Data Binding). Vázání poskytuje mechanismus pro synchronizaci dat s uživatelským rozhraním. Směry tohoto vázaní jsou čtyři druhy: [3]

- **OneTime** - hodnota se zobrazí pouze při vykreslení, i při změně zdroje nedojde k aktualizaci
- **OneWay** - jedná se o jednosměrnou synchronizaci zdroj → cíl, v případě aktualizace na straně cíle nedojde ke změně ve zdroji, využívá se např. pro zobrazení hodnot jen pro čtení
- **TwoWay** - jde o obousměrnou synchronizaci, pokud proběhne změna v cíli, aktualizuje se i zdroj a naopak, tohoto se využívá v situaci, kdy uživatel může upravovat stav objektu
- **OneWayToSource** - je opakem OneWay, tedy cíl aktualizuje zdroj, ale zdroj již nezasahuje do cíle
- Výchozí hodnotou pro uživatelsky zapisovatelné prvky je TwoWay druh, ostatní se vážou OneWay



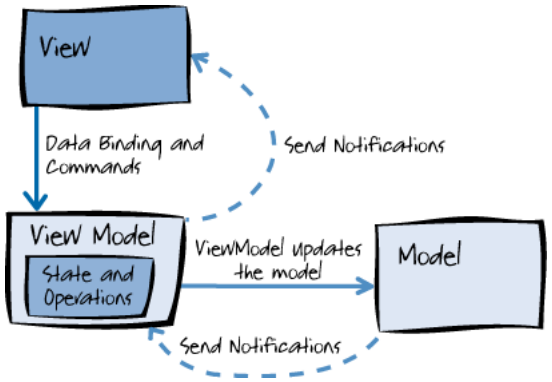
Obrázek 2.2: Ilustrace Data Binding [3]

2.5 Extensible Apllication Markup Language

Jazyk Extensible Apllication Markup Language (XAML) je deklarativní značkovací jazyk firmy Microsoft založený na jazyku XML. V rámci platformy .NET slouží k deklaraci a inicializaci objektů pro uživatelské rozhraní, včetně jejich vlastností a událostí. Obsahuje stromovou hierarchii těchto objektů, kterým říkáme elementy. Tato struktura vždy začíná kořenovým prvkem (např. `Window`, `UserControl`). Každý element je vždy potřeba ukončit, buď přes párovou značku (např. `<TextBlock> </TextBlock>`) nebo v rámci jedné značky `<TextBlock />`. [16]

2.6 Model-View-ViewModel

Model-View-ViewModel (MVVM) je návrhový vzor vyvinutý pro práci s WPF. Vychází z návrhového vzoru Model-View-Controller (MVC). Smyslem návrhových vzorů je především přehledný a snadno upravitelný výsledný kód. Jednoduchá změna v návrhu tedy neznamená přepis celé aplikace. Návrhový vzor neřeší konkretní situaci. Jedná se spíše o šablonu nebo popis, jak je daný problém možno řešit, aby nedocházelo k problémům při dalším vývoji. Návrhové vzory MVVM a MVC se řadí mezi vzory pro implementaci uživatelského prostředí. [19]



Obrázek 2.3: Komponenty návrhového vzoru MVVM a interakce mezi nimi [11]

První vrstva návrhového vzoru Model představuje samotná data, se kterýma bude aplikace pracovat. Ta znázorňují stav objektu nebo obsah databáze. V rámci modelu se neděje žádná logika běhu aplikace a nemá informace o stavu jiných částí. V případě aplikace se jedná o data centrický přístup, tedy model znázorňuje obsah databáze přes třídy jazyka `C#`, resp. tabulek SQL databáze.

Uživatelské prostředí je implementováno v rámci vrstvy View. Jedná se tedy o grafické znázornění informace a je to vrstva, přes kterou uživatel komunikuje s programem (vstup, výstup, atd.). View může být vytvořeny pod technologií WPF jazykem XAML a neobsahuje v souvislosti s návrhovým vzorem žádnou Code Behind logiku, tak jako je tomu u této aplikace.

Poslední vrstva ViewModel spojuje právě tyto dvě vrstvy. View je s ViewModelem spojeno pomocí data binding a ViewModel poskytuje View svůj obsah k zobrazení. ViewModel zajišťuje logiku aplikace. ViewModel umožňuje spouštění příkazů na straně View a upozorňuje View na změny v obsahu dat.

Upozorňování na změny v obsahu dat probíhá díky rozhraním `INotifyPropertyChanged` a pro kolekce rozhraní `INotifyCollectionChanged`. Zde je využito právě dvou typů atributů jazyka `C#` - vlastností a polí. Kromě toho, že vlastnost zpřístupňuje příslušné pole, tak právě přes implementaci rozhraní `INotifyPropertyChanged` událostí `PropertyChanged` upozorňuje View o změně, a View se tedy následně překreslí.

Pro kolekce prvků se v návrhovém vzoru používá třída `ObservableCollection<T>`. Oproti ostatním kolekcím třída implementuje rozhraní `INotifyCollectionChanged`. Díky tomu kolekce poskytuje oznámení o přidání či mazání prvků nebo v případě, kdy je kolekce obnovena. Třída poskytuje velké množství metod, především přes třídu `Collection<T>`, ze které třída dědí, např. funkce `Add(T)`, `Delete(T)`, `Contains(T)`. Mezi další užitečné me-

tody této třídy patří například metoda `Any<T>()`, resp. `Any<T>(Func<T,Boolean>)`, která umožňuje zjistit, jestli kolekce obsahuje nějaký prvek (případně nějaký vyhovující podmínce), metoda All<T>(Func<T,Boolean>) zjišťující, jestli všechny prvky vyhovují podmínce nebo třeba metoda `First<T>(Func<T,Boolean>)` vracející první prvek odpovídající podmínce.

2.7 Entity Framework

Entity Framework je technologie pro objektově relační mapování pro ADO.NET od firmy Microsoft. Entity Framework byl součástí .NET Frameworku, od verze 6 už součástí není a jedná se o open source projekt. Framework zjednodušuje práci s databázi pro vývojáře, ten není nucen psát složíte kódy pro přístup k databázi. Dotazy nad databází může provádět stále pomoci jazyka SQL, ale EF umožňuje využití i dotazovacího jazyka LINQ. EF je užitečný ve 3 situacích. V prvním případě se jedná o situaci, kdy již máme existující databázi, nebo chceme návrh uskutečnit před zbytkem aplikace. Při druhé situaci jsou nejdříve vytvořeny třídy a z nich se poté vytvoří odpovídající databáze. Třetí přístup počítá s využitím nástroje pro grafický návrh a z něj se poté vytváří databáze a příslušné třídy. [14]

2.8 Microsoft SQL Server a služba LocalDB

Microsoft SQL Server (MSSQL Server) je relační systém řízení báze dat (SŘBD) vyvinutý firmou Microsoft (původně se společností Sybase). Systém nabízí především technologie pro uchovávání dat, ale také jejich zabezpečení, transformaci dat do přehledů či analýzu dat pomocí jazyka R, určenému pro statistické účely. Kromě relačního SŘBD se jedná také o objektově relační systém řízení báze dat. Dotazovacím jazykem systému je jazyk Transact-SQL. Systém je primárně určený pro operační systémy firmy Microsoft. Od roku 2016 je také k dispozici beta verze pro operační systém Linux.[21] Systém lze ovládat pomocí grafického rozhraní (např. přes Visual Studio či Microsoft SQL Server Management Studio). Systém lze spravovat také pomocí příkazů a skriptů napsaných v jazyce Transact-SQL. Microsoft SQL Server je k dispozici v několika variantách - Enterprise, Standart, Express a Developer. [8]

Pro aplikaci je důležitá verze Express. Od verze MSSQL Server 2015 pod verzi Express funguje i verze LocalDB. Jedná se o odlehčenou verzi SQL Serveru primárně pro účely vývoje aplikací, lze ji ale použít i v rámci produkční fáze. [5] Přestože se jedná o odlehčenou verzi, poskytuje velkou funkcionalitu včetně procedur. Na rozdíl od ostatních verzí se jedná o verzi určenou pro lokální účely, databáze je určena vždy pro konkrétního uživatele operačního systému. Verze neumožňuje vzdálená připojení nebo vzdálenou správu. Tomu odpovídá i poměrně jednoduchá instalace a konfigurace.

LocalDB vznikl jako nástupce SQL Server Compact (SQL CE). Ta oproti LocalDB fungovala jen jako knihovna pod samotnou aplikací, LocalDB je spouštěna v rámci samostatného procesu. Dalším rozdílem je velikost na disku, SQL CE zabírá 4MB oproti 140MB LocalDB. Hlavním rozdílem je ale samotná funkcionalita. Např. SQL CE neumožňuje procedury, maximální velikost databáze jsou 4GB (oproti tomu u LocalDB lze využít až 10GB). SQL CE pro ukládání databáze využívá souborový format `.sdf`.

Transact-SQL je dotazovacím jazykem společností Microsoft a Sybase. Jedná se o rozšíření dotazovacího jazyka SQL pro interakci s relačními databázemi. [13]

Kapitola 3

Možnosti získávání dat za pomoci API

API, neboli Application Programming Interface, je balík knihoven, funkcí, protokolů a tříd, které lze použít v rámci programování aplikace. Rozlišují se API na úrovni operačních systému, na úrovni programů či webů. [1] Pro nás nejzajímavější je API na úrovni webových služeb. Mezi známé API patří také technologie DirectX či OpenGL, obě pro co nejlepší využití grafického jádra při vykreslování grafických prvků nebo Berkeley sockets disponující API pro meziprocesorovou komunikaci.

Webové API funguje jako webová služba, obvykle dostupná přes webovou adresu. Původní technologií pro výměnu dat přes protokol HTTP byl Simple Object Access Protocol (SOAP). Komunikace probíhá stylem dotaz-odpověď. Pro přenos dat se využívá protokol XML, přes formát XML probíhala komunikace oběma směry. Jedná se o technologii zaměřenou na služby. Nástupcem SOAP je technologie orientovaná na zdroje Representational State Transfer (REST). REST je dnes využíván velkým množstvím API mnoha webových služeb a definuje standardy, jak se přistupuje k datům. Ty jsou přístupné přes různé skripty či vnořené datové struktury a kolekce. Každý ze zdrojů musí byt adresovatelný pomocí URI, k jejich manipulaci se používá menší množství metod. Zdroje, přestože jsou identifikované přes jedno URI, mohou mít více výstupních formátů. Mezi nejvíce používané patří formáty pro výměnu dat Extensible Markup Language (XML) nebo JavaScript Object Notation (JSON). Webové stránky pro své potřeby mohou také požadovat formát HTML. Použít lze ale v podstatě jakýkoliv formát spadající pod internetová média. Komunikace probíhá bez stavů, vše je tedy uloženo v komunikaci, server nemusí ukládat žádné data. [18]

API pro identifikaci klientské aplikace často používají tzv. Client ID. Jedná se o identifikační číslo přidělené službou API pro využívání daného API. To se spolu s dalšími povinnými částmi dotazu posílá na webovou službu.

Po prozkoumání možností ohledně API filmových a seriálových databází jsem se rozhodl pro využití 2 API, konkrétně služeb Trakt a The Movie Database (dále jen TMDb). K těmto službám jsem se rozhodl z důvodu rozsahu jejich databáze a možnostem, které tyto API nabízí. Výčet informací, které lze z jednotlivých služeb není celý a jsou uvedeny především ty, které jsou podstatné v rámci aplikace.

3.1 Trakt API

API služby Trakt je využito jako hlavní API aplikace. API služby běží na adrese https://api.trakt.tv . Mezi povinné hlavičky pro komunikaci patří používaná verze API, aktuálně je Trakt API verze 2. Další povinnou hlavičkou je Client ID. Posledním části je určení typu internetového média. Trakt API umožňuje pouze formát JSON, resp. `appliacation/JSON`. [12]

Trakt API poskytuje poměrně rozsáhlé možnosti získání informací z jejich serveru. Služba umožňuje vyhledávat seriály, filmy i osoby podlé názvu, resp. jména. Pro seriály umí získat jejich přehled, získat sezóny a epizody včetně informací, statistiky sledovanosti na této službě nebo oblíbené seriály. Podobné informace a data API nabízí také pro filmy.

Služba také poskytuje informace o osobách, kromě samotného vyhledávání umožňuje získaní informací o osobě nebo seznam filmů a seriálů ve kterých daná osoba působila. Stejně tak umožňuje získat obsazení filmů a seriálů.

Služba umí také získat seznam žánrů, které jsou v rámci služby využity. Dále také seznam např. seriálů, resp. jejich epizod, nebo filmů, které vychází v daném časovém úseku.

3.2 The Movie Database API

Jako druhé API jsem se rozhodl použít API služby TMDb. Tuto službu jsem více prozkoumal až v průběhu implementace jako alternativu k Traktu. Jako jeho hlavní výhodu považuji především jazykové lokalizace, včetně českého jazyka. API běží na adrese http://api.themoviedb.org/3/ . Client ID se zadává v tomto případě přímo do URL. Stejně, jako je tomu u služby Trakt, je i odpověď ze serveru TMDb ve formátu JSON. [10]

Stejně jako API od Traktu, API služby TMDb poskytuje velké množství metod. Oproti službě Trakt ale umožňuje získat více informací. Jako hlavní výhodu lze považovat možnost získat jazykové lokace informací. Jazyk je definován v samotném dotazu za pomoci `ISO-639-1`[1] kódu jazyka. Tento krok lze ještě rozšířit pro jazyky, kterými se mluví ve více státech (např. španělština či portugalština v Jižní Americe), přidáním `ISO-3166-1`[2] kódu za kód jazyku. Výsledný dotaz potom končí `language=pt-BR` pro brazilskou portugalštinu. Pro český jazyk je kód `cs`, případně `cs-CZ`.

Služba umí získat detaily o seriálech, jejich hercích a členech štábu, a také podobné nebo populární seriály. Navíc umí získat trailery či obrázky k seriálu, certifikaci seriálu nebo alternativní názvy seriálu. Oproti API služby Trakt ale umí získat více informací k sezónám a seriálům, poskytuje dotazy pro detaily, herce a členy štábu. Informace získatelné pro filmy se opět dost shodují se seriálovými. Navíc lze ale získat premiéry podle státu včetně data vydání na DVD apod.

K osobám umí služba získat jejich detaily či účast na filmech a seriálech. Také umí získat seznam rolí členů štábu. Ze služby je možné také získat seznam certifikací, žánru nebo časových zón. Dále také informace o filmových společnostech a filmech, na kterých se podílela, stejně tak pro televizní společnosti.

Služba samozřejmě umožňuje vyhledávání různých položek. V rámci TMDb lze zvolit z více způsobů. První způsob je přes metody `Search`. Ty umožňují vyhledávání filmu, seriálů, osob a filmových společností pomocí vyhledávaného spojení, např název filmu. Druhou variantou vyhledávání je `Discover`. Ta umožňuje vyhledávání filmů a seriálů podle mnoha

[1] Dostupný na http://www.loc.gov/standards/iso639-2/php/English_list.php
[2] Dostupný na https://www.iso.org/obp/ui/

filtrů, které služba nabízí. Umožňuje vyhledávat filmy, kde hraje konkretní herce či se podílí člen štábu, podlé žánrů, roku vydání, filmové společnosti, a mnoho dalších. U seriálů sice není tolik možností filtrování, přesto lze vyhledávat podle vysílacího dne, žánrů nebo třeba produkční televize. Poslední varianta vyhledávání `Find` umožňuje vyhledávání v rámci TMDb pomocí identifikačního čísla jiné služby, např. TBDb nebo IMDb.

3.3 Další možnosti API

Mezi další služby, které by šly na aplikaci napojit lze zařadit například služba TVmaze nebo Tvdb jako zdroje informací o seriálech. Pro informace o filmové tvorbě lze použít také OMDb. V tomto případě se jedná o službu založenou na uživatelském zapojení, data k filmům zadávají uživatelé, a provoz této služby je závislý právě na uživatelských příspěvcích. Jako další zdroj by šlo použít také API Wikipedie, ze kterého by šlo získat informace např. také o filmech, soundtracky a další věci podle potřeby. Další služba, nad kterou by šlo potenciálně uvažovat by byla služba Tunefind poskytující data k hudbě filmů a seriálů, ovšem z této služby nelze data dlouhodobě ukládat.

K dalším možnostem pro získání informací o hudbě lze zařadit Discogs API, poskytující databázi hudebních autorů či značek (Labels) vydávající hudbu. Další možností je napojení na Last.fm API poskytující získání informací o autorech, albech, či samotných písních. Služba umožňuje také písně vyhledávat. Tyto služby lze zkombinovat se službou Tunefind pro ukládání a přiřazování hudby k jednotlivým filmům či seriálům.

Pro získávání informací ohledně knížek lze využít Google Books API. To umožňuje vyhledávat knihy a následně získat informace. Další alternativou je třeba Open Library Books API nebo API služby Amazon.

Přestože je v aplikaci implementováno identifikační číslo pro službu TvRage, tato služba již delší dobu není funkční, její identifikační čísla jsou ale stále funkční, a lze podle nich hledat na ostatních službách.

Ze služeb, které implementovat nelze, zmíním především české servery Česko-slovenská filmová databáze nebo Filmová databáze online. Ze zahraničních se jedná především o známou Internet Movie Database. Všechny zmíněné servery neposkytují vlastní API a jejich podmínky neumožňují legálně využít data v aplikacích třetích stran.

3.4 Formáty dat používané v komunikaci API

Přestože není stanoven oficiální formát dat, používají se v praxi především Extensible Markup Language (XML) nebo JavaScript Object Notation (JSON). Oba se používají pro serializaci dat. Serializace je proces, kdy se objekt převádí do sériové podoby. Tato podoba může být následně uložena jako soubor nebo přenášena mezi počítači. [24] Oba formáty umožňují bezproblémovou serializaci kolekcí (polí) a struktur (objektů).

3.4.1 XML

XML je značkovací jazyk pro ukládání a přenos dat nezávislý na systému. Samotný jazyk ovšem nevykonává žádnou činnost, pouze data ukládá. Přestože je zde značná podobnost s jazykem HTML, na rozdíl od něj nemá předdefinované značky. Ty si určuje autor dokumentu sám. Samotný jazyk je snadno rozšířitelný, přidání nové značky by nemělo udělat aplikaci využívající XML nefunkční. Jeho standart spravuje konsorcium W3C, které má na starosti mnoho webových standardů, mezi nimi například HTML či SOAP. Výsledný soubor

dokumentu lze snadno prezentovat nebo převádět do jiných formátů. Jazyk není svázán se softwarem konkétní společnosti, proto je vhodný pro přenos informací. [17]

Aby šlo o validní XML soubor, musí soubor obsahovat právě jeden kořenový element. Název tohoto elementu může být libovolný, důležité je pouze počet. Každý element musí mít svoji ukončovací značku . Ukončení je možno provést i pomocí lomítka v úvodní značce. Jednotlivé elementy se nesmějí překrývat, tzn. počáteční i koncová značka musí být v rámci jednoho elementu. Všechny atributy elementu musí být řádně uzavřeny mezi jednoduché nebo dvojité uvozovky (vždy stejným typem). Jazyk XML je citlivý na velikost a v názvech elementů umožňuje použití libovolného kódování.

```
<?xml version="1.0" encoding="UTF-8" ?>
<student>
        <jmeno>Lukas</jmeno>
        <prijmeni>Velecky</prijmeni>
        <vek>23</vek>
        <bakalarskaPrace>
                <jmeno>Databaze filmu a serialu</jmeno>
                <klicoveSlova>C#</klicoveSlova>
                <klicoveSlova>API</klicoveSlova>
                <klicoveSlova>.NET</klicoveSlova>
                <klicoveSlova>MSSQLServer</klicoveSlova>
        </bakalarskaPrace>
</student>
```

Algoritmus 3.1: Ukázka XML souboru

3.4.2 JSON

JSON je formátem pro přenos a serializaci dat vycházející z jazyka JavaScript. Stejně jako XML je nezávislý na systému. Data v rámci formátu jsou ukládána v kolekcích nebo strukturách ve formě dvojice jméno - hodnota. Datové typy, které lze v rámci JSON využít jsou řetězec, číslo, objekt, pole, logický datový typ boolean nebo hodnota null. Stejně jako XML, JSON neřeší kódování textu. Řetězce musí být ohraničeny v uvozovkách, jednoduchých či dvojitých, pár ale musí vždy souhlasit. Objekt je neseřazenou dvojicí jméno - hodnota a je ohraničen složenými závorkami. Pole je neseřazenou kolekcí hodnot uzavřenou v hranatých závorkách. Tyto hodnoty jsou od sebe odděleny čárkami. [7]

```
{
  "student": {
    "jmeno": "Lukas",
    "prijmeni": "Velecky",
    "vek": "23",
    "bakalarskaPrace": {
      "jmeno": "Databaze filmu a serialu",
      "klicoveSlova": [
        "C#",
        "API",
        ".NET",
        "MSSQLServer"
      ]
    }
  }
}
```

Algoritmus 3.2: Ukázka JSON souboru, obsah dat se shoduje s ukázkou XML

Kapitola 4

Návrh aplikace

Pří návrhu aplikace jsem uvažoval všechny možné aspekty sledování seriálů a filmů. Pro fanouška seriálů, resp. filmů, je často důležitou informací, které seriály mu aktuálně vychází, či které teprve vycházet začnou nebo chce teprve sledovat. Podobná situace se opakuje pravidelně i u filmů či knížek, kdy uživatel chce ukládat informace a mít informace o datech vydání. Navíc se filmy a seriály spolu s knihami různě překrývají. Filmy se točí podle knižní předlohy, náměty seriálů vznikají na motivy komiksů a knihy vznikají jako pokračování či alternativní příběhy seriálů či filmů. Jejich provázání je tedy obrovské. Ačkoliv na filmy a knihy existuje mnoho desktopových aplikaíe, u seriálů je situace již horší.

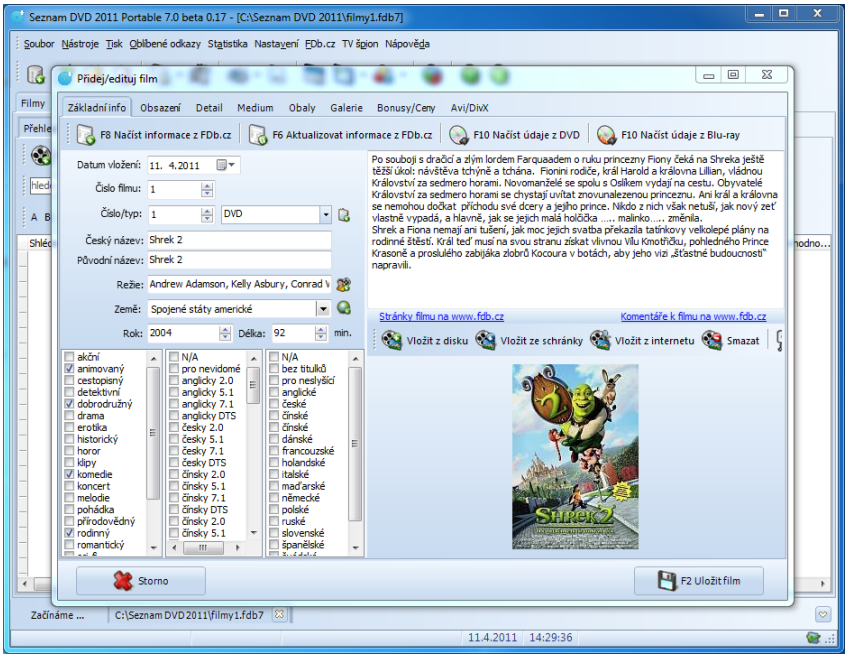
4.1 Webové portály

U online aplikací je situace lepší snad ve všech směrech. Pro filmy lze uvést české stránky Česko-Slovenska filmová databáze (ČSFD.cz) nebo Filmová databáze (FDb.cz), ze zahraničních třeba Internet Movie Database (IMDb.com). Ta umožňuje práci s epizodami seriálů. Sice na stránce existuje tzv. Watch List, ale pořád se nejedná o požadavky uživatelské situace sledování seriálů. Například služba Trakt již umožňuje označovat episody seriálů a samotné filmy jako zhlédnuté.

Český portál Edna (edna.cz) v průběhu vývoje aplikace představil funkci *Bedna*. [25] Ten nahradil předchozí funkci oblíbené seriály. Ta umožňuje třídit seriály na 3 skupiny: „Rozkoukané“, „Dokoukané“ a „Chci vidět“. Tato funkce se nejblíže přibližuje tomu, jak bych chtěl aplikaci vytvořit. Přestože je portál primárně zaměřen na seriály, lze na něm dohledat informace i o filmech souvisejících se seriálovým světem. Server Edna lze tedy brát s rezervou jako předlohu pro tuto aplikaci.

4.2 Offline aplikace

Oproti webovým portálům je množství desktopových aplikací omezenější. Pro portál Filmová databáze vznikla aplikace Seznam DVD. Ten umožňuje přehlednou správu filmů. Aplikace je ale především zaměřena na fyzická média. Řeší půjčování a správu těchto médií. Mezi její další funkce patří také export do různých formátů (HTML, Excel, a další). Umožňuje tisk obalů pro DVD či Blu-ray nebo správu kontaktů. Funkčností se tedy jedná o odlišné požadavky oproti aplikaci vyvíjené v rámci bakalářské práce. Přesto aplikace umožňuje načítat informace z webového portálu.



Obrázek 4.1: Editační okno filmu v programu Seznam DVD [9]

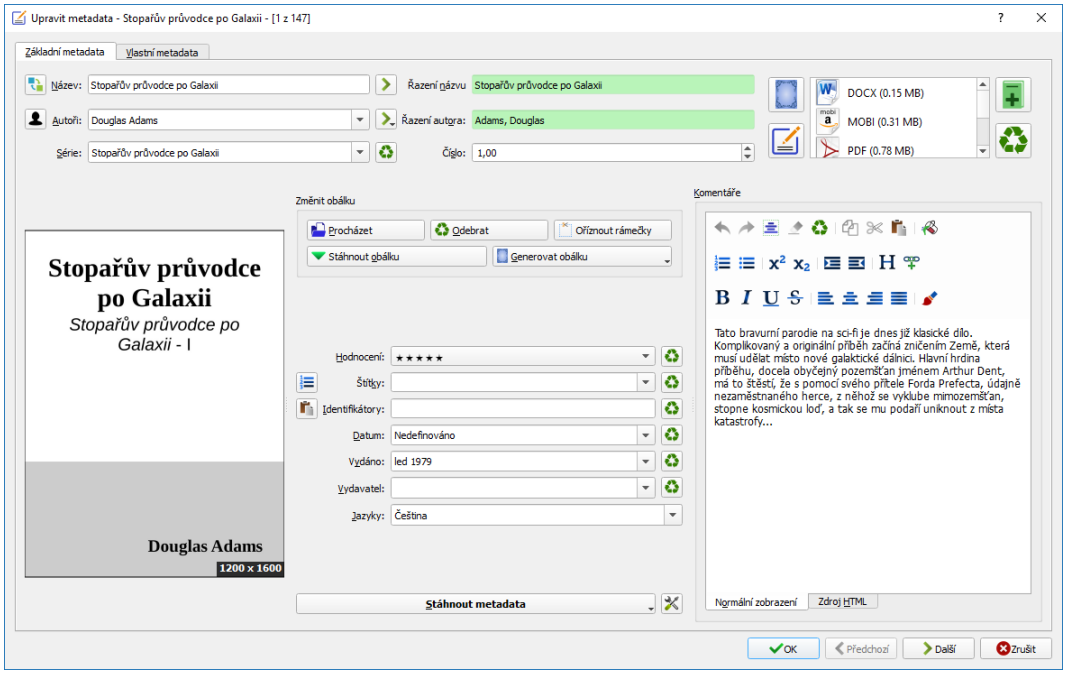
Na internetu lze najít mnoho dalších aplikací, které umožňují správu filmů. Velká část je, stejně jako aplikace Seznam DVD, zaměřena spíše na fyzická média a často pouze na filmy. Mezi jednu z aplikací lze zařadit i aplikaci MediaMonkey. Ta slouží především k přehrávání medií, ale také k jejich správě. V aplikaci lze přehrávat a spravovat informace o audio i videu. Aplikace je ale spíše uzpůsobena ke správě hudby, i vlastnosti videí, které lze editovat a spravovat v rámci databáze více odpovídají hudebním klipům než filmům všeobecně.

Mnoho aplikací využívající některé z API často umožňuje správu filmů, ale ve většině případů jen filmů a často jen za podpory API, nejedná se tedy o offline aplikace v pravém smyslu, jako spíš o desktopové zobrazení informací z API. Tyto aplikace často pracují jen nad API, zobrazují co aktuálně uživatel sleduje, co má v plánu sledovat nebo co by mohl sledovat. Velké množství takových aplikací je vytvořeno především pro mobilní operační systémy nebo pro systémy fungující jako mediální centra. Kromě klasických aplikací vznikají i různé pluginy do multimediálních přehrávačů či k online službám typu Netflix či HBO GO.

Poslední aplikace, kterou bych chtěl zmínit je Calibre. Ta umožňuje správu knížek uživatele. Aplikace je především určena pro správu elektronických verzí. Kromě ukládání informací umožňuje konverzi mezi formáty elektronických knížek nebo tvorbu knižních sérií. Kromě toho také umožňuje vyhledávat informace o knížkách z externích služeb, jako je Amazon nebo Google. Přestože vyvíjená aplikace není zaměřená na knihy a správu souborů k nim, Calibre sloužila jako inspirace pro některé z vyvíjených funkcí.

4.3 Specifikace požadavků

Při návrhu aplikace je bráno především snadnost jeho rozšíření. Jednotlivé atributy objektů by neměly být extra závislé na jiných atributech. Jako příklad lze uvést sledování seriálů. V rámci návrhu není nutné vědět, kterou epizodu uživatel viděl jako poslední. Stačí znát,



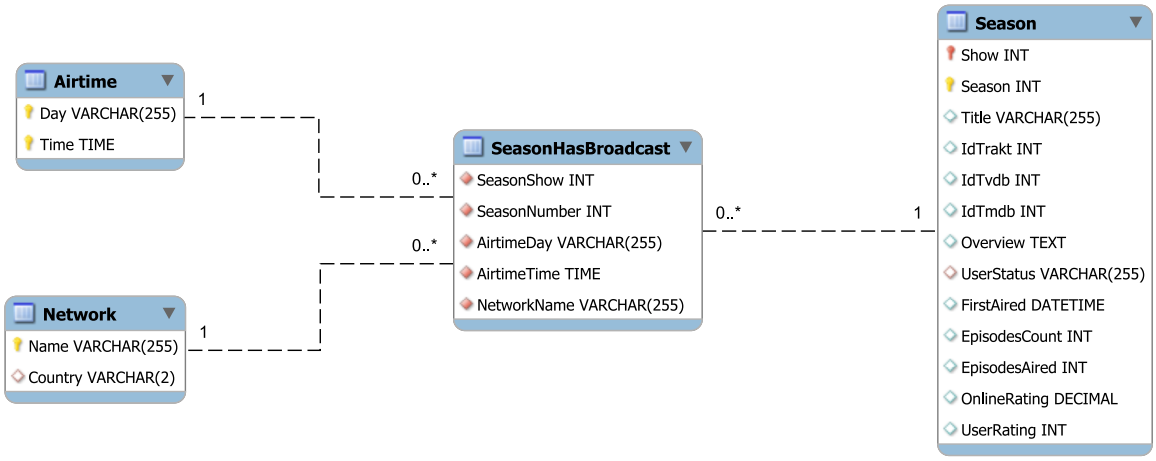
Obrázek 4.2: Editační okno knihy v programu Calibre

které uživatel již viděl, resp. zatím neviděl, a díky tomu získat epizodu, kterou má uživatel vidět.

V rámci seriálů je samozřejmě důležitý název seriálu. Zatím uvažuji pouze jeden název, v rámci návrhu neberu v potaz rozlišení lokalizovaného či originálního jména. Kromě toho uživatel chce vědět o čem je daný seriál, jestli se seriál stále vysílá a jaká je jeho přibližná délka. Podstatný je také počet epizod a počet již odvysílaných epizod. Řešení počtu epizod, resp. těch odvysílaných, se nabízí dvě. První řešení spočívá v jednoduchém ukládání tohoto počtu v rámci databáze, druhé řešení tyto čísla vypočítá z počtu epizod seriálu uložených v databázi. V případě odvysílaných epizod se navíc jedná o nutnost uvádění data odvysílání, a poté vůči aktuálnímu datu lze zjistit, jestli epizoda jež měla premiéru. Obě řešení jsou silně závislá na editaci těchto údajů a obě lze aktualizovat jak uživatelem, tak přes API. Podobná situace se vyskytuje i při sezónách seriálů, kdy sezóna obsahuje počet epizod, v rámci jednotnosti je u obou variant (sezóna i seriál) využito řešení s ukládáním hodnot v rámci databáze, které vyžaduje méně interakce uživatele.

Další informace o seriálech, které chce uživatel vědět, je pod jakou televizí seriál běží a v které dny a čase seriál vychází. Problém v těchto hodnotách je ten, že se v průběhu vysílaní může čas vysílaní změnit. Kromě změny času vysílaní se může změnit také televize, pod kterou se seriál běží. Většina těchto změn se děje především mezi sezónami, občas změna nastává i v průběhu samotné sezóny. Původní návrh obsahoval propojení seriálu s televizním kanálem a sezóny s vysílacím časem. Po zvážení možností došlo k přesunutí propojení televizního kanálu na úroveň sezóny a propojení s vysílacím časem. Toto spojení je ideální i z toho hlediska, že vysílací čas je často důležitý jen v rámci dané televize, informace ostatních televizí jsou důležité maximálně z hlediska konkurence.

Důležitou informací pro seriály a filmy jsou také účinkující herci a členové filmového štábu. Pro filmy je tato skutečnost vcelku jednoduchá, herec hraje postavu ve filmu nebo



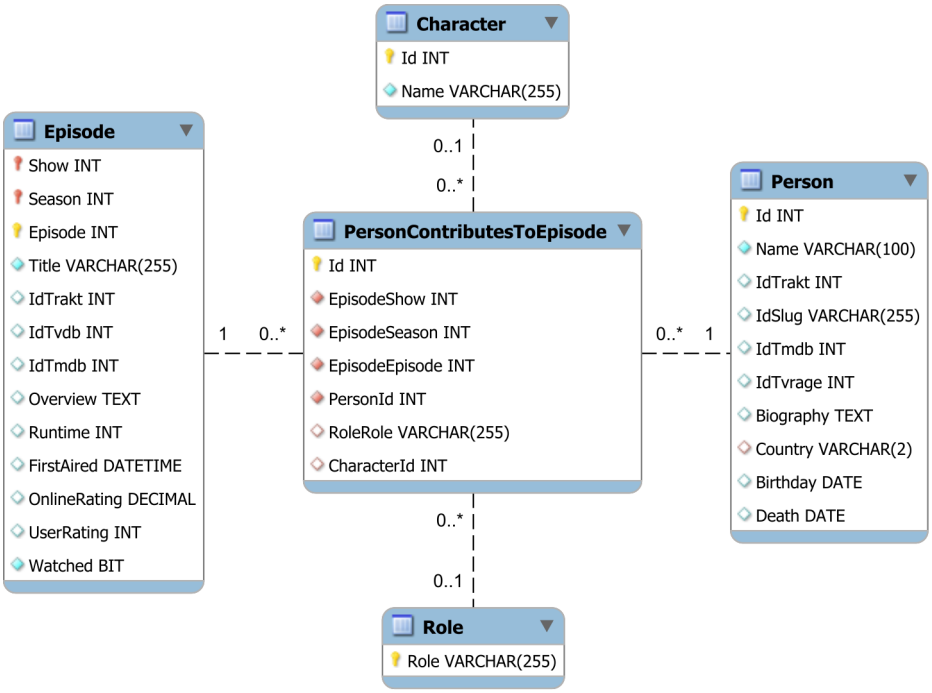
Obrázek 4.3: Grafické znázornění návrhu pro vysílání sezóny

člen štábu má nějakou funkci pro film. Přestože u seriálů jsou požadované informace podobné, od filmů se odlišují účastí především filmového štábu, ale také postav v rámci jednotlivých dílů či sezón. Postavy v seriálech lze rozlišit na hlavní, sezónní nebo epizodní. Hlavní postavy se objevují ve většině epizod celého seriálu nebo v průběhu více sezón, tyto postavy jsou důležité v kontextu celého seriálu. Další část postav se objevuje jen v rámci sezóny, tyto postavy se ale občas objevují i mimo konkrétní sezónu. Tyto postavy mají často významný vliv na dějové linky konkrétní sezóny, a můžou více či méně ovlivnit i dějové linky následujících sezón. Poslední skupinu tvoří herci, resp. jejich postavy, které se objevují v seriálu spíše jen na jednu či menší počet epizod. Tyto postavy spíše ovlivňují jen konkrétní epizodu, případně epizody. Občas se může jednat o *hostující hvězdu* (Guest Star). Mezi toto účinkování lze zařadit i herce z předchozích kategorií, které již v seriálu neúčinkují, přesto se natáčení pár epizod zúčastnili. K těmto rolím se řadí i tzv. *Cameo role*. Cameo je krátké (často v rámci jednoho střihu) účinkování známé osobnosti. Tyto role nejsou často předem ohlášeny a nebývají uváděny ani v titulcích.

Podobný problém jako s herci nastává často také u filmového štábu. U mnoha seriálů se stává, že režii epizody má na starosti jiný režisér, někdy se jich na sezóně podílí více, jinde je to co epizoda to režisér. Tato situace se týká nejen režisérů, ale také scénáristů a dalších členů filmového štábu. Z tohoto důvodu došlo k umožnění napojení osob jak na seriál, tak na sezóny i epizody.

Pro filmy lze také kromě klasického přehledu získat tzv. slogan (Tagline). Jedná se o krátký text, který by měl být snadno zapamatovatelný, sloužící k propagaci daného média. Často se vyskytuje na obalech filmů a pro filmové série tento slogan často zůstává stejný, či podobný. Příklady takových sloganů jsou *"Před dávnými časy, v předaleké galaxii..."* (A long time ago in a galaxy far, far away) pro Star Wars: Epizoda IV nebo pro první díl trilogie Pána prstenů *Jeden prsten vládne všem* (One ring to rule them all).

Stejně jako na filmech a seriálech, tak i na knihách se podílejí určití lidé, kteří se na ní podíleli různými způsoby, například autor knihy či překladatel. Oproti filmům a seriálů zde nedochází k propojení osoby s konkretní postavou. Místo toho jsou osoby řešeny jako samostatná vlastnost. Díky tomuto lze propojovat filmové postavy s těmi knižními při knižních předlohách a podobně.



Obrázek 4.4: Grafické znázornění návrhu pro podíl osob

Situace pro písně je mírně odlišná. Osoba se na hudbě může podílet přímo jako autor, nebo např. jako člen skupiny. Proto došlo k vytvoření autora. Autor seskupuje více lidí do jednoho útvaru s názvem, např. do hudební skupiny. Tento útvar tedy reprezentuje autora písní. Kromě uskupení lidi se může jednat i o jednu osobu. Píseň může mít vždy jen jednoho autora. V rámci návrhu nelze přímo navrhnout album písní, to lze až přes *vztah*. Původní návrh pro písně obsahoval přímo album písní. Ten byl následně zjednodušen na aktuální bez přímého návrhu alba. K zjednodušení došlo z důvodu až velké nepřehlednosti.

Nejen pro vytvoření alba vznikl *vztah*. Jedná se o požadavek na propojení seriálů či filmů. Mnoho filmů, seriálů a knih je v dnešní době nějakým způsobem propojeno. Komiksy fungují jako předloha pro mnoho filmů či seriálů, tyto tvorby se následně spojují do velkých filmových světů. Knihy jsou někdy založené na filmových či seriálových postavách. A takových příkladů lze najít mnohem více.

Kapitola 5

Implementace

Aplikace je vytvořena v rámci jednoho *řešení* (Solution). Jednotlivé části jsou rozděleny na celky - projekty. Hlavní část aplikace tvoří projekty `Model`, `ViewModel` a `Desktop`. Tyto 3 projekty jsou části návrhového vzoru MVVM, kdy projekt `Desktop` zastupuje část vzoru *View*. Projekt `Repository` má na starosti práci s daty v databázi. Projekty `TraktAPI` a `TmdbAPI` obsahují moduly pro získávání dat z příslušných API. Případná data typu výčtový typ (`enum`) nebo slovník (`dictionary`) jsou uložena nebo je lze získat pomocí projektu `Data`. Návrh databáze je vytvořen v rámci projektu `DatabaseModel`.

Výstupem většiny těchto projektů je knihovna DLL, výstupem projektu `Desktop` je spustitelný EXE soubor. SQL skript je výstupem projektu `DatabaseModel`.

5.1 Databáze

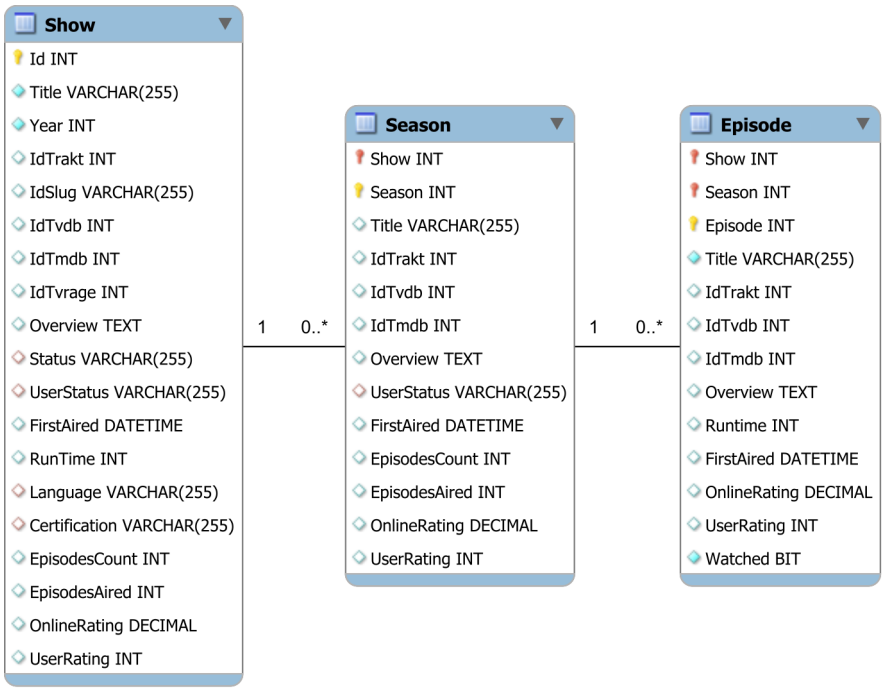
Databáze byla navrhnuta v jazyce *Transact SQL*. Každá z tabulek pro jednotlivé entity je vytvořena v samostatném SQL skriptu. Pro identifikační čísla je použita vlastnost `IDENTITY` nad datovým typem `INTEGER`. Primární klíče s názvem `PKjménotabulky` jsou vytvořené v rámci samostatných skriptů, stejně tak cizí klíče s názvem `FKjménospoje`. Tabulky, primární a cizí klíče pro vztahy M:N jsou vždy umístěny ve stejné složce se stejným pojmenovaným uvedeným výše. V rámci některých tabulek byly vytvořeny indexy pro některé z jejich sloupců, které nejsou součástí primárních klíčů. V databázi bylo vytvořeno také několik procedur, především pro mazání přes SQL.

5.1.1 Migrace databáze

Převod databáze do tříd jazyka *C#* proběhl metodou *Code First Migration*. [4] Je to jedna ze dvou variant migrace, kterou Entity Framework nabízí. Druhou variantou je *Database First*. Přestože se zdá, že situaci v této aplikaci je bližší právě druhá zmíněná metoda, není tomu tak. Výsledkem této metody je soubor .edmx. Jelikož ale chci v rámci aplikace používat klasické *C#* třídy, použil jsem první zmíněnou metodu nad již existující databází.

Výsledkem této metody jsou nejen samotné třídy, které souhlasí s návrhem v rámci databáze, ale také `DbContext`. Ten mapuje třídy na tabulky v databázi a obsahuje seznamy entit.

V rámci jednotlivých tříd mají některé z jejich vlastností anotace. Tyto anotace odpovídají jejich alternativám (omezením, datovým typům) v rámci databáze. Např. vlastnost s anotací `[required]` odpovídá SQL omezení `NOT NULL`.



Obrázek 5.1: Grafické znázornění entit pro seriál, sezónu a epizodu

5.2 Projekt Model

Jako základní entity jsem si zvolil entity týkající se seriálů - `Show`, `Season` a `Episode`, dále entitu pro filmy `Movie` a entitu zastupující osobu `Person`. K těmto hlavním entitám jsem se rozhodl do návrhu přidat i entity pro písně `Song` a knihy `Book`. Tyto entity jsem hromadně nazval jako *Média*.

Kromě těchto hlavních entit se v návrhu nachází i několik dalších skupin entit, které doplňují hlavní entity o další atributy. Nejpodstatnější skupinou je skupina, kterou lze nazvat jako *vlastnosti*. Tato skupina zastřešuje entity, které rozšiřují význam hlavních entit a můžou sloužit k jejich vyhledávání či třídění. Mezi tyto entity patří např. žánr `Genre` nebo jazyk `Language`. Další skupinou jsou entity znázorňující podíl osoby a jeho roli na některém z médií. Poslední skupinou entit jsou entity *vztahové*. Tyto znázorňují vztahy mezi jednotlivými médii.

Všechny zmíněné entity implementují základní třídu `BaseClass`. Přes tu jednotlivé třídy implementují rozhraní `INotifyPropertyChanged`.

5.2.1 Popis entit

Seriály jsou zastoupeny 3 entitami `Show`, `Season` a `Episode`, znázorňující postupně samotný seriál, jeho sezóny a epizody.

Seriál je identifikovatelný identifikačním číslem `Id`. Jako alternativu je teoreticky možné použít kombinaci Název `Title` a Rok `Year`. U této kombinace ovšem není jistota, jestli nemůže docházet k duplicitním seriálům. Jednotlivé sezóny či epizody už jsou ovšem vždy specifické pro konkrétní seriál, proto jako primární klíč může posloužit identifikační číslo seriálu s pořadím sezóny, resp. pořadím epizody.

Tyto entity také obsahují další identifikační atributy. Tyto ovšem neslouží k identifikaci v rámci aplikace, ale pro účely API, např. služeb Trakt nebo The Movie Database (TMDb).

Atributy seriálu a sezóny jsou napojené na některé z *vlastností*. V případě seriálu je to na entity Země a Žánr v rámci M:N vztahů. V případě M:1 vztahů se jedná o Uživatelský status, Status, Jazyk a Certifikaci. U sezóny se jedná o M:1 napojení na Uživatelský status a na entitu vysílání `Broadcast`. Vysílaní je reprezentace ternárního vztahu spojujícího entity seriál, televize a vysílací čas.

Entitu reprezentující film jsem navrhl obdobně jako entitu pro seriál. Stejně jako seriál i film má ze stejných důvodů jako primární klíč identifikační číslo místo kombinace Název a Rok. I ostatní atributy a napojení na *vlastnosti*, vyjma entity Status. Oproti seriálu obsahuje i méně identifikačních atributů.

Jednotlivé osoby jsou identifikovány identifikačním číslem. Pro účely a vyhledávání v API i tato entita obsahuje identifikaci pro tyto služby. Napojení na zemi původu je v tomto případě pouze M:1, a jedná se o jedinou *vlastnost*, na kterou je osoba přímo napojena.

Vlastnosti, které jsou pro účely práce s osobami použité, jsou entity pro roli `Role` a postavu `Character`. První zmíněná entita znázorňuje jednotlivé role filmového štábu, např. režiséra nebo scenáristu.

Propojení osob s médii se děje přes tabulky `PersonContributesToX`. Toto propojení zastupuje, jak se osoba na daném *médiu* podílela. V případě herců jakou postavu ve filmu hrál. Vždy se jedná o tabulku spojující osobu, dané medium a roli, v případě filmů a seriálů se na tabulku váže ještě charakter. Původní návrh těchto tabulek neobsahoval identifikační číslo, ale identifikace probíhala pomocí identifikace média, osoby a role. Tento návrh ovšem neumožňoval například situaci, kdy jeden herec hraje více rolí v jednom filmu. Proto došlo k přidání identifikačního čísla. Rozpoznání herce od ostatních členů filmového štábu probíhá pomocí hodnot role a charakter. Herec má nastavenou roli na hodnotu `null`, zatímco ostatní mají hodnotu `null` v poli charakter. V situaci, kdy osoba působí jako herec a člen štábu, jsou vytvořeny dvě položky v databázi.

Podobnost mezi entitami médií je znát i u knih. Opět je kniha identifikovatelná pomocí identifikačního čísla. Oproti předchozím ale již neobsahuje identifikaci v rámci různých API, protože aktuálně aplikace nemá žádné plánované napojení na API pro knihy. Je opět napojená na jazyk a uživatelský status, a přes M:N vztah i na žánry. Kromě napojení na osoby je kniha spojena vztahem M:N i s postavami, které se v knize vyskytují.

Pro audio jsem se nakonec rozhodl využít 2 entit pro píseň a autora. Autor je uskupení lidí, kteří na písní pracovali, např. hudební skupina. Stejně jako u knih, ani jedna z entit není napojená na API, proto obsahují pouze ID pro identifikaci v rámci aplikace.

Entita pro titulky je opět identifikovatelná identifikačním číslem. Kromě něj také obsahuje cestu k souboru a jazyk titulků. Titulky obsahují odkaz na film nebo konkrétní epizodu seriálu. Nemůže tedy nastat situace, kdy titulky patří jak k filmu tak k seriálu, toto původně ošetřeno i přímo v databázi pomocí triggeru, bohužel Entity Framework neumí s triggerama fungovat, proto toto omezení lze kontrolovat jen na úrovni databáze.

Úlohou vztahových entit je spojit *média* do větších celků v závislosti na jejich vztahu. Jako příklad lze uvést spojení více filmů do filmové série nebo spojit knihu a film (případně i více filmů), kdy kniha slouží jako předloha filmu nebo naopak. Každý z těchto vztahů je identifikován identifikačním číslem. Druh tohoto vztahu je dán *vlastností* Typ vztahu `RelationshipType`. Kromě *médií* může vztah spojovat i jiný vztah a vytvářet pak vztahy, kdy kniha je předlohou filmové trilogie.

5.3 Projekt ViewModel

Projekt ViewModel obsahuje 2 části - ViewModely a Příkazy (Commands). Každé z těchto části se budu věnovat samostatně.

5.3.1 ViewModely

Úkolem ViewModelu je zveřejnit data z modelu pro View. V rámci ViewModelů je potřeba rozlišit ViewModely pro jednotlivé položky a ViewModely pro seznam položek.

Základní třídou je `BaseViewModel`. Ta implementuje rozhraní `INotifyPropertyChanged` potřebné při komunikaci mezi ViewModelem a samotným View. Dále definuje atributy pro `Service` pro komunikaci s databází a související metody pro ukládání a načítání dat. Také definuje odkaz na rodičovský ViewModel.

Třída `BaseCollectionViewModel` je základní třídou pro ViewModely se seznamem položek. Tato třída implementuje třídu `BaseViewModel` a oproti ní navíc definuje vlastnosti `NewItem`, která slouží jako uchování informace pro objekt přidávaný do databáze, `SelectedItem` sloužící pro vybrání konkretního prvku seznamu a především pole `Items`, které obsahuje všechny položky v rámci kolekce.

Všechny použité ViewModely dědí z jedné z výše zmíněných tříd. Přes třídu `BaseViewModel` také všechny ViewModely dědí rozhraní `IViewModel`. Toto rozhraní umožňuje snazší předávání rodičovského ViewModelu. Ten je používán jak v příkazech, tak ve ViewModelech pro přístup i k dalším ViewModelům, např. za účelem změny `SelectedItem`.

Základním ViewModelem je `MainViewModel`. Ten implementuje rozhraní `IViewModel` a `INotifyPropertyChanged`. Tento ViewModel zpřístupňuje ViewModely těm View, které jej využívají. Pro View jsou zde také další vlastnosti, a to konkrétně `CurrentXView`, kde X je konkretní entita. Tyto umožňují měnit View konkrétního okna na požadovaný pohled.

5.3.2 Příkazy

Příkaz vykonává určitou posloupnost příkazů na podnět uživatele (např. stisknutí tlačítka). Příkaz vždy pracuje nad ViewModelem (může pracovat i nad více ViewModely). V rámci návrhového vzoru MVVM třída pro příkaz implementuje rozhraní `ICommand`. Základní implementaci tohoto rozhraní zaštiťuje třída `BaseCommand`.

Metody třídy `BaseCommand`, které je potřeba v rámci rozhraní `ICommand` implementovat, jsou `CanExecute` a abstraktní `Execute`. Metoda `Execute` je brána jako akce příkazu. Metoda `CanExecute` vrací, jestli je možné daný příkaz spustit. Návratová hodnota této metody se projeví například povolením či zakázáním tlačítka, na který je daný příkaz vázán. Základní implementací této metody může být hodnota `true`.

V rámci samotných příkazů je důležitá především implementace právě metody `Execute`. Kromě této některé příkazy redefinují metodu `CanExecute`, aby za určitých podmínek (např. nevyplnění všech povinných polí) nešlo spustit příkaz.

5.4 Projekt Desktop

Projekt `Desktop` je prostředním prvkem návrhového vzoru MVVM. Jeho úkolem je vytvořit uživatelské prostředí, zobrazující data z Modelu přes ViewModel. Uživatelské rozhraní je vytvořeno za pomoci jazyka `Extensible Application Markup Language` (XAML) v knihovně tříd `Windows Presentation Foundation` (WPF).

Pro aplikaci je využito hlavní okno `MainWindow`. S tímto oknem je svázaný hlavní ViewModel `MainViewModel`. Přes toto okno lze zobrazit jednotlivé prvky uživatelského rozhraní (User Control). Tyto prvky jsou vytvořené např. pro *vlastnosti* nebo *média* a vytváří jednotlivé pohledy (View).

V případě převedení jednoduchých entit, především vlastností, postav a rolí, se jedná o jednoduché User Control zobrazující seznam těchto entit a ovládací prvky pro přidání, resp. odebrání položek této entity.

Při složitějších entitách, jako je třeba seriál nebo film, se jedná o uživatelské rozhraní složené z více User Control. Hlavním view pro tyto entity je User Control s názvem `EntityMainView`. Tyto view pouze obsahují odkaz na požadovanou User Control, která se zobrazí. Pro toto využívám prvek `Content Control`, který slouží k dynamickému měnění pohledu. Toto okno je svázané s vlastností `CurrentEntityView` z `MainViewModelu`. Podle ViewModelu, které je přiřazeno do vlastnosti, se zobrazuje konkretní View.

Pro zobrazení seznamu těchto entit se stejně jako u jednodušších entit využívá View se seznamem těchto entit. Tyto view kromě seznamu umožňují také zobrazit částečný náhled přes `EntityPartialDetailView`, přidat novou položku do seznamu nebo přejít na okno pro API dané entity.

`EntityFullDetailView` slouží k zobrazení a editaci konkretní entity, včetně mazaní konkretní entity. Umožňuje upravovat položky entity, přidávat či upravovat různé vlastnosti dané entity (např. změnit jazyk seriálu, přidat žánr k filmu nebo přidat autora knihy). V případě filmů a seriálů lze získat data či např. herce pomocí API.

Přidávání nových seriálů a filmů lze také uskutečnit přes `EntityTraktView`, které obsahuje ovládací prvky pro tyto účely.

Pro sezóny a epizody je vytvořen View, na který se lze dostat přes nadřazenou entitu (tedy seriál, resp sezónu) po kliknutí na požadovanou entitu. Jedná se vždy o podobné rozhraní umožňující editovat či mazat danou entitu, získat data přes API či přiřadit herce k entitě.

5.4.1 Prvky pro zobrazení

Pro grafické znázornění jednoduchých vlastností entit je použit především prvek `TextBlock`, pro jejich editaci prvek `TextBox`. Pro zobrazení seznamu entit je využito prvků `ListBox` nebo `ComboBox`. První je využito v případě především přehledného zobrazení (např. seznam filmů), druhé pro výběr jedné z položek seznamu (např. pro vybrání jazyka seriálu). Pro rozdělení uživatelského rozhraní je použit prvek `TabControl` spolu s jeho podprvkem `TabItem`. Tyto panely může uživatel libovolně přepínat pro zobrazení požadovaných informací.

5.4.2 Konvertory

Převod některých hodnot ViewModelu do uživatelského rozhraní probíhají přes konvertory. Ty slouží v situaci, kdy vlastnost je vázána na nějaký prvek nekompatibilním způsobem. V rámci aplikace jsou využity konvertory pro převod vlastnosti `PartialVisibility` některých ViewModelů na zobrazení částečného náhledu na entitu.

5.4.3 Soubor App.xaml a Resources

Soubor `App.xaml` je startovacím souborem projektu. Soubor určuje, které okno se bude spouštět jako úvodní. Dále tento soubor obsahuje globální zdroje (Resources). Ty obsahují například odkazy na ViewModely či modely pro účely designéru nebo odkazy na konvertory.

Dále definují tzv. datové šablony (DataTemplate). Stejně jako samotné okna, i data lze složit z více prvků. DataTemplate tedy zajišťují, jakým způsobem se budou data zobrazovat. Tohoto prostředku se využívá např. pro znázornění entity v seznamu entit. [15]

Mezi další zdroje, které lze v souboru najít jsou šablony pro styly (Style). Tyto lze následně přiřadit prvkům v UserControl a upravovat tak jejich vzhled. Tohoto se využívá pro opakované přiřazování vzhledu.

Resources lze také definovat jen v rámci okna. Potom je daná Resource použitelná jen v rámci daného okna. V aplikaci jsou použity zdroje především pro změnu obsahu okna, kdy DataTemplate obsahuje jako svůj prvek právě odkaz na požadovanou UserControl podle přiřazeného ViewModelu.

5.5 Projekt Repository

Projekt `Repository` slouží ke komunikaci mezi ViewModelem a databází. Třída `Repository`, ze které ostatní služby (Services) vychází, implementuje metody mimo jiné pro načítání, ukládání, mazání a aktualizaci dat. Načítání entity probíhá ve dvou krocích. Metoda `LoadData` nejprve načte data z databáze do lokální úrovně a z něj potom metodou `GetData` vrátí požadovaný seznam právě z této lokální úrovně.

Pro každou entitu, se kterou se pracuje v rámci databáze, je vytvořena služba (Service). V případě jednoduchých entit (většina *vlastností*, případně další) se jedná jen o vytvoření této služby za účelem snazšího vytváření těchto Service pro tyto entity.

Situace je ovšem odlišná pro entity, které obsahují odkaz (cizí klíč v rámci návrhu databáze) v rámci M:1 vztahu. Pro tyto Repository je potřeba aktualizovat nejen danou entitu, ale také odkazované entity. Důvodem tohoto je skutečnost, že odkazovaná vlastnost obsahuje seznam entit, které mají tuto vlastnost nastavenou a musí dojít k aktualizaci tohoto seznamu. Tato situace nenastává při M:N vztazích zprostředkovaných tabulkou.

Vztahy `PersonContributesToX`, `PersonIsPartOfAuthor` a `SeasonHasBroadcast` jsou znázorněním ternárních vztahů, proto je potřeba k nim přistupovat jako k ostatním entitám. Pro ně jsou vytvořeny jak samostatné modely a ViewModely, tak i samotné Repository.

Mazání z databáze je řešeno 2 způsoby. U služeb, které to umožňují se jedná o mazání právě přes Repository, u složitějších, kde docházelo ke konfliktům cizích klíčů, probíhá mazání za pomoci SQL procedur.

5.6 Projekt TraktAPI

Jedná se o projekt obsahující třídy a metody pro získávání dat z API služby Trakt. Základem je statická třída `TraktConnection`, která nastavuje Http klienta pro danou službu. V případě Trakt se jedná o nastavení verze API, ID klienta a typ média Content Type na hodnotu `application/json`. Pro získání dat, resp. zavolání samotného dotazu na API, je v této třídě také metoda `GetDataFromTrakt`, která se přes klienta snaží splnit požadavek. Všechny metody jsou implementovány i jako asynchronní, ty jsou označeny i v názvu pomocí slova `Async`.

```
public static async Task<HttpReturnType>
    GetDataFromTrakt(string requestUri)
{
    var returnValue = new HttpReturnType();
    try
    {
        var response = await _client.GetAsync(requestUri);
        returnValue.Code = response.StatusCode;
        returnValue.Data =
            await response.Content.ReadAsStringAsync();
    }
    catch(Exception ex)
    {
        returnValue.Code = HttpStatusCode.InternalServerError;
    }
    return returnValue;
}
```

Algoritmus 5.1: Implementace metody GetDataFromTrakt

Jednotlivé dotazy jsou roztříděny do modulů příslušících entitám. V rámci modulu `TraktShowModule` pro seriál lze vyhledat seriál podle jména metodou `FindShowByName`. Tato vrací seznam seriálů odpovídající vyhledávanému řetězci a obsahuje kromě jména seriálu také jeho identifikační čísla a rok prvního vysílání. Další metodou, tentokrát už pro získání přímo dat podle identifikačního čísla, je metoda `GetShowData`. Ta už vrací jen jeden seriál s kompletními informacemi o seriálu. Poslední metodou je `GetPeopleForShow`, která vrací seznam herců a členů štábu daného seriálu. V rámci osoby vrací jeho roli nebo charakter, jméno a identifikační čísla pro API. Stejné metody existují i v rámci modulu `TraktMovieModule` pro filmy.

V rámci modulu pro sezóny lze získat seznam sezón pro daný seriál. Modul pro epizody zahrnuje metody pro získání seznamu epizod pro sezónu a informace o konkrétní epizodě.

Informace o osobách lze získat přes modul `TraktPersonModule`, umožňující vyhledávat osoby a získat data konkrétní osobě. Posledním modulem je `TraktGenreModule` umožňující získat seznam žánru pro film nebo seriál.

Protože je z těchto metod potřeba získat více návratových hodnot (kromě samotných dat také stavové kódy), tak pro asynchronní metody byly vytvořeny datové typy `ItemReturnType` a `ListReturnType`. Ty obsahují objekt (seznam objektů) a pravě stavový kód, který se používá pro kontrolu, jakým způsobem proběhla operace. Metoda `GetDataFromTrakt` třídy `TraktConnection` využívá jako návratový typ `HttpReturnType`, který obsahuje spolu se stavovým kódem i JSON uložený v textové formě.

Jelikož API vrací JSON, je tyto data potřeba převést do jednotlivých objektů. Pro tyto účely slouží třída `JsonConverter`. Vstupem těchto metod není přímo JSON řetězec, ale dynamický objekt vytvořený konverzí JSON pomocí třídy `JavaScriptSerializer`. V rámci třídy je pro každý ze získaných objektů (seriál, sezóna, epizoda, film, osoba a žánr) vytvořen konvertor. Konvertory pro prvních 5 zmíněných objektů jsou implementovány ve dvou režimech daných parametrem `type` výčtového typu `TraktQueryType`. Ty se odlišují množstvím převáděných (získaných) dat z API.

5.7 TmdbAPI

Projekt TmdbAPI je implementován podobně jako TraktAPI. Jedná se o projekt obsahující moduly k jednotlivým hlavním částem API. Základem je statická třída `TmdbConnection`, která se obdobně jako `TraktConnection` stará o inicializaci klienta a samotné získávání dat z API. Inicializace v rámci Tmdb API je oproti Trakt API jednodušší, jedná se v podstatě jen o nastavení základní adresy API. Oproti Trakt API je verze Tmdb API uvedena v přímo v URL, stejně jako klientský identifikátor. Ten je uveden v dotazovací sekci. Stejně tak se uvádí i jazyková lokalizace požadovaných dat. Příklad celého dotazu posílaného metodou `GetDataFromTmdb` pro získání informace o seriálu může vypadat např. následovně: `https://api.themoviedb.org/3/tv/61889?api_key=CLIENT_ID&language=cs-CZ`, identifikátor klienta je nahrazen slovním vyjádřením „CLIENT_ID“ z důvodu délky tohoto identifikátoru.

Jednotlivé dotazy jsou rozděleny do modulů, podle příslušnosti k návratovému typu. V aplikaci jsou implementovány moduly pro seriál, sezónu, epizodu, film a osobu. Služba je v aplikace použita pro získání české lokalizace některých dat. Jedná se o přehled (overview) pro seriál, sezónu, epizodu a film. Pro film také pro získání lokalizované verze sloganu (tagline). Pro osoby se jedná o lokalizovanou verzi jejich biografie. Přestože se na službě mnoho českých biografií nenachází, aplikace možnost tato data stáhnout poskytuje.

Pro návratové hodnoty byly vytvořeny třídy, které umožňují vrátit jak samotná data, tak i návratový kód. Pro tyto účely slouží třídy `HttpReturnType` jako návratová hodnota samotného spojení (tedy metody `GetDataFromTmdb`) a `ItemReturnType` pro samotné metody jednotlivých modulů.

Služba vrací požadováná data ve formátu JSON. O převod těchto dat se stará třída `JsonConverter`, která konvertuje dynamický objekt na požadovány model. Tento konvertor byl vytvořen pro každou položku z implementovaných modulů.

Kapitola 6

Testování

V průběhu vývoje došlo i k testování průběžných změn a úprav aplikace. Testování probíhalo formou uživatelské interakce autora s programem, případně za pomoci SQL skriptů při testování databáze. Testování probíhalo v několika vlnách. První vlna zahrnovala otestování funkčnosti samostatné databáze. Za pomoci testovacích dat byly otestovány integritní omezení polí nebo cizí klíče.

V další vlně se jednalo o testování jednotlivých částí, které zastupují prvky návrhového vzoru MVVM. Testování spočívalo v kontrole jednotlivých příkazů, funkčnosti zobrazení vlastností entit a aktualizace dat. Testování probíhalo pomocí uživatelského rozhraní.

Poslední část testování se soustředila na aplikační programová rozhraní API. Testování nejprve probíhalo v konzolové aplikaci, následně přes příkazy uživatelského rozhraní.

V průběhu vývoje byla z API stažena data k desítkám seriálům a filmům. K některým z těchto filmů a seriálů bylo staženo také herecké složení včetně členů filmového štábu. Pro některé seriály byly staženy také informace o sezónách a epizodách. Při testování byly staženy k otestování také informace k některým osobám. V rámci služby Trakt zde nebyly nalezeny problémy, u služby TMDb díky otestování došlo k zjištění, že služba lokalizované bibliografie téměř neobsahuje, přesto došlo v aplikaci k zachování možnosti stažení těchto dat z API. Testovaná data byla následně kontrolována, jestli jsou shodná s informacemi uvedenými na webových stránkách těchto služeb.

Pro další vývoj by bylo vhodné vytvořit automatizované testy. Díky struktuře jednotlivých projektů je možno vytvořit testy pro každý projekt nezávisle na ostatních. Kromě testů pro projekty TraktAPI a TmdbAPI jsou testy vhodné i pro projekty Model a ViewModel zastupující částí návrhového vzoru.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat databázi pro aplikaci Databáze filmů a seriálů, včetně implementace samotné aplikace. Výsledkem tohoto úkolu je aplikace s grafickým rozhraním umožňující správu filmů a seriálů, a především umožňující stahování dat přes API služeb s touto tematikou.

Nejprve jsem v kapitole 2 probral jednotlivé technologie, které je možno použít, a detailněji ty, které jsem v rámci implementace použil. V rámci kapitoly 3 jsem se věnoval možnostem získávání dat z API. Zde jsem probral možnosti jednotlivých služeb, které byly implementovány, včetně služeb, které je možno implementovat v rámci budoucího vývoje. Jako základní API v průběhu vývoje bylo zvoleno API služby Trakt, přestože služba TMDb poskytuje více možností zisku dat. V rámci dalšího vývoje by bylo rozumné data získávat z druhé služby.

Aplikace byla po prozkoumání možností implementována pod platformou .NET, především v jazyce `C#` s využitím Microsoft SQL Server pro databázovou část. V průběhu návrhu byly zkoumány jednotlivé API některých webových služeb zaměřených na filmy a seriály. V průběhu samotné implementace jsem nejprve vytvořil databázi (5.1). Následným krokem bylo vytvořený jednotlivých prvků návrhového vzoru Model(5.2)-View(5.4)-ViewModel(5.3). Poslední fází bylo napojení na API služeb Trakt (5.6) a The Movie Database (5.7).

Aplikace bude i nadále vyvíjena a rozšiřována. Jako jeden z hlavních kroků v dalším vývoji je vytvoření instalačního souboru. O vytvoření tohoto souboru byly pokusy i při vývoji v rámci bakalářské práce. Problém při instalaci souvisí s instalací databázového systému na uživatelský počítač. Po konzultaci s Ing. Romanem Jaškem byla jako jedna z možností navrhnut přesun z databázového systému Microsoft SQL Server na SQLite. Dalším plánovaným rozšířením je větší množství oken pro zobrazení dat, například seznam epizod, které vyšli v daném úseku. Mezi další rozšíření lze uvést větší propojení oken tak, aby se šlo přímo dostat ze seznamu herců na kartu herce nebo implementace vyhledávání a třídění entit.

Literatura

[1] *Co je API a jak jej používat ve vašem podnikání | onemark.cz*. [Online; navštíveno 7.5.2017].
URL http://www.onemark.cz/clanky/co-je-api-a-jak-jej-pouzivat-ve-vasem-podnikani

[2] *Common Type System*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/en-us/library/zcx1eb1e(v=vs.110).aspx

[3] *Data Binding Overview*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx

[4] *Entity Framework Code First to an Existing Database*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/cs-cz/data/jj200620

[5] *Introducing LocalDB, an improved SQL Express – SQL Server Express WebLog*. [Online; navštíveno 7.5.2017].
URL https://blogs.msdn.microsoft.com/sqlexpress/2011/07/12/introducing-localdb-an-improved-sql-express/

[6] *Introduction to WPF*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/en-us/library/mt149842(v=vs.110).aspx

[7] *JSON Introduction*. [Online; navštíveno 7.5.2017].
URL https://www.w3schools.com/js/js_json_intro.asps

[8] *MS SQL Server Quick Guide*. [Online; navštíveno 7.5.2017].
URL https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_quick_guide.htm

[9] *Seznam DVD - Průvodce verzí 2008 - Filmová databáze online FDb.cz*. [Online; navštíveno 7.5.2017].
URL https://www.fdb.cz/seznamdvd/nahledy-pruvodce.html#akce0

[10] *The Movie Database API*. [Online; navštíveno 7.5.2017].
URL https://developers.themoviedb.org/3/getting-started

[11] *The MVVM Pattern*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/en-us/library/hh848246.aspx

[12] *Trakt.tv API · Apiary*. [Online; navštíveno 7.5.2017].
URL http://docs.trakt.apiary.io/#

[13] *T-SQL Quick Guide*. [Online; navštíveno 7.5.2017].
URL https://www.tutorialspoint.com/t_sql/t_sql_quick_guide.htm

[14] *What is Entity Framework?* [Online; navštíveno 7.5.2017].
URL http://www.entityframeworktutorial.net/what-is-entityframework.aspx

[15] *Working with App.xaml - The complete WPF tutorial*. [Online; navštíveno 7.5.2017].
URL http://www.wpf-tutorial.com/wpf-application/working-with-app-xaml/

[16] *XAML Overview (WPF)*. [Online; navštíveno 7.5.2017].
URL https://msdn.microsoft.com/en-us/library/mt149842(v=vs.110).aspx

[17] *XML Introduction*. [Online; navštíveno 7.5.2017].
URL https://www.w3schools.com/xml/xml_whatis.asp

[18] Čapek, J.: *Api Online Webových Aplikací*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2016, vedoucí práce Samek Jan.

[19] Dajbych, V.: *MVVM: Model-View-ViewModel*. [Online; navštíveno 7.5.2017].
URL http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[20] ECMA International: *Standard ECMA-334 - C# Language Specification*. June 2006.
URL
http://www.ecma-international.org/publications/standards/Ecma-334.htm

[21] Guthrie, S.: *Announcing SQL Server on Linux - The Official Microsoft Blog*. [Online; navštíveno 7.5.2017].
URL https://blogs.microsoft.com/blog/2016/03/07/announcing-sql-server-on-linux/#sm.001fln9dg1ebndv71014x31fu0g8x

[22] Petrusha, R.; Chraiet, M. W. M.; Wagner, B.: *Get started with the .NET Framework | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https:
//docs.microsoft.com/cs-cz/dotnet/articles/framework/get-started/index

[23] Čápka, D.: *1. díl - Úvod do C# a .NET frameworku*. [Online; navštíveno 7.5.2017].
URL https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

[24] Puš, P.: *Poznáváme C# a Microsoft.NET 40. díl – serializace – Živě.cz*. [Online; navštíveno 7.5.2017].
URL http://www.zive.cz/Clanky/Poznavame-C-a-MicrosoftNET-40-dil-serializace/sc-3-a-126553/default.aspx

[25] Rubeš, O.: *Upravujeme uživatelský profil a představujeme Bednu | Blog | Edna.cz*. [Online; navštíveno 7.5.2017].
URL http:
//www.edna.cz/blog/upravujeme-uzivatelsky-profil-a-predstavujeme-bednu/

[26] Wagner, B.; Latham, L.; Wenzel, M.: *Introduction to the C# Language and the .NET Framework | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https://docs.microsoft.com/cs-cz/dotnet/articles/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[27] Wagner, B.; Petrusha, R.; Wenzel, M.: *Properties (C# Programming Guide) | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https://docs.microsoft.com/en-us/dotnet/articles/csharp/programming-guide/classes-and-structs/properties

[28] Wagner, B.; Wenzel, M.: *Fields (C# Programming Guide) | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https://docs.microsoft.com/en-us/dotnet/articles/csharp/programming-guide/classes-and-structs/fields

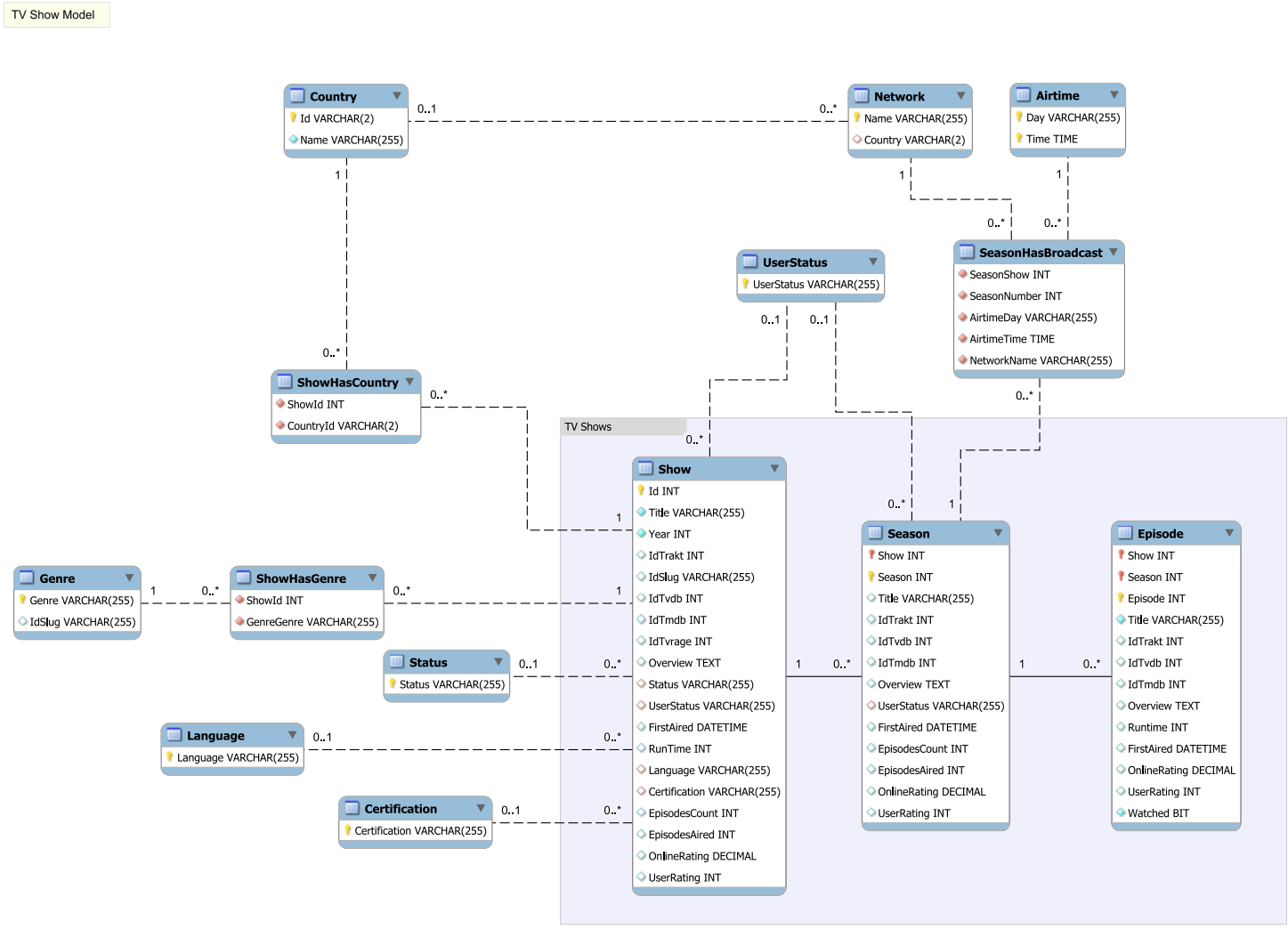
[29] Wagner, B.; Wenzel, M.: *Generics (C# Programming Guide) | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https://docs.microsoft.com/en-us/dotnet/articles/csharp/programming-guide/generics/index

[30] Wenzel, M.; Wagner, B.: *.NET Core and Open-Source | Microsoft Docs*. [Online; navštíveno 7.5.2017].
URL https://docs.microsoft.com/cs-cz/dotnet/articles/framework/get-started/net-core-and-open-source

Přílohy

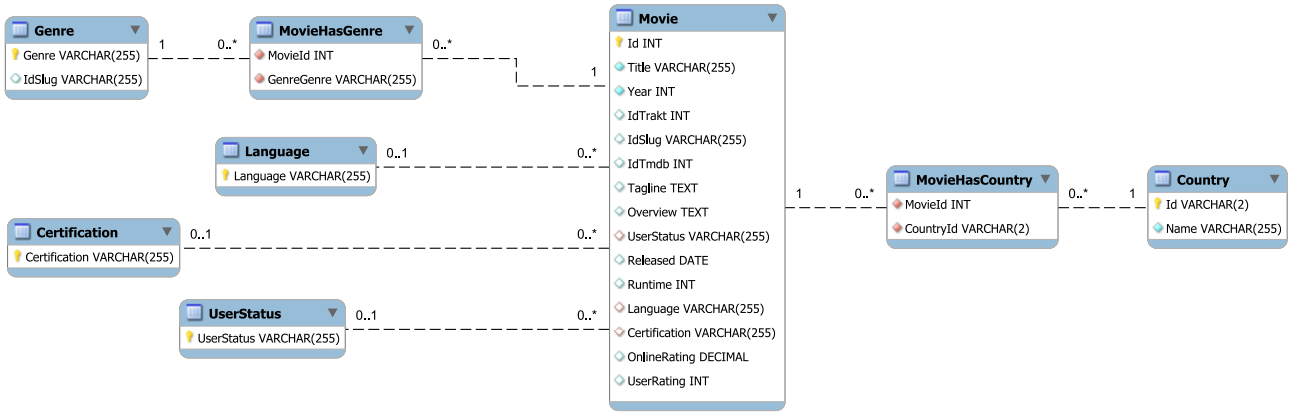
Příloha A

ER diagramy

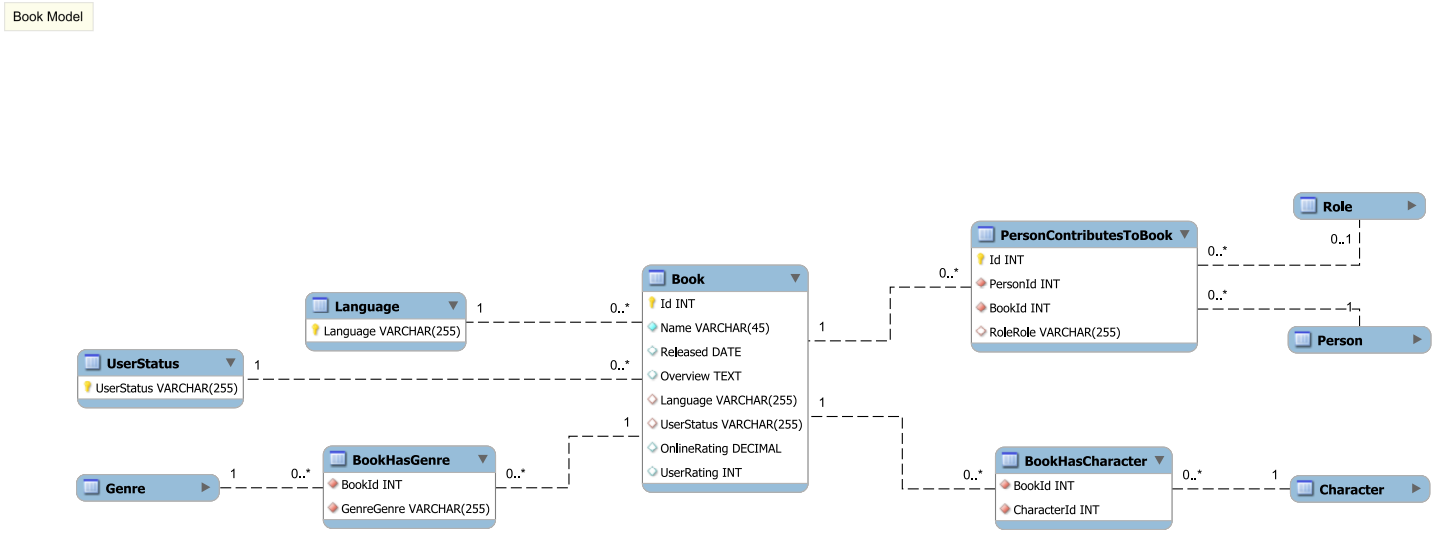


Obrázek A.1: Diagram pro seriál

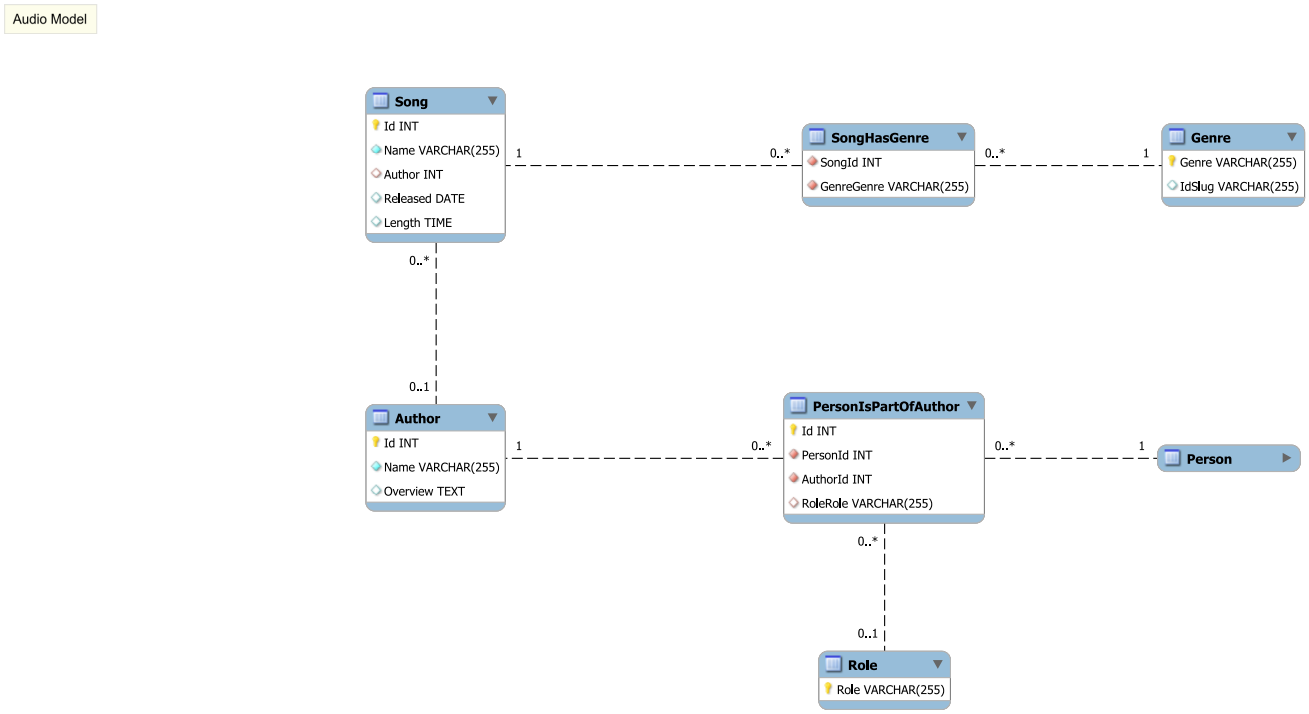
Movie Model



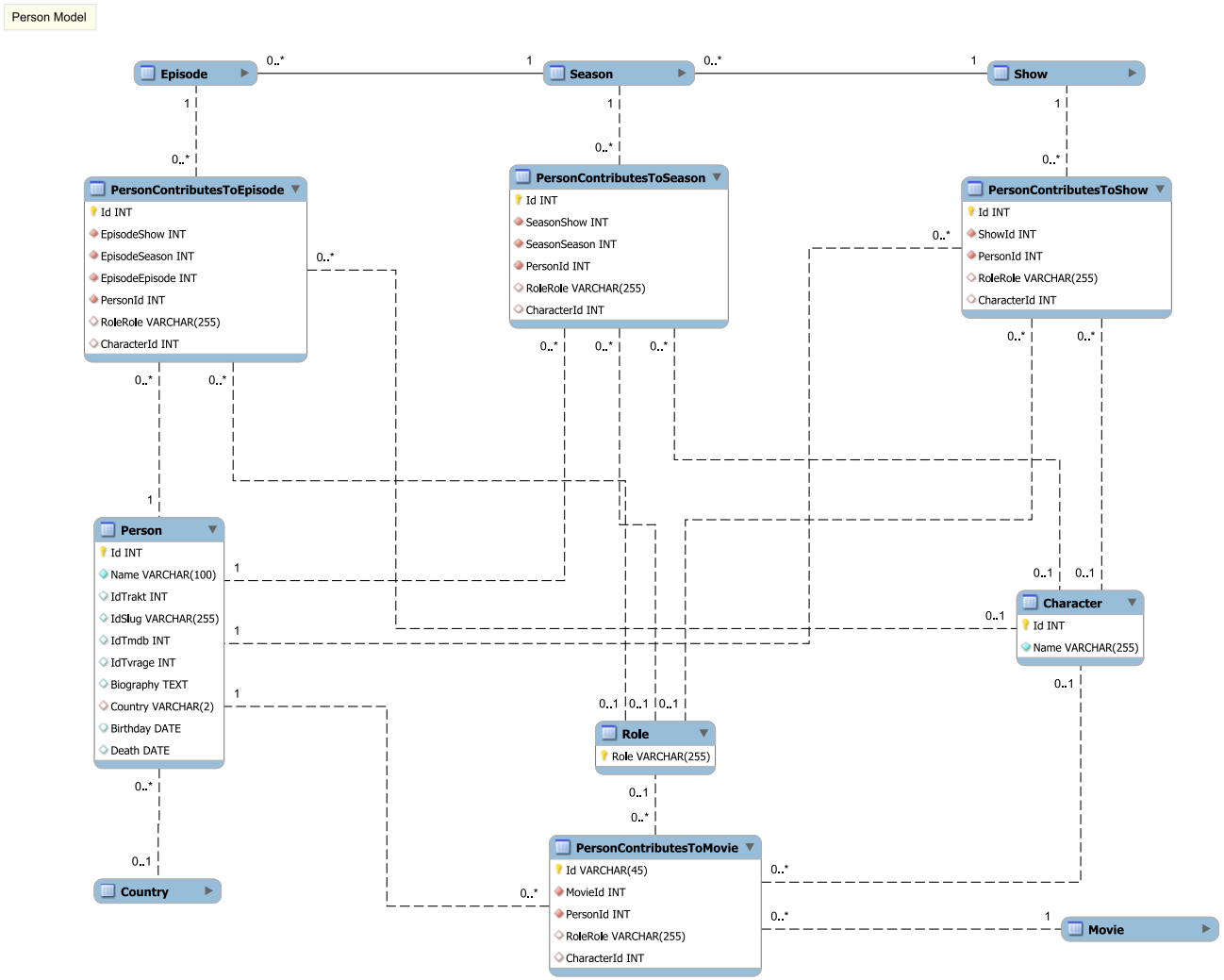
Obrázek A.2: Diagram pro film



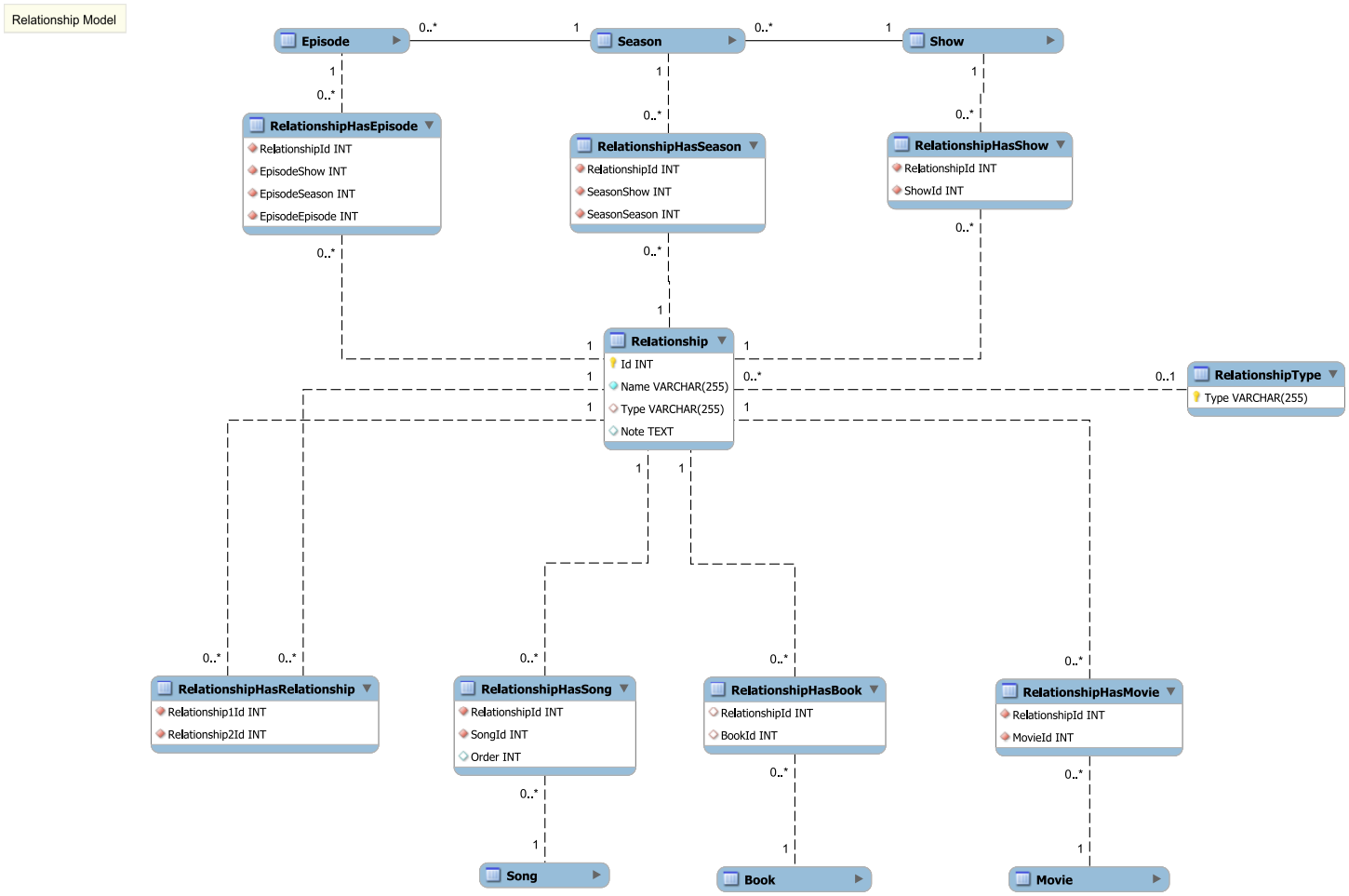
Obrázek A.3: Diagram pro knihu



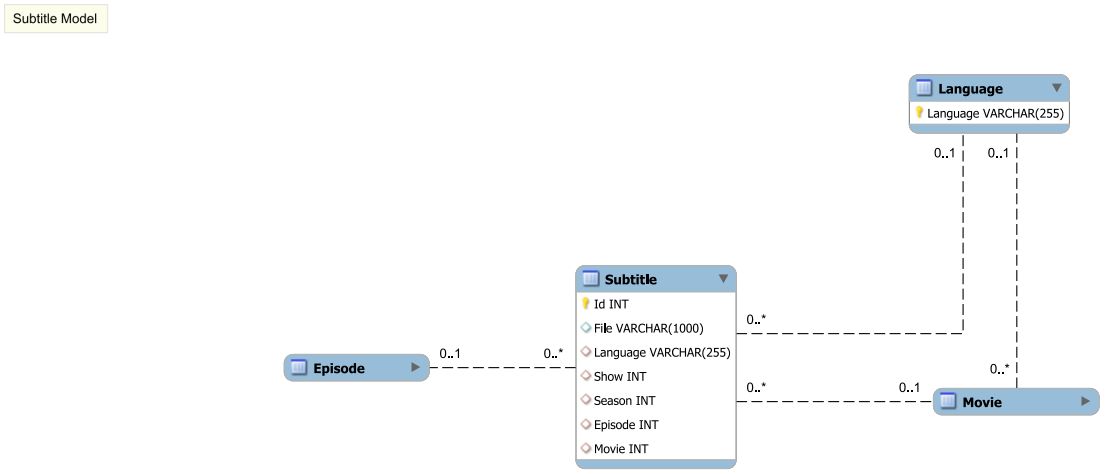
Obrázek A.4: Diagram pro hudbu



Obrázek A.5: Diagram pro osoby



Obrázek A.6: Diagram pro vztah



Obrázek A.7: Diagram pro titulek