ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Hudební přehrávač a organizér s grafovou reprezentací dat
Student:	Bc. Kryštof Šplíchal
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Na základě analýzy požadavků navrhněte, implementujte a otestujte lokálně běžící přehrávač a organizér hudební knihovny s webovým rozhraním. Hlavním konceptem je použití grafu pro reprezentaci vazeb mezi základními entitami (umělec, album, žánr a další).
Funkcionalita by měla zahrnovat tyto oblasti:
- Tvorba, správa a procházení hudební knihovny (z lokálních souborů v adresáři) podobně jako u tradičních přehrávačů, včetně pohledů na základní entity ve stylu seznamu s možností filtrace.
- Možnost definování vlastních vztahů mezi entitami, což umožní propojování, seskupování a označování entit dle specifických potřeb uživatele.
- Vizualizace entit a vztahů mezi nimi pomocí interaktivních grafových pohledů.
- Generování seznamů skladeb (playlistů) na základě grafové struktury podle uživatelem zvolených kritérií a správa těchto playlistů.
- Automatická agregace informací o albech z veřejně dostupného zdroje propojených dat.
- Zajištění vlastního přehrávání hudby.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Hudební přehrávač a organizér s grafovou reprezentací dat

Bc. Kryštof Šplíchal

Vedoucí práce: Ing. Jaroslav Kuchař, Ph.D.

2. května 2017

Poděkování

Chtěl bych tímto poděkovat Ing. Jaroslavu Kuchařovi, Ph.D. za odborné vedení práce a cenné rady poskytnuté v průběhu její realizace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. května 2017 .....

Odkaz na tuto práci

Šplíchal, Kryštof. *Hudební přehrávač a organizér s grafovou reprezentací dat.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem práce je vytvořit opensource webovou aplikaci pro správu a přehrávání lokální hudební knihovny s novým pohledem na data. Motivací je umožnit posluchačům hudby na základě grafové reprezentace dat zadávat vlastní vazby mezi entitami. Aplikace zároveň následuje standardy používané v tradičních hudebních přehrávačích. Grafová struktura je využita pro interaktivní vizualizace, seskupování entit a generování seznamů skladeb. Cílem těchto funkcí je usnadnit uživatelům procházení a používání rozsáhlých hudebních knihoven, tato problematika není jinde dostatečně řešena. Práce popisuje doménu přehrávačů, proces a výsledky analýzy, návrhu a implementace aplikace a testování uživatelského rozhraní.

V rámci práce jsem vytvořil aplikaci sestávající ze serverové a klientské části. Pro řešení problému jsem využil grafovou databázi a různé specifické opensource knihovny pro dílčí části. Navrhnul a implementoval jsem algoritmus pro generování seznamů skladeb na základě grafové struktury. Vytvořil jsem dotazy pro agregaci metadat o skladbách z propojeného zdroje dat. Testování ověřilo funkčnost implementovaných definovaných scénářů a identifikovalo některé problematické oblasti. Projekt je řádně připraven pro další opensource vývoj v rámci komunity.

Práce přináší novou metodu zpracování lokální hudební knihovny a vzniklá aplikace umožňuje komunitě vyvíjet na jejím základě další funkce pro zlepšení navigace v rozsáhlých hudebních knihovnách.

Klíčová slova přehráváč hudby, organizér hudební knihovny, graf, grafová vizualizace, generování playlistu, agregace metadat

Abstract

The goal of this thesis is to create an opensource web application for managing and playing a local music library with a new view on the data. The motivation is to allow the listeners of music to input custom relationships between the main entities. This results in a graph data structure. The application also follows the standards that are implemented in traditional music players. The graph structure is used for interactive visualisations, grouping of entities and generation of music playlists. The goal of those functions is to make browsing and using large music libraries easier for the end user. This has not yet been researched enough. The thesis describes the domain of music players, the process and results of analysis, design and implementation of the application and testing of the user interface.

I have created an application consisting of server and client parts. For the solution, I have used a graph database and various specific opensource libraries for the different parts. I have designed and implemented an algorithm for generation of music playlists based on the graph structure. I have created queries used for aggregation of track metadata from a linked data source. The testing has verified the functionality of implemented defined scenarios and has identified some problematic areas. The project is ready for further opensource community development.

The thesis gives a new method of working with a local music library and the created application allows the community to implement new features for better navigation in large music libraries using the application as the base.

Keywords music player, music library organizer, graph, graph visualisation, playlist generation, metadata aggregation

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem práce je vytvořit hudební přehrávač a organizér vlastní hudební knihovny, jehož přidaná hodnota oproti tradičním přehrávačům spočívá v možnosti propojovat relevantní entity speciálními vazbami. Tato vlastnost poskytne uživatelům nový pohled na data hudební knihovny, který lépe znázorní vztahy, které se mezi entitami vyskytují, ale není běžně možné je v aplikacích využít.

Vazby tvoří spolu s entitami graf, který poskytuje základ pro vizualizace různých pohledů na data a specifickou funkcionalitu založenou na procházení tohoto grafu.

Tato propojená podstata dat a z ní vycházející vysoká míra přizpůsobitelnosti je v existujících aplikacích domény hudebních přehrávačů zastoupena nedostatečně. Seskupování entit je realizováno nejčastěji pomocí prvků, které byly navrženy pro jiný účel, nebo neposkytují dostatečnou požadovanou granularitu (viz 1.1.1). Využitím uživatelem zadaných vztahů aplikace poskytuje systém, který lépe umožňuje zvolit si způsob, jakým data budou seskupena a třízena, a pohled na data, který je pro uživatele nejdůležitější.

Práce vedle problematiky grafové reprezentace také detailně řeší standardní funkcionalitu hudebních přehrávačů a organizérů, jejíž znalost je nezbytná pro realizaci celého projektu a správné začlenění originální funkcionality do uživateli očekávaného rozhraní.

Práce nejdříve podává přehled aktuálně dostupných hudebních přehrávačů, pojmů a operací spjatých s touto doménou. Na základě tohoto přehledu poukazuje na některé problémy vyvstávající z ustáleného modelu, který je ve většině těchto aplikací použit. Následně je provedena analýza sebraných požadavků, podle které je zpracován návrh nové aplikace a popsána její implementace. V následné kapitole je provedeno testování implementované aplikace a podány výsledky testování.

Cíle práce

Aplikace si klade za cíl umožnit uživatelům v rámci různých interaktivních pohledů propojovat entity podle libovolných kritérií v rámci specifikovaného modelu, při zachování očekávané funkcionality, která plyne z dlouhodobé zkušenosti uživatelů s hudebními přehrávači, a poskytnout další funkcionalitu, která především usnadní orientaci v rozsáhlých hudebních knihovnách. Pro úspěšnou implementaci je tak nutné analyzovat, jakým způsobem jsou realizovány základní procesy v podobných aplikacích, a navrhnout, jakým způsobem s těmito procesy budou kooperovat nové procesy, které rozšířenou funkcionalitu poskytují.

Aplikace umožní přehrávat lokálně dostupné hudební soubory, vytvořit a spravovat hudební knihovnu, zadávat různé typy vazeb a propojovat tak entity, prohlížet vzniklý graf v rámci různých interaktivních kontextových pohledů. Dále zajistí funkcionalitu generování sekvence skladeb v reálném čase a usnadní zadávání metadat pro entity pomocí agregace těchto dat z veřejného zdroje.

Systém bude vedle specifické funkcionality poskytovat i klasické, často používané funkce, na které jsou již uživatelé zvyklí v ustálené podobě, v jaké tato funkcionalita bývá implementována v jiných prohlížečích - příkladem může být pohled na podmnožinu dat ve formě filtrovaného seznamu, editace entity ve formuláři, apod.

Opensource podstata

Vedle zmíněných cílů je silnou motivací snaha o vytvoření platformy, na jejímž základě postupně vznikne komunita uživatelů, kteří vedle běžného používání aplikace zároveň budou mít zájem přispívat k rozvoji další funkcionality a spravovat tu stávající. Z tohoto důvodu byla zvolena `opensource` licence, viz sekce 1.5.

Vzhledem k absenci aplikací tohoto typu (viz sekce 1.3) lze v budoucnu očekávat aktivitu zmíněné komunity. Pojetí systému jako platformy pro další rozvoj však neznamená, že aplikace vytvořená v rámci této práce není funkčním celkem - na základě analyzovaných požadavků byly vybrány nejdůležitější případy užití, které jsou implementovány.

Kapitola 1

Popis zkoumané domény

V této kapitole je podán přehled domény z pohledu aplikací s funkcionalitou přehrávače a hudebního organizéru. Na základě provedeného průzkumu často používaných přehrávačů jsou popsány hlavní entity a procesy, které aplikace realizují. Popisy entit a procesů zahrnují rozbor okolností, které jsou důležité pro pozdější návrh aplikace. Dále jsou nastíněny některé problémy s realizací těchto procesů v přehrávačích, které bude aplikace adresovat.

Poslech hudby je velmi oblíbenou aktivitou, kterou provozuje široká paleta různých posluchačů, jejichž potřeby a preference se v některých případech značně liší. Spolu s popularitou poslechu hudby roste poptávka po aplikacích, které jednotlivým potřebám uživatelů vyhovují.

Na trhu se vyskytuje velké množství aplikací, které se snaží poskytnout kvalitní řešení. Mnoho těchto aplikací je velmi vyspělých a mají velkou a věrnou uživatelskou základnu. Míra variability těchto řešení je poměrně nízká, aplikace jsou v jádru velmi podobné, liší se většinou přídavnou funkcionalitou, která se základními principy fungování přehrávače a organizéru nesouvisí.

Vedle tradičně založených přehrávačů existují různé specializované nástroje, které se zaměřují na úzkou podmnožinu této problematiky - např. nástroje na úpravu metadat přímo v souborech (viz 1.1.2.3).

Navrhovaná oblast přidané funkcionality aplikace může připomínat různé doporučovací služby. Doména je skutečně podobná, avšak zde realizovaná aplikace se liší v principu operace. Data používaná v aplikaci jsou lokální soubory a vazby vznikají přímo uživatelskou akcí. Doporučovací služby zpravidla pracují s velkými objemy uživatelských dat nebo přímo analyzují obsahy skladeb a snaží se na základě těchto dat vypočítat podobnost entit [1]. Dále bývají tyto služby (příkladem [1], [2]) úzce spjaty s poskytováním samotného multimediálního obsahu - aplikace (alespoň v této fázi) bude pracovat nad lokální knihovnou hudebních souborů.

[1] Google Play Music - `https://play.google.com/store/music`

[2] Apple iTunes - `https://www.apple.com/cz/itunes/`

1.1 Důležité pojmy a operace

V doméně hudebních přehrávačů se vyskytují základní entity a procesy, které zde pro úplné pochopení popisované domény uvádím. Tyto pojmy se budou vyskytovat dále v práci v obou jazykových variantách, v závislosti na kontextu - pro popisy používám české ekvivalenty, v aplikaci jsou entity pojmenovány anglicky pro zachování konzistence a srozumitelnosti.

Uvedené popisy vychází z průzkumu funkcionality oblíbených hudebních přehrávačů [3], který jsem provedl v rámci analýzy. Zkoumané přehrávače jsou dostupné pro různé operační systémy na PC, omezil jsem se v popisech na tuto oblast z důvodu podobnosti s navrhovanou aplikací, v jiných oblastech (např. mobilní zařízení) je však situace velmi podobná.

1.1.1 Základní entity

Základní entity v následujícím popisu jsou průnikem všech zkoumaných přehrávačů. Entity se v aplikacích objevují v téměř totožné podobě, která se v této oblasti po dobu vývoje ustálila. Rozdíly mezi aplikacemi jsou spíše v konkrétním provedení základních operací a obecné rozšiřující funkcionalitě.

Nepodařilo se mi najít literaturu, která by se podrobněji zabývala rozborem těchto entit, popisy tedy vychází zejména z vlastního průzkumu fungování zmíněných aplikací, příp. z článků uvedených u konkrétních aspektů.

Pro účely konzistentní katalogizace hudebních děl a událostí nabízí velmi podrobný model domény organizace hudby např. hudební databáze MusicBrainz [2]. Tento model je značně složitější, než u zmíněných aplikací. Pokrývá i další rozšířené vlastnosti, které jsou vhodnější z pohledu katalogizace a nefigurují tradičně v přehrávačích. Nejsou v něm však specifikovány např. entity „žánr“ a „playlist“, které jsou také často používány v tradičních přehrávačích. I přes odlišnou povahu modelu jsou části této databáze v aplikaci využity pro agregaci metadat, což je popsáno detailněji v dalších kapitolách.

1.1.1.1 Track - skladba

Skladba je reprezentace hudebního díla ve formě souboru, který lze přehrát, a náleží do alba. Definice skladby je pevně vázána k definici alba, jelikož za skladbu je považována určená oddělená sekce na hudebním nosiči - tedy např. krátká pořízená nahrávka nějakého zvuku by tuto definici nesplňovala a bez vazby na album by v tomto kontextu neměla smysl.

Hudební soubory obsahující hudební díla mohou mít na PC různý formát. Formáty se podle výsledné kvality přednesu dělí na ztrátové a bezztrátové [3]. Ztrátové formáty se používají zejména tam, kde je důležité udržet co nejmenší velikost souboru při akceptovatelné kvalitě - tento proces je odstupňován podle

[3] Jmenovitě Winamp, Apple iTunes, Windows Media Player, MusicBee, Foobar2000 a VLC

požadované výsledné kvality. Mezi často používané bezztrátové formáty patří FLAC, ALAC a WMA. Časté ztrátové formáty jsou MP3, AAC, OGG (Vorbis). [3] Důležitou částí každého hudebního přehrávače je systém pro dekódování těchto formátů (jejich podpora se napříč spektrem přehrávačů liší) a jejich přehrání na cílovém systému.

1.1.1.2 Album

Album je uspořádaná množina skladeb. Obsah alba i pořadí jednotlivých skladeb je určeno při vydávání alba, skladby jsou takto zaneseny na příslušný datový nosič (nebo nosiče) a distribuovány. Ve speciálních případech může album obsahovat např. jednu skladbu, která je delší, nebo kombinuje více skladeb - příkladem může být nahrávka živého koncertu. Některé organizace kladou na alba požadavky - např. UK Albums Chart požaduje, aby album mělo trvání alespoň 25 minut nebo aby obsahovalo více než 4 skladby [4]. Uskupení skladeb spolu s názvem jsou unikátním identifikátorem každého alba.

Alba se dělí podle podstaty hudebního obsahu. Nejčastější alba jsou studiová, dalšími typy jsou alba koncertní, kompilace a soundtracky (hudba z filmu nebo hry). Toto dělení je podstatné pro porozumění způsobu, jakým uživatel na dané album nahlíží a jak vnímá jeho vztahy v hudební knihovně.

Album může být dále rozděleno na více disků (zde se používá abstrakce ve slově disk, nerozlišuje se např. mezi CD a vinylem), u většiny hudebních děl bývá typický počet CD 1 nebo 2, více CD mají různé hudební kompilace, např. celoživotní dílo klasického skladatele nahrané jedním orchestrem. Podle preference uživatele a možností aplikace lze každé CD považovat za samostatné album s příznakem udávajícím pořadové číslo CD nebo lze disky uvádět jako rozdělení skladeb v rámci jednoho alba, což aplikace viditelně oddělí.

Některá alba mají více vydání (tzv. release) - důvodem může být vytvoření speciální edice pro nějakou zemi nebo specifický účel. Taková alba mohou obsahovat tzv. bonusové skladby, které se na jiných vydáních nenacházejí, a mohou se tak stát předmětem zájmu sběratelů. Za různá vydání lze také považovat distribuci na různých datových nosičích, alba se v současné době běžně distribuují digitálně, na CD a vinylových deskách, v minulosti byla také běžná distribuce na páskových kazetách.

1.1.1.3 Artist - umělec

Umělec je jedinec, hudební skupina nebo uskupení hudebníků, kterým je připisováno vytvoření alba. Entita umělce tedy zastřešuje široké spektrum možných autorů alba a závisí silně na cílové skupině, pro kterou je album určeno.

Typicky je umělec hudební skupina identifikovaná názvem, sestávající z několika hudebníků. V případě klasické hudby se může jednat např. o orchestr, dirigenta či sólistu. V případě elektronické hudby se často jedná o jednotlivce

se speciálním uměleckým jménem. Tato diverzita může způsobovat problémy při snaze o jednotné pojmenovávání umělců v hudební knihovně (viz 1.2.1).

1.1.1.4 Genre - žánr

Žánr reprezentuje příslušnost alba k hudebnímu stylu nebo tradici. Entita bývá používána v různých kontextech různými způsoby.

Z často používaných žánrů je možné uvést např. Rock, Jazz, Pop, Vážná hudba nebo Metal. Obecné hudební žánry lze dále dělit do podžánrů, které poskytují užší specifikaci. Zde lze ilustrovat zmíněnou kontextovost - nadšený posluchač konkrétního hudebního žánru má tendenci skupiny dále třídit do užších podžánrů, zatímco posluchač s méně obsáhlou knihovnou se častěji spokojí s obecným žánrem, přestože by bylo možné žánr specifikovat ještě podrobněji.

Byly učiněny mnohé snahy vytvořit standardní seznam hudebních žánrů pro účely zvýšení konzistence mezi jednotlivými knihovnami a službami [4], problém nekonzistence napříč knihovnami je však stále přítomen, protože je obtížné cílové poskytovatele nebo přímo konzumenty nutit tato rozdělení implementovat. Dalším problémem je odlišnost názorů na samotnou klasifikaci alb podle žánrů - mnohé používané žánry nejsou přesně definovány a překrývají se ve významu, problematika přiřazení žánru k albu je tím spíše velmi subjektivní.

1.1.1.5 Playlist - seznam skladeb

Playlist je uspořádaná množina skladeb nebo alb. Narozdíl od alba je playlist vytvořen zpravidla z iniciativy posluchače. Skladby bývají seskupeny podle libovolného kritéria a opět platí, že v používání této entity lze pozorovat velkou variabilitu. Někteří uživatelé využívají playlisty pro sestavení sekvence skladeb pro nějakou příležitost (např. taneční večer, posilovna, posezení s přáteli), jiné využití může být např. seskupení alb podle časového období nebo jiného společného atributu. V tomto případě je flexibilita použití malá a toto je jedna z oblastí, kterou se tato práce snaží vylepšit.

Problémy jsou způsobeny nesprávným používáním této entity a zároveň absencí vhodné funkcionality, která by problematiku řešila. Alba a skladby mají v tomto případě dané pořadí a jsou vizuálně seskupeny v jednom seznamu - chybí tedy možnost vytvářet specifičtější skupiny, provádět složitější dotazy a skladby mají povinně pořadí.

V případě online streamovacích služeb playlisty plní i sociální funkci sdílení. Např. v rámci služby YouTube je možné sdílet veřejný playlist [5], který seskupuje libovolná videa (v případě hudební domény oficiální nebo neoficiální klipy nebo samotné hudební dílo s jiným doprovodným vizuálním materiálem).

[4] Aktuálně velmi aktivní projekty tohoto typu jsou MusicMap - `http://www.musicmap.info/` a Musicalized - `http://en.musicalized.com/`

U lokálních přehrávačů je tato možnost omezená - playlist lze sdílet např. v podobě seznamu umělců a skladeb, přehrávače již neposkytují možnost sdílet i hudební soubory (mezi jinými důvody je na místě je zmínit, že sdílení duševního vlastnictví tohoto typu bývá nelegální).

1.1.2 Základní operace

Nad základními entitami jsou v přehrávačích implementovány operace, které ve stručnosti uvedu. Z používaných procesů jsem vybral ty, s kterými budou korespondovat nějaké procesy v navrhované aplikaci.

1.1.2.1 Organizace hudební knihovny

Za hudební knihovnu lze považovat interní reprezentaci základních entit v aplikaci spolu s rozšiřujícími metadaty a dalšími informacemi o entitách. Některé hudební přehrávače tuto funkcionalitu neobsahují [5] a omezují se tak pouze na přehrávání přímo ze systému souborů bez ukládání metadat v rámci aplikace, většina dostupných přehrávačů však organizaci poskytuje.

Problematika organizace knihovny zahrnuje především úpravu metadat entit (viz 1.1.2.3), tvorbu vlastních playlistů a zadávání hodnocení skladeb nebo alb.

Hodnocení se typicky zapisuje pomocí tzv. hvězdiček - uživatel má možnost skladbě nebo celému albu udělit 0 až 5 hvězdiček podle oblíbenosti, u některých přehrávačů lze použít i poloviční hvězdičky, např. 2 a půl hvězdičky. Přesný význam tohoto hodnocení není definován, záleží tedy na uživateli, jaký smysl jakému počtu hvězdiček přiřadí. Hvězdičkové hodnocení může hrát důležitou roli např. při generování playlistů (pokud je tato funkcionalita implementována) - oblíbené skladby mohou být vybírány častěji/dříve.

1.1.2.2 Přehrávání hudby

Přehrávání hudby je hlavní činnost, kterou poskytuje většina aplikací tohoto typu. V samotném procesu přehrávání mohou být drobné odchylky podle zvoleného návrhu a implementace, základ však bývá velmi podobný. Skladby (reprezentující celé hudební soubory) jsou přidány do hlavního playlistu, který je v uživatelském rozhraní rychle dostupný. Hlavní playlist je v principu velmi podobný playlistům popisovaným v 1.1.1.5, rozdíl je v tom, že tento playlist je v aplikaci pouze jeden, a lze do něj přidávat skladby, alba i celé uložené playlisty. Tyto entity se transformují na seznam skladeb s určeným pořadím, tento seznam je zpravidla stále viditelný a při přehrávání upravovatelný.

Ve chvíli, kdy se v playlistu nachází alespoň jedna skladba, lze zahájit samotné přehrávání. Tato akce je iniciována typicky dvojklikem na skladbu v playlistu, právě přehrávaná skladba je poté pro přehlednost vizuálně odlišena

[5] Např. VLC Media Player obsahuje v současné pouze velmi základní knihovnu

od ostatních skladeb. Přehrávání skladeb lze tradičně ovládat také pomocí známých ovládacích prvků. Prvky jsou tzv. **progress bar** - prvek zobrazující aktuální polohu přehrávání souboru, který zároveň posouváním indikátoru umožňuje polohu určit. Dále jsou přítomna tlačítka **stop** - zastavení přehrávání, **previous** - posun o 1 skladbu zpět, **play/pause** - přehrát/pozastavit, **next** - posun o jednu skladbu dále, **volume** - nastavení hlasitosti.

Přehrávače, které nejsou ve funkcionalitě velmi těsně vázány na reprezentaci souborů ve vnitřní hudební knihovně, umožňují přehrát skupinu souborů po přidání z kontextové nabídky, zadáním souborů jako části příkazu na příkazové řádce nebo přetažením skladeb do přehrávače v grafickém rozhraní. Tyto soubory se mohou a nemusí vyskytovat v hudební knihovně, pokud aplikace nějakou má. Tato funkcionalita je také závislá na cílové platformě, pro kterou je aplikace realizována.

1.1.2.3 Úpravy entit a tagování

Přehrávače obsahují funkcionalitu pro úpravu metadat základních entit. Tato metadata jsou uložena přímo v databázi aplikace nebo přímo v hudebních souborech jako tzv. tagy. Nejčastěji používanou variantou jsou tzv. ID3 tagy u MP3 souborů [6]. ID3 tagy bývají poskytovány spolu se soubory u digitálních distribucí hudebních děl, zároveň existují mnohé služby (např. MusicBrainz, Discogs, program beets), které k souborům příslušné tagy stahují ze své databáze.

ID3 tagy jsou definovány z pohledu jednotlivých souborů reprezentujících hudební skladby, v hudební knihovně se však pomocí těchto entit nebo dalších struktur na úrovni vnitřní databáze popisují i ostatní entity. Například přiřazením textové hodnoty reprezentující umělce tato entita vzniká v aplikaci a je možné ji následně přiřadit dalším albům či skladbám. Je tedy úkolem aplikace mapovat tyto hodnoty a udržovat konzistentní reprezentaci dat. Výhodou udržování metadat přímo v ID3 tags je, že při přenesení hudebních souborů jinam (např. na mobilní zařízení, kde se nachází jiný nezávislý přehrávač) se tyto údaje zachovávají. Problémem je omezená sada definovaných tagů, které nemusí stačit pro potřeby uživatele. Proto některé přehrávače volí variantu reprezentace metadat ve vlastní databázi, případně poskytují mechanismus zápisu povolených korespondujících tagů do souborů.

Pro účely ilustrace problematiky čtení a zápisu metadat ve formě tagů uvádím základní přehled používaných variant ID3. Další formáty mají své vlastní systémy metadat, jejichž podpora není ve všech přehrávačích ustálena.

ID3v1

Specifikace ID3v1 obsahuje pouze základní entity a omezení vycházející z fixní velikosti datové struktury, která tagy drží [7].

Název	**Omezení**
Song Title	30 znaků
Artist	30 znaků
Album	30 znaků
Year	4 znaky
Comment	30 znaků
Genre	1 bajt

Tabulka 1.1: Přehled ID3v1 tagů

ID3v2

Modernější variantou jsou ID3v2 tagy, pro ilustraci uvádím některé vybrané tagy v nejnovější verzi ID3v2.4. Hlavním požadavkem při vývoji tohoto standardu byla vysoká míra flexibility [8].

ID	**Obsah**
TALB	Album/Movie/Show title
TBPM	BPM (beats per minute)
TCOM	Composer
TFLT	File type
TIT2	Title/songname/content description
TLEN	Length
TMCL	Musician credits list
TMED	Media type
TOAL	Original album/movie/show title
TOFN	Original filename
TOLY	Original lyricist(s)/text writer(s)
TOPE	Original artist(s)/performer(s)
TPE1	Lead performer(s)/Soloist(s)
TPE2	Band/orchestra/accompaniment
TPE3	Conductor/performer refinement
TPE4	Interpreted, remixed, or otherwise modified by
TPUB	Publisher
TRCK	Track number/Position in set
TORY	Original release year

Tabulka 1.2: Přehled vybraných ID3v2.4 tagů

1.2 Vybrané problémy

Motivací pro vznik nové aplikace je také pozorování některých problémů, které vychází z tradičního modelu realizovaného zkoumanými přehrávači, který v

jistých případech není dostatečně flexibilní. Následuje popis těchto vybraných problémů.

1.2.1 Tagování

Problémy s konzistencí při tagování skladeb způsobuje odlišnost pojetí pojmenování různých umělců. Jak bylo naznačeno v 1.1.1.3, v některých kategoriích (např. klasická hudba) se naskýtá více možností, jak vyplnit pole Artist. Např. v případě orchestrální nahrávky by bylo možné jako umělce podle preferencí zvolit název orchestru, jméno dirigenta, jméno skladatele, jména sólistů a další. Přestože standard ID3v2 poskytuje i pole jako Composer (skladatel) nebo Conductor (dirigent), které umožňují korektní zanesení těchto informací, uživatelé i aplikace se často omezují na využívání základních entit. Hlavní pohledy aplikací jsou nejčastěji centrovány kolem pohledů na seznamy umělců (pole Artist) a alb.

Dalším problémem může být, pokud album patří pod více umělců. Existují např. nahrávky, kde orchestr zpracuje díla dvou skladatelů, a toto dílo je vydáno jako jedno album. V případě, kdy uživatel za umělce dosazuje skladatele (např. z důvodu přenositelnosti této informace do přehrávačů, které nepodporují ID3v2), nastává problém, kdy je nucen zadat do pole Artist textovou kombinaci obou skladatelů, což způsobuje nekonzistenci - pokud si uživatel zobrazí alba od skladatele-umělce „X“ a dané album je zařazeno např. pod umělci „X & Y“, toto album nebude v předchozím dotazu zahrnuto. Některé přehrávače proto umožňují zadat jako umělce více hodnot (což podporují přímo i ID3v2 tagy).

V případě digitálních distribucí hudebních děl způsobuje komplikace také nedostatečná pozornost ke korektnímu vyplnění všech tagů, tento problém se snaží řešit různé tagovací služby, např. již zmíněné MusicBrainz.

Navrhovaná aplikace z těchto důvodů volí odlišný model, který využívá propojování entit a je popsán v kapitole 3.

1.2.2 Pohledy na data

V oblasti přehrávačů, které pracují nad lokální knihovnou, jsou realizovány typické pohledy v podobě filtrovatelných seznamů umělců nebo alb. Pohledy se lehce liší v provedení, v principu poskytují stejnou funkcionalitu. Toto řešení může být problematické pro velké knihovny z pohledu rychlé orientace v knihovně.

Z požadavků analyzovaných v následující kapitole lze vyvodit potřeba uživatelů se v knihovně rychle vizuálně orientovat, při standardním modelu však neexistuje způsob, jakým toho dosáhnout. Uživatel je odkázán na svoji paměť a představivost při hledání hudby pro poslech. Způsob reprezentace dat lze popsat spíše jako katalogizaci, chybí možnost zaznamenat specifické vztahy

mezi entitami, které lze následně využít v personifikovaných pohledech. Toto je další z oblastí, pro kterou se práce snaží poskytnout řešení.

1.3 Podobné aplikace a koncepty

Před provedením analýzy požadavků jsem provedl průzkum, jehož cílem bylo najít podobné aplikace nebo platformy.

Existuje velké množství komerčních i nekomerčních hudebních přehrávačů a organizérů hudební knihovny. Přehrávání hudby je velmi populární aktivita a svědčí o tom i množství řešení pro poslech hudby - na PC i na jiných zařízeních. Úspěšné aplikace na PC poskytují funkcionalitu, která je popsána v předchozích sekcích, je v programech velmi podobná a liší se spíše v detailech - příkladem může být přizpůsobitelnost vzhledu, podpora synchronizace přenosných zařízení, přehrávání audioknih, apod. Další průzkum těchto rozdílů není pro tuto práci přínosný, jedná se o funkce, které mají doplňkový charakter, nesoustředí se na podstatu zpracování dat programem a je možné je následně doimplementovat.

V rámci průzkumu jsem nenalezl žádnou aplikaci ani platformu, která by umožňovala nad hudební knihovnou provádět v požadovaném rozsahu speciální operace podobné těm, které bude obsahovat tato aplikace. Překryv funkcionality mezi tradičními aplikacemi a novou aplikací bude pochopitelně právě v již popsaných základních schopnostech, kterými jsou procházení entit v seznamech, přehrávání hudby, možnost editovat entity, funkcionalita playlistů a další.

Aktuálním trendem je přechod od lokálních hudebních knihoven do cloudu. Mezi velmi populární služby se v současnosti řadí např. Google Music, Apple iTunes nebo Spotify - tyto platformy umožňují přehrávat hudbu po zaplacení předplatného nebo po zakoupení konkrétních hudebních děl a poskytují možnost tuto hudbu přehrávat jednoduše i v přenosných zařízeních. Výhodou těchto služeb je propracovaný systém doporučování hudby a funkcionalita sociální sítě. Právě zde lze hovořit o jisté podobnosti s podstatou této aplikace, princip je však odlišný, jak bylo popsáno v úvodu této kapitoly.

Obchodní model striktně tzv. streamovacích služeb (Spotify, Deezer a další) obvykle nezahrnuje možnost hudební soubory používat mimo klienta aplikace. Hudba je poskytována v komprimovaných formátech (zejména MP3) pro účely streamování, což může být místem pro kritiku uživatelů, kteří kvalitu souborů považují za důležitou vlastnost. Uživatel je tedy odkázán na aplikační řešení poskytnuté provozovatelem služby.

Někteří uživatelé tedy i přes popularitu tohoto trendu stále preferují mít vlastní lokální hudební knihovnu a procházet tato data programem, který splňuje jejich konkrétní potřeby.

1.4 Cílová skupina

Specifickou funkcionalitu, kterou aplikace bude poskytovat, využije jen podmnožina uživatelů, kteří poslouchají hudbu na PC. Aplikace nebude mít smysl např. pro uživatele, jejichž hudební knihovna má desítky hudebních alb nebo poslouchají hudbu pouze online pomocí nějaké služby nebo nemají potřebu detailně třídit entity ve své knihovně. Přestože by takto aplikace také fungovala, přidaná hodnota se projeví až s většími objemy dat, zadáním a využíváním zmíněných vazeb.

Aplikace je tedy určena zejména pro hudební nadšence, hudebníky nebo sběratele, kteří mají velké hudební knihovny (řádově desetitisíce skladeb a výše) a potřebují specifičtější pohledy na data, než poskytují tradiční přehrávače. Jediná další omezení vyplývají ze zvolené platformy, uživatelé budou s aplikací interagovat přes počítač nebo jiné vhodné zařízení s webovým prohlížečem.

Vzhledem ke zvolenému způsobu vývoje lze také očekávat, že v počátečních fázích vývoje (po uvolnění kódu implementovaného v této práci) bude uživatelská báze sestávat z velké části z jedinců, kteří se na vývoji také mají zájem podílet. V momentě, kdy bude aplikace uznána dostatečně „vyspělou“ pro širší trh, by se tato báze měla rozšířit i o „běžné“ uživatele, kteří chtějí aplikaci pouze používat.

1.5 Zvolená licence pro software

Práce si klade za cíl vytvořit aplikaci, která bude dále vyvíjena v rámci open-source komunity. Z tohoto důvodu jsem zvolil licenci `MIT`[6]. Projekt bude uvolněn na velmi rozšířenou platformu pro správu projektů GitHub[7].

[6] `https://opensource.org/licenses/MIT`
[7] `https://github.com`

Kapitola 2

Analýza

V rámci analýzy problému jsem definoval systémové požadavky. Na jejich základě jsem vypracoval případy užití. Na konci kapitoly je uvedena tabulka s přehledným mapováním vzniklých případů užití na funkční požadavky. Výstupy z provedené analýzy poslouží k formulaci návrhu aplikace.

2.1 Důležité používané pojmy

V požadavcích a scénářích se nově vyskytují pojmy, které zde definuji z důvodu snadnější orientace v textu. Další časté základní pojmy a operace jsem definoval v sekci 1.1.

Grafový pohled

Pohled na data, který využívá interaktivní vizualizace grafové struktury.

Seznamový pohled

Pohled na data využívající zobrazení ve formátovaném seznamu.

Playlisty

V textu je rozlišován hlavní playlist a uložené playlisty. Hlavní playlist odpovídá oblasti popsané v sekci 1.1.2.2, je to struktura, ve které jsou zařazeny skladby určené k okamžitému přehrání. Uložené playlisty jsou uložené seznamy skladeb určené k pozdějšímu přehrání, je možné je zařadit do hlavního playlistu.

2.2 Systémové požadavky

Pro úspěšnou analýzu problému a návrh řešení výsledné aplikace bylo nejprve nutné formulovat požadavky na systém vycházející z potřeb různých uživatelů. Tyto požadavky jsou rozděleny dle tradičního dělení na funkční a nefunkční. [9]

2.2.1 Funkční požadavky

Funkční požadavky definují, co má aplikace splňovat z pohledu funkcionality. Tyto požadavky slouží jako základ pro specifikaci uživatelských scénářů. [9]

F1 **Načtení hudebních souborů**
Systém umožní uživatelům načíst z definovaného lokálního adresáře hudební soubory v tradičních formátech, nejvíce preferovaný formát je MP3. Tyto soubory budou po načtení automaticky přidány do hudební knihovny.

F2 **Využití existujících metadat**
Systém bude při načítání hudebních souborů detekovat existující metadata zapsaná v souborech a použije je pro účely počáteční anotace entit v hudební knihovně. V případě, že metadata nebudou dostupná nebo čitelná, systém se pokusí sestavit základní metadata z cesty k hudebnímu souboru.

F3 **Kontrola konzistence hudební knihovny**
Systém umožní opakovaně kontrolovat definovaný lokální adresář a detekovat změny ve struktuře. Nové soubory budou přidány do knihovny, dále systém bude informovat o chybějících souborech a umožní je odstranit z knihovny.

F4 **Správa hudební knihovny**
Systém umožní ve formuláři editovat metadata základních entit, s důrazem na alba a skladby. Dále poskytne možnost entity přidávat a mazat.

F5 **Přehrávání hudby**
Systém bude poskytovat možnost přehrát hudební soubory v aplikačním rozhraní pomocí standardních ovládacích prvků.

F6 **Generování playlistu v reálném čase**
Systém umožní na základě množiny skladeb v hlavním playlistu a zadaných vztahů v reálném čase přidávat do playlistu automaticky další související skladby k přehrání. Tuto funkcionalitu bude možné během přehrávání libovolně zapínat a vypínat.

F7 **Tvorba a správa playlistů**
Systém umožní aktuální hlavní playlist uložit a následně nabídne možnost uložené playlisty spravovat.

F8 **Zobrazení alb v pohledech se seznamem**
Systém poskytne tradiční pohled na entity ve formě seznamů. Základní pohledy budou zahrnovat seznam umělců, formátovaný seznam alb ve formě jejich obrázků a detailnější pohled zahrnující základní metadata alba a seznam jeho skladeb.

F9 **Filtrování a řazení alb a umělců v pohledech se seznamem**
Systém umožní v seznamových pohledech filtrovat umělce podle jména a alba podle názvu a přiřazených štítků. Systém také poskytne možnost tyto entity seřadit podle různých atributů. Tento proces bude implementován tak, aby k jeho vykonání bylo potřeba co nejméně uživatelských kroků.

F10 **Zadání a odstranění podobnosti entit**
Systém umožní v interaktivním grafovém pohledu zadat podobnost entit. Zadání tohoto vztahu bude probíhat ve formě interakce s vizualizací grafu. Entity budou zobrazeny jako uzly grafu a vztahy jako hrany.

F11 **Propojení entit podle společné vlastnosti**
Systém umožní propojit základní entity podle zvolené společné vlastnosti. Propojení bude možné vytvořit v seznamovém i grafovém pohledu.

F12 **Přiřazení štítku entitám**
Systém poskytne možnost přidávat k albům a skladbám štítky s krátkým textovým označením.

F13 **Správa hudebních žánrů**
Systém umožní spravovat hudební žánry a přiřazovat je k jednotlivým albům. Systém umožní žánru přiřadit zvolenou barvu pro lepší vizuální rozlišení v systému a v grafových pohledech.

F14 **Agregace metadat**
Systém umožní při editaci alba stáhnout metadata z veřejného zdroje propojených dat a vybraná metadata těmito aktualizovat.

F15 **Interaktivní grafové pohledy**
Systém bude volitelně zobrazovat entity v grafových pohledech, které budou umožňovat interaktivně přesouvat a zvýrazňovat jednotlivé uzly. Systém vypočítá iniciální rozložení grafové struktury v pohledu tak, aby entity byly dobře viditelné.

2.2.2 Nefunkční požadavky

Nefunkční požadavky popisují další parametry, které má systém splňovat, a které nesouvisí přímo s funkcionalitou systému. Jsou to kvalitativní požadavky, které jsou také nedílnou součástí specifikace - např. použité technologie, výkon, spolehlivost, zajištění bezpečnosti. [9]

N1 **Nezávislé komponenty**
Systém bude rozdělen na backend a frontend část, které spolu budou komunikovat přes definované RESTful[8] API. Bude tak zajištěna separace datové a prezentační vrstvy a umožněna budoucí podpora jiných frontend klientů, např. pro mobilní zařízení.

N2 **Webová aplikace**
Aplikaci bude možné nasadit na webový server s podporou technologií Node.js[9] a Neo4j[10]. Rozhraní frontendové části bude dostupné ve webovém prohlížeči jako tzv. single-page aplikace[11].

N3 **Použité technologie**
Aplikace bude implementována v prostředí Node.js, na frontend část bude použit framework Vue.js[12].

N4 **Zveřejnění na GitHub**
Projekt bude zveřejněn na službě GitHub[13] jako veřený open-source projekt.

2.3 Formulace případů užití

Případy užití jsou formulovány na základě definovaných funkčních požadavků. Popsané případy užití popisují hlavní operace, které v aplikaci bude možné provádět. Vedle systémové role se v doméně vyskytuje pouze hlavní uživatelská role, z tohoto důvodu nejsou další aktéři systému popsáni. Případy užití, které jsou triviální nebo doplňkové (nesouvisejí přímo s hlavní funkcionalitou), jsem pro stručnost vynechal.

UC1 - Přidat sklady do playlistu

Popis

Uživatel vybere z nabídky skladby, které chce přehrát, a zařadí je na příslušné místo do hlavního playlistu.

[8] https://en.wikipedia.org/wiki/Representational_state_transfer
[9] https://nodejs.org/
[10] https://neo4j.com/
[11] https://en.wikipedia.org/wiki/Single-page_application
[12] https://vuejs.org/
[13] https://github.com

Předpoklady

Systém obsahuje skladby.

Hlavní tok

1. Případ užití začíná, když uživatel zobrazí pohled s výpisem alb a skladeb nebo playlistů.
2. Systém zobrazí seznam skladeb v podobě seznamu, kde je možné jednotlivé položky označovat.
3. Uživatel kliknutím na řádky skladeb vybere skladby. Systém umožní označit více skladeb u sebe tažením myši se zmáčknutým levým tlačítkem nebo pomocí držení tlačítka Ctrl a klikání.
4. Systém viditelně označí vybrané skladby a zobrazí na konci řádku tlačítko pro přetažení skladeb do playlistu.
5. Uživatel klikne a podrží tlačítko myši na zobrazeném tlačítku a posune ukazatel myši do oblasti hlavního playlistu.
6. Systém v hlavním playlistu zobrazí prvek, který bude indikovat, na jakou pozici se skladby přidávají. Pokud je playlist prázdný, systém umožní přidat skladby na jedinou pozici.
7. Uživatel zvolí pozici v hlavním playlistu a pustí tlačítko myši.
8. Systém přidá skladby do hlavního playlistu na zvolenou pozici. Skladby jsou nyní viditelně zobrazeny jako součást seznamu skladeb v hlavním playlistu.

Alternativní toky

1. Pro krok 5: Uživatel se rozhodne vybrané skladby nepřidat do playlistu.
   a) Uživatel kurzor myši přesune ven z oblasti hlavního playlistu a pustí tlačítko myši.
   b) Systém zruší přidání skladeb a viditelně informuje uživatele. Skladby nejsou přidány do playlistu.

UC2 - Přehrát skladbu

Popis

Uživatel zahájí přehrávání skladby v hlavním playlistu, systém skladbu přehraje.

Předpoklady

V hlavním playlistu jsou skladby.

Hlavní tok

1. Případ užití začíná, když uživatel chce přehrát nějakou skladbu v hlavním playlistu.
2. Uživatel provede dvoj-klik na vybranou skladbu v hlavním playlistu.
3. Systém nastaví a viditelně označí zvolenou skladbu jako aktuálně přehrávanou.
4. Systém zobrazí informace o aktuálně přehrávané skladbě.
5. Systém začne přehrávat skladbu.

UC3 - Editovat metadata alba

Popis

Uživatel edituje ve formuláři metadata náležící konkrétnímu albu.

Předpoklady

Systém obsahuje alba.

Hlavní tok

1. Případ užití začíná, když uživatel zobrazí pohled pro editaci alba.
2. Systém zobrazí formulář pro úpravu jednotlivých atributů. Tato pole jsou vyplněna stávajícími hodnotami.
3. Uživatel vybere ukazatelem pole, které chce editovat, a vloží novou hodnotu.
   a) V případě textové hodnoty (např. atribut *Album title*) přepíše původní hodnotu.
   b) V případě nabídky pro výběr jedné hodnoty (např. vztah *Main artist*) myší zvolí novou hodnotu nebo do nabídky napíše název entity. V takovém případě systém zobrazí v nabídce entity vyhovující dotazu a umožní entitu zvolit stejným způsobem.
   c) V případě nabídky pro výběr více hodnot (např. vztah *Other artists*) uživatel postupuje stejně, jako v předchozím bodě. Systém navíc v nabídce zobrazí všechny vybrané hodnoty a u každé poskytne možnost ji odebrat z výběru.

4. Uživatel klikne na tlačítko pro uložení.
5. Systém uloží provedené změny a informuje uživatele o úspěchu.
6. Systém aktualizuje zobrazená data editované entity ve všech zobrazených pohledech.

Alternativní toky

Chybové toky

1. Pro krok 4: Uživatel se rozhodne neuložit provedené změny.
   a) Uživatel klikne na tlačítko pro zavření aktuální nabídky.
   b) Systém zavře editační pohled a neuloží provedené změny.
2. Pro krok 5: Selhala validace vstupů z důvodu nepovolených hodnot.
   a) Systém zobrazí chybovou hlášku.
   b) Uživatel opraví hodnoty a znovu vykoná krok 4 hlavního scénáře.

UC4 - Přiřadit štítek k albu

Popis

Uživatel ke zvolenému albu přidá existující nebo nový štítek.

Předpoklady

Systém obsahuje alba.

Hlavní tok

1. Případ užití začíná, když má uživatel zobrazen nějaký z pohledů s detaily alba a chce k albu přiřadit štítek.
2. Systém v pohledu zobrazí již přiřazené štítky a také tlačítko pro editaci štítků.
3. Uživatel klikne na tlačítko pro editaci štítků.
4. Systém zobrazí ve vyskakovacím okně formulář s polem pro výběr ze štítků.
5. Uživatel píše do pole název požadovaného štítku a filtruje tak již přítomné štítky, zvolený štítek potvrdí stisknutím klávesy Enter. Tento krok uživatel opakuje pro všechny štítky, které chce přidat.

6. Systém zobrazí v nabídce všechny štítky, které bude entita mít přiřazeny po uložení.

7. Uživatel klikne na tlačítko pro uložení.

8. Systém přiřadí štítky k entitě a informuje uživatele o úspěšném provedení operace. Systém zavře vyskakovací okno. Systém zobrazí v původním výpisu aktuální štítky a aktualizuje je ve všech relevantních pohledech.

Alternativní toky

1. Pro krok 5: Uživatel chce zadat nový štítek.

   a) Uživatel napíše název nového štítku a stiskne Enter.

   b) Systém v nabídce zobrazí nový štítek mezi již přidanými štítky.

   c) Systém při ukládání uloží všechny nové štítky.

UC5 - Použít generování playlistu

Popis

Systém na základě skladeb v hlavním playlistu generuje další skladby k přehrání.

Předpoklady

Systém obsahuje skladby. Hlavní playlist obsahuje nějaké skladby.

Hlavní tok

1. Případ užití začíná, když uživatel chce spustit funkci automatického generování skladeb do hlavního playlistu.

2. Uživatel klikne na tlačítko pro spuštění funkcionality generování.

3. Systém informuje uživatele o zahájení generování.

4. Systém vygeneruje na základě množiny skladeb v hlavním playlistu nové skladby a přidá je na konec hlavního playlistu.

5. Systém opakuje předchozí krok vždy, když se aktuální pozice přehrávání v playlistu blíží konci, aby byla zachována kontinuita přehrávání.

6. Systém ukončí generování v momentě, kdy uživatel klikne na tlačítko pro ukončení generování.

Alternativní toky

1. Pro krok 2: Volba počátečních skladeb.
    a) Uživatel označí skladby v hlavním playlistu.
    b) Uživatel spustí generování stejným způsobem, jako v kroku 2.
    c) Systém pro inicializaci generování použije označené skladby.

Chybové toky

1. Alternativní ukončení generování v případě, že není možné přidat další skladby
    a) Systém zobrazí chybovou hlášku informující uživatele o faktu, že nelze vygenerovat další skladby.
    b) Systém ukončí funkci generování a umožní funkci spustit znovu s jiným počátečním nastavením.

UC6 - Zadat podobné entity

Popis

Uživatel v grafovém pohledu označí dvě entity jako podobné. Ilustrováno na entitě *Umělec*, analogické pro entitu *Album*.

Předpoklady

V systému jsou alespoň dva umělci.

Hlavní tok

1. Případ užití začíná, když uživatel zobrazí grafový pohled pro editaci podobnosti umělců.
2. Systém zobrazí grafovou vizualizaci, kde umělci jsou uzly a vztahy podobnosti jsou hrany.
3. Systém zobrazí tlačítko pro zobrazení nabídky s možností editace vztahů.
4. Uživatel klikne na zobrazené tlačítko.
5. Uživatel klikne na prvního umělce, pro kterého chce zadat podobnost a při držení tlačítka myši přetáhne ukazatel na druhého umělce, který je prvnímu podobný.
6. Systém zobrazí interaktivní hranu následující kurzor myši.

7. Systém po puštění tlačítka myši s kurzorem na uzlu druhého umělce vytvoří vazbu.

8. Systém zobrazí přidanou vazbu jako hranu v grafu a informuje uživatele o úspěšném vytvoření vazby.

Chybové toky

1. Pro krok 7: Uživatel pustí tlačítko myši dříve, než je proveden výběr druhého umělce.

    a) Systém zruší akci zadávání vztahu.

UC7 - Zkontrolovat nové skladby

Popis

Systém zkontroluje, zda v adresáři s hudebními soubory byly přidány nové soubory nebo jestli nějaké soubory chybí.

Předpoklady

V systému je zadána cesta k adresáři s hudebními soubory reprezentujícím hudební knihovnu.

Hlavní tok

1. Případ užití začíná, když si uživatel zobrazí pohled pro nastavení aplikace a otevře sekci pro kontrolu knihovny.

2. Uživatel klikne na tlačítko pro spuštění akce.

3. Systém začne skenovat adresář.

4. Systém informuje v pohledu uživatele o průběhu kontroly.

5. Systém přidá v momentu dokončení kontroly nové skladby do hudební knihovny a informuje uživatele o úspěšném vykonání akce spolu se statistikami obsahujícími počet přidaných skladeb.

Alternativní toky

1. Systém nalezne soubory, které jsou v knihovně a chybí v adresáři.

    a) Systém informuje uživatele o těchto souborech a nabídne možnost skladby odstranit z knihovny.

Chybové toky

1. Uživatel přeruší kontrolu kliknutím na tlačítko pro přerušení.

    a) Systém ukončí kontrolu a informuje uživatele o doposud přidaných a chybějících skladbách.

UC8 - Uložit aktuální playlist

Popis

Uživatel uloží aktuální hlavní playlist pro pozdější využití.

Předpoklady

Hlavní playlist obsahuje nějaké skladby.

Hlavní tok

1. Případ užití začíná, když uživatel klikne v hlavním playlistu na ikonku pro uložení aktuálního playlistu.
2. Systém zobrazí nabídku s polem pro zadání názvu nového playlistu.
3. Uživatel zadá název playlistu a potvrdí operaci tlačítkem pro uložení.
4. Systém uloží playlist a informuje uživatele o úspěšně provedené akci.

Chybové toky

1. Pro krok 3: Uživatel se rozhodne neuložit nový playlist.

    a) Uživatel klikne na tlačítko pro zrušení akce.

    b) Systém neuloží playlist a zobrazí původní pohled.

UC9 - Přehrát uložený playlist

Popis

Uživatel zvolí uložený playlist a zařadí jej do přehrávání.

Předpoklady

Systém obsahuje uložené playlisty.

Hlavní tok

1. Případ užití začíná, když uživatel zobrazí pohled s uloženými playlisty.
2. Systém zobrazí obrazovku se seznamem uložených playlistů.
3. Uživatel zvolí kliknutím playlist, který chce přehrát.
4. Systém zobrazí detaily uloženého playlistu spolu se seznamem skladeb.
5. Uživatel klikne na tlačítko pro přehrání zvoleného playlistu.
6. Systém odstraní obsah hlavního playlistu a přidá do něj zvolený playlist.
7. Uživatel postupuje jako v UC2.

Alternativní toky

1. Pro krok 5: Uživatel klikne na tlačítko "zařadit skladby na konec playlistu".
   a) Systém namísto odstranění obsahu hlavního playlistu zařadí skladby na jeho konec.
2. Pro krok 5: Uživatel chce přidat jen některé skladby.
   a) Uživatel označí vybrané skladby v seznamu a přetáhne je do hlavního playlistu podobně, jako v UC1.
   b) Systém přidá označené skladby na zvolenou pozici v hlavním playlistu.

UC10 - Přiřadit hudební žánr

Popis

Uživatel přiřadí k zvolenému albu existující hudební žánr.

Předpoklady

Systém obsahuje alespoň jedno album. Systém obsahuje alespoň jeden žánr.

Hlavní tok

1. Případ užití začíná, když uživatel zobrazí pohled pro editaci alba.
2. Systém v rámci editačního formuláře zobrazí pole pro editaci přiřazených žánrů.
3. Uživatel v poli vybere všechny žánry, které chce k albu přiřadit. Podporováno bude vyhledávání psaním názvu v poli.

4. Uživatel potvrdí akci tlačítkem pro uložení alba.
5. Systém informuje uživatele o úspěšném provedení akce.

Alternativní toky

1. Pro krok 3: Uživatel chce přiřadit žánr, který ještě neexistuje.
   a) Systém zobrazí u pole odkaz na obrazovku pro editaci žánrů.
   b) Uživatel klikne na zobrazený odkaz.
   c) Systém ve stejném okně zobrazí obrazovku pro editaci žánrů. Systém nabídne editovat existující žánry nebo vytvořit nový.
   d) Uživatel zvolí akci "vytvořit nový žánr".
   e) Systém zobrazí formulář s poli pro jednotlivé atributy žánru (pole musí zahrnovat název, popis, barvu žánru a nadřazený žánr).
   f) Uživatel vyplní pole a potvrdí tlačítkem pro uložení.
   g) Systém potvrdí uložení a zobrazí původní pohled pro editaci alba. Mezi žánry pro výběr bude i nový uložený žánr.
   h) Uživatel dále postupuje stejně, jako v hlavním scénáři.

UC11 - Použít k editaci metadat agregaci z internetu

Popis

Uživatel při anotování metadat alba zvolí stáhnout jejich návrh z internetové služby.

Předpoklady

Systém obsahuje nějaké album. Systém je připojen k internetu. Služba pro agregaci je dostupná.

Hlavní tok

1. Případ užití začíná, když uživatel pro zvolené album otevře editační pohled.
2. Systém zobrazí ve formuláři tlačítko pro agregaci metadat.
3. Uživatel klikne na zobrazené tlačítko.
4. Systém u jednotlivých polí nabídne navržené varianty.
5. Uživatel klikne na navržené varianty u polí, která chce změnit.
6. Systém nové hodnoty vloží do editačních polí.

7. Uživatel potvrdí uložení tlačítkem pro uložení.
8. Systém uloží album a informuje uživatele o úspěšně provedené akci.

Chybové toky

1. Pro krok 3: Nastala chyba při čtení metadat z internetového zdroje.
   a) Systém zobrazí chybovou hlášku.
   b) Uživatel může kliknout na tlačítko pro agregaci znovu a učinit tak další pokus nebo zadat vlastní metadata. Uživatel pokračuje stejně, jako v hlavním scénáři.
2. Pro krok 3: Agregační služba neposkytla žádná metadata.
   a) Systém zobrazí chybovou hlášku informující uživatele o absenci požadovaných metadat.
   b) Uživatel zadá vlastní metadata a pokračuje stejně, jako v hlavním scénáři.

UC12 - Zobrazit grafový pohled pro umělce a alba

Popis

Uživatel využívá grafový pohled pro zobrazení aktuálně přehrávaného umělce a jeho alb.

Předpoklady

Systém obsahuje nějaké album, které má hlavního umělce. V playlistu je skladba z tohoto alba.

Hlavní tok

1. Případ užití začíná, když uživatel navštíví daný grafový pohled a iniciuje přehrávání skladby v hlavním playlistu.
2. Systém zobrazí v grafovém pohledu jako hlavní uzel umělce, který je přiřazen k hrající skladbě přes album. Další uzly budou alba, která má umělec dále přiřazena. Poslední rovina uzlů budou další umělci, kteří se na albu zúčastnili a tato vazba je v systému. Tyto vztahy budou znázorněny vazbami.

2.4 Mapování případů užití na požadavky

V této sekci uvádím přehlednou tabulku mapování jednotlivých požadavků na příslušné případy užití. Požadavky F8 a F9 nejsou mapovány na žádný Use Case, protože tyto případy užití by byly triviálním rozepsáním příslušných požadavků, a z toho důvodu je neuvádím.

UC	1	2	3	4	5	6	7	8	9	10	11	12
F1							X					
F2							X					
F3							X					
F4			X	X				X		X		
F5	X	X							X			
F6					X							
F7								X	X			
F10						X						
F11				X		X						
F12				X								
F13										X		
F14											X	
F15						X						X

Tabulka 2.1: Mapování případů užití na funkční požadavky

Kapitola 3

Návrh řešení

Na základě analýzy problému jsem vypracoval návrh aplikace. V kapitole jsou popsány jednotlivé entity, zvolená architektura aplikace, vybrané konkrétní problémy, které jsou stěžejní pro řešení, a na vytvořených prototypech je ilustrována cílová podoba aplikace. Tento návrh je vstupem pro následnou implementaci.

3.1 Popis entit a vazeb mezi nimi

Z popisu domény a uživatelských požadavků jsem sestavil datový model aplikace. Přestože bývá zvykem řešit konkrétní zajištění datové vrstvy aplikace až po navržení konceptuálního modelu, již v průběhu analýzy bylo zřejmé, že nejlepší budoucí reprezentací dat bude grafová struktura (konkrétní řešení je popsáno v následující kapitole). Aplikace bude často využívat procházení vztahů mezi entitami a existují databázové stroje[14], které jsou pro tyto operace přímo optimalizovány. Tento fakt ovlivnil i samotný návrh datové vrstvy. Na uvedené entity a vazby je tedy nahlíženo z pohledu grafové reprezentace.

3.1.1 Entity

Následuje výpis jednotlivých entit v systému spolu s jejich atributy. Povinné atributy jsou označeny hvězdičkou na konci názvu.

Entity se částečně překrývají se základními entitami v sekci 1.1.1, pro popis jejich významu tedy čtenáře odkazuji do zmíněné části textu.

Artist

name* Textová hodnota jména umělce, základní zobrazovaná podoba

[14] Např. Neo4j - `https://neo4j.com`

sortName Textová hodnota jména umělce určená pouze pro účely řazení v seznamech (např. „Příjmení, J.“)

bio Textový popis biografie umělce

Z analýzy problému plyne, že systém by měl rozlišovat různé typy jedinců, kteří se podílejí na albu. Kromě typického umělce (hlavního nebo vedlejších) to může být průmyslem definované pole „Album Artist“, jehož významy se v interpretacích liší. Dále některé přehrávače přímo podporují tag „Composer“ (skladatel), případně „Publisher“ (vydavatel) a další. Tuto problematiku vyřeší jednotná entita „Artist“, k níž později bude možné dodefinovat konkrétní typy, které budou zahrnovat zmíněné entity.

Album

title* Textová hodnota názvu alba

inInbox* Binární hodnota určující zda se album nachází v sekci „inbox“, nastaveno defaultně na *true*

year Číselná hodnota popisující rok vydání alba

comments Textové pole pro účely uživatelských komentářů k albu

Sekce „inbox“ zajišťuje jednoduchý systém pro rozlišení nově přidaných alb. Uživatel má možnost v systému alba přesunout do hudební knihovny, např. v momentu, kdy je spokojen s vyplněním metadat (ale není to podmínkou, metadata lze upravovat i posléze).

Track

title* Textová hodnota názvu skladby

filePath* Textová hodnota cesty k souboru v systému souborů

trackNr Číselná hodnota udávající pořadové číslo skladby v albu

diskNr Číselná hodnota udávající pořadové číslo nosiče, na kterém se skladba původně vyskytuje

lyrics Textová hodnota se slovy, která se zpívají ve skladbě

comments Textové pole pro účely uživatelských komentářů ke skladbě

playCount Číselná hodnota udávající počet přehrání skladby v systému

Genre

name* Textová hodnota názvu hudebního žánru

description Textová hodnota se slovním popisem žánru

color Textová hodnota obsahující reprezentaci barvy pro žánr (předpokládá se hexadecimální zápis), barva bude sloužit pro vizuální odlišení žánrů v aplikaci

Label

name* Textová hodnota názvu štítku

description Textová hodnota se slovním popisem štítku

Playlist

name* Textová hodnota názvu playlistu

3.1.2 Vazby

Následuje výčet vazeb, které se v systému budou vyskytovat. Některé entity mohou mít různé vazby stejného typu, které se však liší významem. Při implementaci budou vazby rozlišitelné pomocí nastaveného typu.

3.1.2.1 Pevné vazby

V této sekci jsou popsány vazby, které určují základní vztahy entit. Např. skladba náleží do alba, album má žánr, apod. Nejedná se tedy o vazby, které vytváří uživatel pro potřeby dalšího propojení entit.

Artist-Album Album bylo vydáno pod umělcem

a) Parametr **type** - identifikace vztahu umělce k albu - např. hlavní umělec, vedlejší umělci, skladatel, ...

Album-Track Skladba náleží do alba

Album-Genre Album náleží do nějakého žánru - tato vazba je na pomezí obou kategorií, záleží, jak uživatel vnímá popisovanou problematiku přiřazování žánrů

Genre-Genre Žánr má nadřazený žánr

3.1.2.2 Uživatelské vazby

Uživatelské vazby jsou takové, které uživatel manuálně vytváří za účelem sdružení entit podle nějakého vlastního kritéria.

Artist-Artist Podobnost umělců

Album-Album Podobnost alb

Album-Label Přiřazení štítku k albu

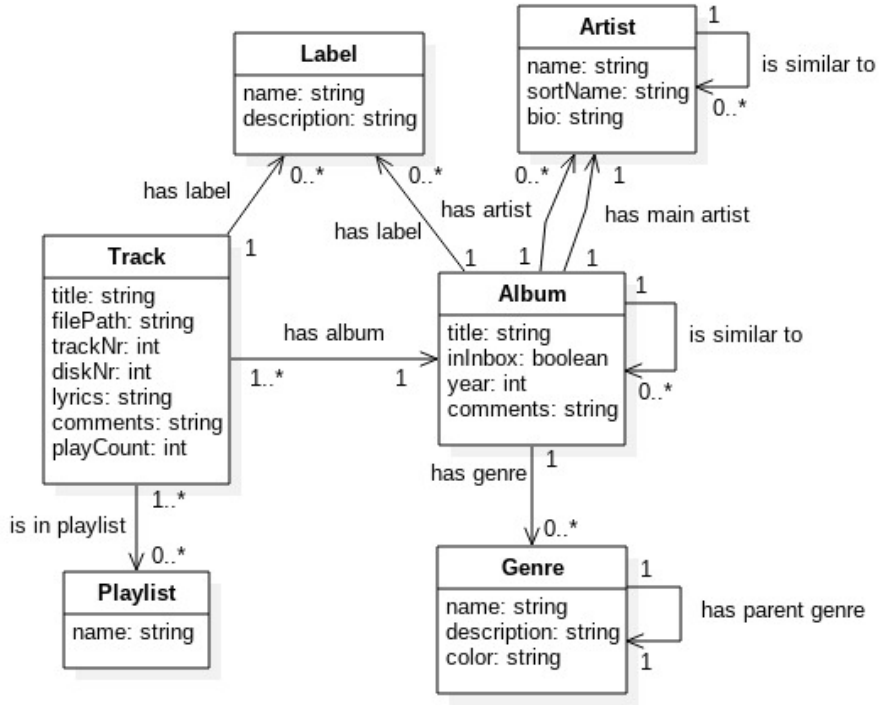
Track-Label Přiřazení štítku ke skladbě

Playlist-Track Zařazení skladby do playlistu

a) Parametr **position** - pozice v playlistu

3.2 Logický model

Následuje model popisující strukturu aplikace.



Obrázek 3.1: Logický model

3.3 Popis architektury

Z nefunkčního požadavku N1 plyne potřeba rozdělení aplikace na dvě oddělené části, které budou komunikovat přes RESTful API. Použití webových technologií umožňuje přirozenou implementaci takového řešení.

Úkolem frontend části aplikace bude poskytovat uživateli vizuální rozhraní, pomocí kterého bude aplikaci ovládat.

Úkolem backend části aplikace bude zajistit správu dat a vytvořit reprezentace zdrojů, které následně budou využity ve frontend části. Tato část bude spjata se souborovým systémem, ve kterém se nalézá adresář s hudebními soubory.

Logika aplikace bude z důvodu komplexity řešení rozdělena mezi obě části, které však budou omezeny právě definovaným API, které je tedy pojícím prvkem. Názvy „frontend“ a „backend“ jsou v tomto případě synonyma pro „klient“ a „server“. Tuto terminologii volím z důvodu názornosti koncepce jednotlivých částí.

Výhodou tohoto řešení je i separace zodpovědností. Jednotlivé části aplikace budou mít jasně definované funkce, které jsou na sobě málo závislé. To výrazně usnadňuje a zpřehledňuje vývoj celé aplikace. Naskýtá se např. možnost obě části udržovat v oddělených repozitářích, takže je jednodušší koordinovat změny v projektech - toho bude využito při publikaci projektu na internetu. Část komunity se může soustředit na backend část, zatímco jiná část na frontend.

3.3.1 Definice RESTful API

Součástí specifikace datového a komunikačního modelu je formulace API, pomocí kterého může frontend a backend část komunikovat. API musí být korektně definováno zejména v případě, kdy může nastat možnost existence více druhů komponent, zejména frontend klientů. Cílem práce je vytvořit jednu aplikaci sestávající z obou částí, nicméně nelze vyloučit, že v budoucnu vzniknou další klienti - např. klient pro mobilní zařízení nebo alternativní klient pro PC.

Použité metody budou vracet kód 500 - Internal Server Error v případě, že dojde k chybě při vykonávání dotazu. Tento kód je společný, v popisu uvádím specifické kódy.

Na odkazu `https://app.swaggerhub.com/api/krystofspl/gmlbackend/1.0.0` je dostupná aktuální verze API s podrobným popisem jednotlivých metod a reprezentací zdrojů. Tento popis bude i součástí dokumentace GitHub projektu a členové opensource komunity budou mít možnost se na API podílet. API je vytvořeno pomocí frameworku Swagger, kde je možné pomocí jazyku YAML zapsat vlastnosti API a následně jej exportovat do různých formátů.

Pro přehlednost uvádím seznam navržených zdrojů a použitých metod. Podrobný popis API je k dispozici na zmíněném odkazu.

Identifikátor	Použité metody
/genres	GET, POST
/genres/{id}	GET, PATCH, DELETE
/artists	GET
/artists/{id}	GET
/tracks	GET
/tracks/{id}	PATCH, DELETE
/tracks/{id}/file.mp3	GET
/rescan	GET, PUT
/graphs/artists-albums-graph	GET
/graphs/genres-albums-graph	GET
/graphs/artists-artists-graph	GET
/graphs/multi-graph	GET
/albums	GET
/albums/{id}	GET, PATCH, DELETE
/album-arts/{id}	GET
/playlists	GET, POST
/playlists/{id}	GET, PATCH, DELETE
/labels	GET, POST
/labels/{id}	GET, PATCH, DELETE
/relationships/album-similarity	GET, POST
/relationships/album-similarity/{start}/{end}	DELETE
/relationships/artist-similarity	GET, POST
/relationships/artist-similarity/{start}/{end}	DELETE
/relationships/labels-parent	GET, POST
/relationships/labels-parent/{start}/{end}	DELETE

Tabulka 3.1: Souhrn navržených zdrojů RESTful API

3.4 Popis vybraných složitějších problémů

Při řešení jednotlivých částí aplikace se vyskytly některé zajímavé problémy související s různými částmi funkcionality. Popisy problémů obsahují diskusi možných řešení i finální zvolenou podobu.

3.4.1 Modelování podobnosti hudebních entit

Při navrhování správy uživatelských vazeb v systému jsem narazil na některé problémy (viz dále), k jejichž řešení by mohly přispět algoritmy využívající automatickou klasifikaci podobnosti hudebních skladeb.

Modelování podobnosti v hudební doméně je v současnosti aktivně zkoumanou problematikou. Výzkumy naznačují, že se jedná o velmi komplexní a obtížný problém. [10] Na formulaci podobnosti lze nahlížet z pohledu sběru a analýzy velkého množství uživatelských dat nebo z pohledu analýzy samotného hudebního obsahu. Metody, které se zabývají doporučováním, tradičně zkoumají chování uživatelů v globálním měřítku a nejsou pro tuto aplikaci příliš zajímavé, protože aplikace je primárně zamýšlena pro účely správy lokálních dat pro jednoho uživatele. Dále byly navrženy mnohé metody výpočtu podobnosti skladeb, které často využívají extrahované hudební deskriptory, které mohou poskytnout vyhodnocení akustické podobnosti. Navržené metody nezahrnují aspekt sémantického smyslu podobnosti a jsou cíleny zejména na automatickou klasifikaci typů hudby, např. příslušnost do hudebního žánru. Různé výzkumy z poslední doby se snaží kombinací analýzy hudebního obsahu a dolování kulturních metadat z webu oba pohledy kombinovat, problém však stále do značné míry čeká na uspokojivé vyřešení. [10]

Tato práce si neklade za cíl navrhnout novou metodu modelování podobnosti, inspirace aktuálními experimenty však může do budoucna poskytnout vhodný směr postupu pro řešení některých problémů (absence vazeb při prvním spuštění - sekce 3.4.3, pomocné kritérium pro generování playlistu na základě podobnosti - sekce 3.4.2 a další).

Metody pro modelování podobnosti na základě vybraných extrahovaných vlastností (tradičně z deskriptorů MPEG7 nebo MFCC[15], které jsou dobře dostupné) používají různé podobnostní míry (Euklidovská vzdálenost, Earth Mover's Distance a další). Další používanou metodou, která je intenzivněji prozkoumávána zejména v posledních několika letech, jsou různé varianty strojového učení. [10]

Práce [11] popisuje automatickou klasifikaci skladeb do několika základních hudebních žánrů a podžánrů, k čemuž využívá neuronovou síť pro lokální vztahy mezi extrahovanými vlastnostmi a klasifikátor k-nearest neighbours pro obecnější vztahy. Neuronová síť vyžaduje větší čas pro natrénování a je tedy použita pouze pro složitější vztahy, kde svoji pomalost kompenzuje vysokou účinností. Výběr samotných vlastností pro extrakci je proveden pomocí genetických algoritmů. Autoři uvádí 98% úspěšnost klasifikace pro 3 nejzákladnější žánry a 90% úspěšnost pro 9 podžánrů při použití trénovací sady 950 charakteristických skladeb.

Výsledky zmíněné práce [11] by mohly být aplikovány na navrhovanou aplikaci. Uživatel by měl možnost nadefinovat několik základních charakteristických kategorií (pravděpodobně hudební žánr, ale při zachování požadavku charakterističnosti jsou možná i jiná kritéria, např. asociovaná nálada), pro které by vybral několik charakteristických alb nebo větší množství jednotlivých skladeb. U průměrného alba lze očekávat cca 7 skladeb, kombinací několika základních alb by tedy nemělo být pro uživatele složité sestavit dostatečný

[15] Mel-frequency cepstrum coefficients

počet skladeb pro trénování sítě. Pro každou zvolenou kategorii by tedy v systému byl vytvořen a natrénován klasifikátor, poté by bylo možné ohodnotit všechny skladby v knihovně za relativně krátkou dobu.

V případě algoritmu generování playlistu by výsledky klasifikace (a z nich plynoucí náležitost do jisté skupiny) mohly být dalším z kritérií pro výběr následných skladeb. Klasifikace by také mohla poskytnout základní počáteční seskupení alb a odlišení těchto skupin. Akce uživatele v podobě manuálního výběru charakteristických alb a kategorií by stále byla nutná, nicméně se jedná o významně časově méně náročnou operaci, než manuální zadání mnoha vazeb.

3.4.2 Generování playlistu v reálném čase

Důležitou částí aplikace bude funkcionalita pro generování seznamu skladeb v reálném čase. Navrhování dalších skladeb z knihovny bude založeno na procházení grafové struktury. Prioritně budou využity manuálně zadané uživatelské vazby, dále vazby pevné.

Volba metody

Při navrhování této funkcionality jsem zvažoval dvě možné varianty řešení. První varianta, kterou jsem zvolil pro finální návrh, využívá procházení zadaných vazeb a je popsána v následující sekci. Další variantou je metoda, která provede extrakci vlastností ze zvukového obsahu skladeb a použije je pro generování podobnosti. Princip metody je popsán výše.

Metoda procházení grafu pro ideální funkčnost vyžaduje, aby v systému byl vyplněn významný počet vazeb. Čím více tedy uživatel „investuje“ do vytvoření dat v systému, tím bude metoda podávat lepší a zajímavější výsledky.

Metodu analýzy hudebního obsahu jsem se rozhodl v této fázi z časových důvodů neimplementovat, dalším důvodem je nejistota úspěchu (zejména oproti zvolené metodě) - výzkumy sice nabízí uspokojivé výsledky klasifikace, záleží však i na použitých technologiích a obecných podmínkách. V budoucnu by však implementace této metody mohla dále pomoci kvalitě výsledků algoritmu.

Popis vytvořeného algoritmu

Vstupem algoritmu je množina počátečních skladeb (tzv. „seed tracks“), což může být jedna nebo více skladeb, na jejichž základě se získávají další skladby. Tento výběr je proveden uživatelem ve frontend části aplikace vybráním skladeb v hlavním playlistu nebo aplikace automaticky vybere poslední skladbu v playlistu. Předpokladem tedy je neprázdný hlavní playlist. Identifikátory skladeb se odešlou na backend část, kde se přidají do množiny počátečních skladeb a do množiny již použitých skladeb, aby nedocházelo k opakovanému vrácení stejných skladeb.

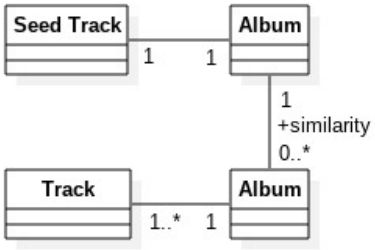
Algoritmus na základě získané množiny počátečních skladeb vykoná databázový dotaz pro všechny tyto skladby a všechny definované cesty (viz následující sekce). Všechny takto získané skladby se přidají do poolu vygenerovaných doporučení. Definovaná priorita jednotlivých cest určuje, kolikrát budou tyto skladby přidány. Tento multiplikátor má za cíl prioritizovat uživatelské vazby (které jsou z definice nejméně obecné, a mají z pohledu uživatele největší váhu) a poskytovat tedy větší množství skladeb dle uživatelské preference. Hodnoty jsou diskutovány v následující sekci.

Pool vygenerovaných doporučení se poté náhodně seřadí a jsou náhodně vybrány 3 unikátní skladby, které nejsou na seznamu již použitých skladeb. Skladby jsou zařazeny na seznam použitých a poté odeslány zpět jako odpověď. Frontend část aplikace tyto skladby přidá na konec playlistu nebo informuje uživatele, když nebylo nic vygenerováno. Frontend část opakuje tento dotaz vždy, když je zapnuté generování, a vzdálenost aktuální pozice přehrávání v hlavním playlistu od jeho konce je menší, než 3 skladby. Tento počet je zvolen jako kompromis mezi příliš častým zasíláním dotazů a zahlcením playlistu v rámci jedné operace generování. Uživatel má možnost mezi jednotlivými generačními cykly změnit obsah playlistu i pozice skladeb, generování tedy takto poskytne výsledky relevantní pro aktuální stav přehrávání.

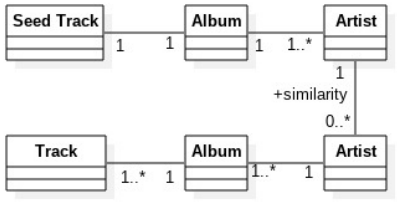
Algoritmus tedy doporučuje nové skladby na základě poslední skladby v playlistu, čímž se snaží vytvořit playlist, který sestává z postupně podobných skladeb.

Použité cesty

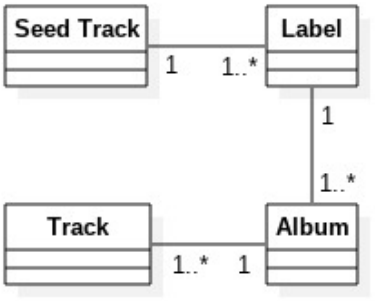
Následuje výčet vybraných cest v podobě symbolických entit a vazeb. Průchod začíná jednou skladbou „Seed Track“ a končí u množiny skladeb, zde reprezentováno jako entita „Track“. Každý dotaz může mít více počátečních skladeb, může tedy být vráceno více výsledků v závislosti na nastavení počátečních podmínek.



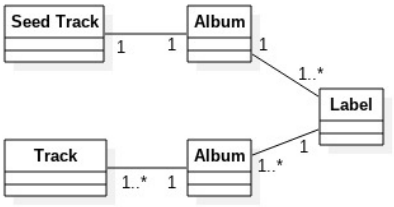
Obrázek 3.2: Generování playlistu - cesta 1



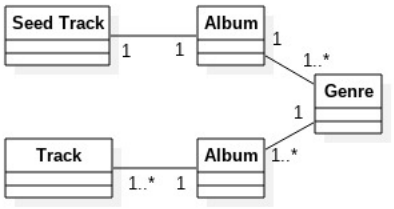
Obrázek 3.3: Generování playlistu - cesta 2



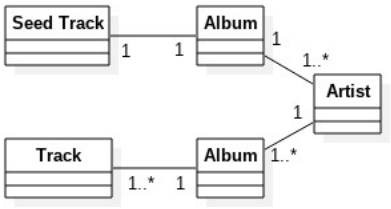
Obrázek 3.4: Generování playlistu - cesta 3



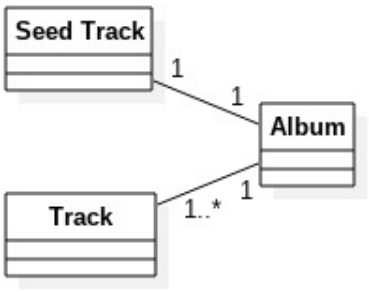
Obrázek 3.5: Generování playlistu - cesta 4



Obrázek 3.6: Generování playlistu - cesta 5



Obrázek 3.7: Generování playlistu - cesta 6



Obrázek 3.8: Generování playlistu - cesta 7

Jednotlivým cestám jsou přiřazeny priority, jimž bude odpovídat multiplikátor pro přidání do poolu. Nejnižší hodnota odpovídá nejdůležitější cestě.

Aktuální nastavení těchto priorit je založeno na jednoduchém odhadu frekvence výskytu příslušných výsledků pro danou cestu spolu s prioritizací konkrétnějších kritérií (např. jedna podobnostní vazba pro umělce zahrnuje 2 entity, zatímco příslušnost do žánru může zahrnovat i stovky entit). Na základě zpětné vazby od uživatelů bude pravděpodobně nutné tyto koeficienty upravit, případně implementovat adaptivní algoritmus, který je dynamicky určí podle velikosti knihovny a množství zadaných vazeb.

Cesta	**Priorita**
1	1
2	2
3	3
4	3
5	3
6	4
7	4

Tabulka 3.2: Priority cest pro generování playlistu

3.4.3 Počáteční vytvoření vazeb

Další problém, který jsem řešil při navrhování aplikace, je absence vazeb ve chvíli, kdy uživatel poprvé začne používat aplikaci. Uživatel má k dispozici grafové pohledy, ale jednotlivé uzly nejsou nijak propojeny, takže tyto pohledy neplní účel.

Aplikace je zamýšlena především jako nástroj, který umožňuje vytvářet vazby podle konkrétních potřeb uživatelů, v kontrastu s tradičními přehrávači nebo službami, které tyto vazby vytváří podle různých algoritmů na základě velkého množství dat (což je popsáno v úvodu kapitoly 1). Argumentem pro uspokojivost řešení pouze manuálního zadávání vazeb tedy může být zvolená cílová skupina hudebních nadšenců (sekce 1.4), kteří nutný čas a energii pro zadání vazeb investují výměnou za přidanou hodnotu aplikace. To však již nelze očekávat od širší skupiny uživatelů, protože tento časově nákladný proces by je mohl odradit od používání aplikace. Z těchto důvodů zde formuluji různá řešení problému navržení či vyplnění počátečních dat vazeb.

3.4.3.1 Pevné vazby

Doplnění pevných vazeb (3.1.2.1) je ekvivalentní operaci tagování z veřejné databáze tagů a bude řešeno v rámci aplikace jako rozšiřující operace pro editaci entit. Počáteční inicializace velké knihovny by tedy sestávala z automatické aplikace této operace na všechna alba. Automatické tagování z veřejné databáze je však problematickou operací v případech, kdy nejsou v souborech dostupná žádná původní metadata, nebo se jedná o málo populární album,

které je v databázích popsáno nedostatečně nebo vůbec. Nelze se tedy spolehnout na správnost této operace a je nutné nová alba dát uživateli ke kontrole a případné opravě.

Lepší výsledky by měla podat agregace dat pomocí služby, která podporuje analyzování a rozpoznání úseků audia a vrací uživateli zpět příslušná metadata z databáze. Tuto funkcionalitu poskytuje v současné době zejména služba EchoNest [16] nebo ACRCloud [17], problémem je však komerční povaha těchto API. Některé služby poskytují přístup ke svému API přes API klíč, o který by však každý uživatel musel žádat, což nepovažuji za vhodné řešení. Navrhovaná aplikace je open-source, využití komerčních plánů tedy (alespoň v této fázi) také nepřipadá v úvahu. Další nevýhodou je omezení služeb v maximálním počtu volání API za daný časový úsek - analýza tisíců skladeb by byla velmi časově nákladná. Tuto variantu tedy zmiňuji jako možné budoucí řešení v závislosti na vývoji zmiňovaných služeb, v tuto chvíli však není vhodné.

Pro účely klasifikace základních žánrů by dále mohl být použit algoritmus popsaný v sekci 3.4.1. Jeho přesný návrh a implementace je však spíše možnou otázkou budoucího vývoje projektu, řešení je nad rámec této práce.

Při vývoji aplikace tedy s přihlédnutím k výše zjištěným možnostem zvolím přístup automatického čtení metadat přímo z hudebních souborů spolu s možností manuální úpravy pro alba, zejména tam, kde jsou metadata chybná nebo chybějící. Dále bude možné využít na stejném místě automatické doporučení metadat pomocí agregační služby.

3.4.3.2 Uživatelské vazby

Doplnění uživatelských vazeb (3.1.2.2) je komplikovanější problém. Definice podobnosti entit (např. skupiny umělců) je ve velké míře subjektivní a aplikace toto musí kvůli vytyčeným cílům respektovat. Zejména oblast přiřazování štítků (za účelem seskupení entit podle nějakého, obecně abstraktního, kritéria) má potenciál být zcela subjektivní a integrace jinde získaných vztahů by v mnoha případech neměla smysl. Existující informace o podobnosti entit však lze využít k doporučení těchto vazeb v rámci aplikace. Uživatelé mohou mít různý názor na využití těchto dat, proto za ideální řešení považuji obecně u diskutovaných metod umožnit uživateli zvolit, zda tyto vazby chce v systému automaticky vytvořit, pouze doporučit nebo vůbec nevyužít.

Metody uvedené v této sekci popisuji z důvodu podání přehledu možných řešení z pohledu implementované aplikace. Některé metody v této fázi nebudou implementovány, protože jejich konkrétní návrh a implementace vyžadují další rozsáhlý průzkum, který není cílem této práce. Práce se snaží vytvořit platformu pro umožnění implementace podobných algoritmů, tento popis tedy může být impulsem pro budoucí komunitní vývoj aplikace.

[16] `https://the.echonest.com`
[17] `https://www.acrcloud.com/`

Samotné získání dat lze řešit podobně, jako u pevných vazeb - komunikací s externí službou, která tato data poskytuje. Vyskytuje se zde stejný problém s komerčními API, jako u pevných vazeb. Namísto získávání službou předpřipravených dat tohoto typu budu v této fázi volit agregaci z veřejného zdroje dat, konkrétně službu DBPedia[18]. Jedná se o službu, která se v rámci komunitní snahy snaží extrahovat metadata z Wikipedie, kde se nachází řada volně dostupných dat i pro účely doporučení podobných umělců. Tato funkcionalita je podrobněji popsána v sekci 3.4.4.

Dalším možným řešením by bylo vytvoření externího serveru, se kterým by jednotlivé instance aplikací komunikovaly, a který by agregoval některá vazební data. Smysluplná by byla zejména agregace dat uživatelsky zadané podobnosti. Server by pak zastával funkci služeb, které jsou popsány výše, a aplikace by v tomto ohledu nebyla závislá na externí službě. Implementace tohoto řešení by v základu měla být poměrně jednoduchá, je však nutné se detailněji zaměřit na výpočet vracených dat, protože nelze obecně očekávat, že získané vazby od jednotlivých uživatelů budou automaticky přispívat k obecně uspokojivému výsledku. Jistým řešením by bylo ohodnocení vazeb na serveru podle počtu přijatých variant od uživatelů. Uživatel aplikace by si poté mohl např. nastavit hranici minimálního počtu jiných uživatelů, kteří se shodnou na vazbě. Bude také nutné do řešení zahrnout nějakou formu předzpracování nebo mapování zasílaných metadat - častým jevem u metadat toho typu je jejich různá podoba pro stejnou entitu (např. název umělce).

Poslední technika, kterou pro úplnost zmiňuji, je sbírání dat o uživatelově chování přímo v aplikaci. Nabízí se sledování posloupností skladeb v hlavním playlistu. Podobnostní vazby by byly automaticky přidávány na základě přidaných skladeb. Toto kritérium však nemusí být spolehlivé, protože se u uživatelů obecně liší preference poslechu (resp. posloupností skladeb). Mnoho posluchačů také do playlistu přidává celá alba, získávání podobnosti by proto bylo vhodnější implementovat na úrovni celých alb nebo umělců získaných z přehrávaných skladeb. Z těchto důvodů by bylo vhodné provést další průzkum zabývající se přínosností tohoto řešení.

3.4.4 Agregace metadat

3.4.4.1 Pevné vazby

Pro získání základních metadat bude použita již popsaná služba MusicBrainz. Služba poskytuje rozsáhlou databázi metadat vyplněných uživateli a je zdarma. V dokumentaci [12] je uvedena podpora SPARQL [19] dotazů, které jsem měl původně v plánu využít. Toto rozhraní však bylo po celou dobu psaní práce nedostupné.

[18] `http://wiki.dbpedia.org/`

[19] SPARQL Protocol and RDF Query Language

Služba dále poskytuje API pro přístup k datům ve formátech XML a JSON. Vzhledem ke zvoleným technologiím jsem se tedy rozhodl využít JSON API.

Aplikace nabídne uživateli v editačním formuláři alba možnost získat metadata na základě názvu umělce a názvu alba (odpovídá UC11 - sekce 2.3). Iniciální verze těchto hodnot pochází z přečtených metadat (nebo cesty k souboru), nemusí však být přítomny. Proto aplikace umožní uživateli zadat manuálně tyto názvy, pro které bude znovu vykonán dotaz.

Zmíněné API má vyhledávací část, která bude použita pro účely získání požadovaných entit. Podrobnější informace o entitách lze získat v hlavní části API po získání unikátního ID dané entity. [13]

Následující dotaz vrátí všechna alba vyhovující zadanému názvu umělce a alba.

```
http://musicbrainz.org/ws/2/release/
?query=release:BOSH AND artist:Cleft
&fmt=json
```

Výpis 3.1: MusicBrainz dotaz 1

V následující odpovědi jsou znázorněny údaje, které budou přímo použity pro anotaci. Ostatní údaje mohou být využity v budoucnu jako rozšíření této funkcionality. Výpis je zkrácen o části, které nejsou relevantní pro aktuální formu funkcionality.

```
{
  "count": 2,
  "created": "2017-04-04T16:28:12.26Z",
  "offset": 0,
  "releases": [
    {
      "artist-credit": [
        {
          "artist": {
            "disambiguation": "UK two piece math-rock band",
            "id": "3ba1a486-cb40-4faf-8ee7-52410d710b72",
            "name": "Cleft",
            "sort-name": "Cleft"
          }
        }
      ],
      "barcode": "5055486935075",
      "country": "XW",
      "date": "2014-02-10",
      "id": "1689ad44-8264-4099-b724-2163498c8d3c",
      "label-info": [
        {
          "catalog-number": "MFR03",
          "label": ...
```

```
25          }
26        ],
27        "media": [
28          {
29            "disc-count": 0,
30            "format": "12\" Vinyl",
31            "track-count": 10
32          }
33        ],
34        "packaging": "Other",
35        "release-events": ...,
36        "release-group": {
37          "id": "71b7a234-ea03-4095-9d32-f54d1484a18a",
38          "primary-type": "Album"
39        },
40        "score": "100",
41        "status": "Official",
42        "text-representation": ...,
43         "title": "BOSH!",
44        "track-count": 10
45      },
46      ...
47    ]
48  }
```

Získaná alba („releases“) budou vypsána uživateli, který bude moci vybrat to, které odpovídá jeho dotazu. Pro rozlišení umělců se stejným názvem je k dispozici pole „disambiguation“. Výběr povede na následující dotaz, kde se odešle vybrané UUID.

```
http://musicbrainz.org/ws/2/release/1689ad44-8264-4099-
    b724-2163498c8d3c
?inc=recordings
&fmt=json
```

Výpis 3.2: MusicBrainz dotaz 2

Následuje odpověď s dalšími podrobnostmi o albu a skladbách. Obdobně jako u variant alba jsou zde prezentovány různé formy vydání alba (mohou se lišit např. v počtu i složení skladeb, jak je popsáno v první kapitole), aplikace umožní výběr.

```
1  {
2    "asin": null,
3    "barcode": "5055486935075",
4    "country": "XW",
5    ...
6    "date": "2014-02-10",
7    "disambiguation": "",
8    "id": "1689ad44-8264-4099-b724-2163498c8d3c",
```

```
  "media": [
    {
      "format": "12\" Vinyl",
      "format-id": "3e9080b0-5e6c-34ab-bd15-f526b6306a64",
      "position": 1,
      "title": "",
      "track-count": 10,
      "track-offset": 0,
      "tracks": [
        {
          "id": "632f05b3-78a0-4154-b544-eb9041201687",
          "length": 256000,
          "number": "1",
          "title": "12 Second Panda" ,
          "recording": ...
        },
        ...
      ]
    }
  ],
  ...
  "title": "BOSH!"
}
```

Na základě těchto dat tedy aplikace navrhne nová metadata v editačním formuláři. Uživatel bude mít možnost se rozhodnout, zda je využít.

3.4.4.2 Návrhy uživatelských vazeb

Jak bylo popsáno výše, podmnožinu uživatelských vazeb lze alespoň u populárnějších umělců a alb získat ze služby DBPedia pomocí vhodného SPARQL dotazu. Služba agreguje data z Wikipedie, která obsahuje v této doméně poměrně uspokojivé množství dat. Níže popsaný dotaz jsem otestoval na desítkách umělců z různých žánrů, u kterých jsem očekával malý objem doporučených umělců a skupin. U většiny vybraných umělců dotaz vrátil alespoň několik doporučení, metoda je tedy pro tyto účely vhodná. Postupným průzkumem používaných entit a vlastností jsem sestavil SPARQL dotaz [14], který vrátí seznam relevantních umělců a skupin pro zadaný název umělce nebo skladatele. Název musí být přesný, snažil jsem se využít funkcionalitu filtrování regulárními výrazy, systém však dotaz vyhodnotil jako příliš časově náročný, z tohoto důvodu je zvolena aktuální forma. SPARQL endpoint nabízí výstup i ve formátu JSON, což je ideální formát pro účely této aplikace.

```
  PREFIX dbo: <http://dbpedia.org/ontology/>
  PREFIX schema: <http://schema.org/>
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  PREFIX dbpedia2: <http://dbpedia.org/property/>

```

```
6   SELECT (SAMPLE(?artistN) as ?aN) (SAMPLE(?relatedArtistN
       ) as ?raN)
7   WHERE {
8   ?artist a ?artistType .
9   ?relatedArtist a ?artistType .
10
11  {?artist foaf:name ?artistN} UNION {?artist rdfs:label ?
       artistN}
12
13  { ?artist dbo:associatedMusicalArtist ?relatedArtist }
14  UNION { ?relatedArtist dbo:associatedMusicalArtist ?
       artist }
15  UNION { ?relatedArtist dbpedia2:associatedActs ?artist }
16  UNION { ?artist dbpedia2:associatedActs ?relatedArtist }
17  UNION { ?artist rdfs:seeAlso ?relatedArtist }
18  UNION { ?artist dbo:bandMember ?relatedArtist }
19  UNION { ?relatedArtist dbo:bandMember ?artist }
20  UNION { ?artist dbo:formerBandMember ?relatedArtist }
21  UNION { ?relatedArtist dbo:formerBandMember ?artist }
22
23  {?relatedArtist foaf:name ?relatedArtistN }
24  UNION
25  {?relatedArtist rdfs:label ?relatedArtistN }
26
27  FILTER (?artistN = "Portishead"@en)
28  FILTER (?artistType IN (yago:Musician110339966, yago:
       Group100031264, dbo:Artist, schema:MusicGroup))
29  FILTER (lang(?relatedArtistN) = "en")
30  }
31  GROUP BY ?artist ?relatedArtist
```

Výpis 3.3: SPARQL Dotaz

Data získaná z endpointu mají následující podobu. Pomocí parsování získané struktury aplikace vyextrahuje názvy umělců a použije je k doporučení vazeb u umělců, kteří se již nachází v hudební knihovně.

```
1 { "head": { "link": [], "vars": ["raN"] },
2   "results": { "distinct": false, "ordered": true, "
       bindings": [
3     { "raN": { "type": "literal", "xml:lang": "en", "value
         ": "Joe Volk" }},
4     { "raN": { "type": "literal", "xml:lang": "en", "value
         ": "The Blessing" }},
5     { "raN": { "type": "literal", "xml:lang": "en", "value
         ": "Massive Attack" }},
6     { "raN": { "type": "literal", "xml:lang": "en", "value
         ": "CirKus" }},
7     ...] }
8 }
```

Doporučování alb by v principu mělo fungovat podobně, z mého pozorování však vyplývá, že data tohoto typu se na Wikipedii vyskytují méně často a vedou spíše na příbuzná témata, než podobná alba. Doporučování alb tedy z tohoto důvodu v této fázi nebudu implementovat.

3.5 Návrh uživatelského rozhraní

Důležitou částí návrhu aplikace je zpracování očekávané podoby uživatelského rozhraní včetně zapracování navržené funkcionality. Před samotnou implementací projektu jsem vypracoval na základě popsaných případů užití návrhy některých pohledů a rozložení komponent, které budou tvořit kostru rozhraní.

3.5.1 „Lo-fi“ model - Wireframe obrazovky

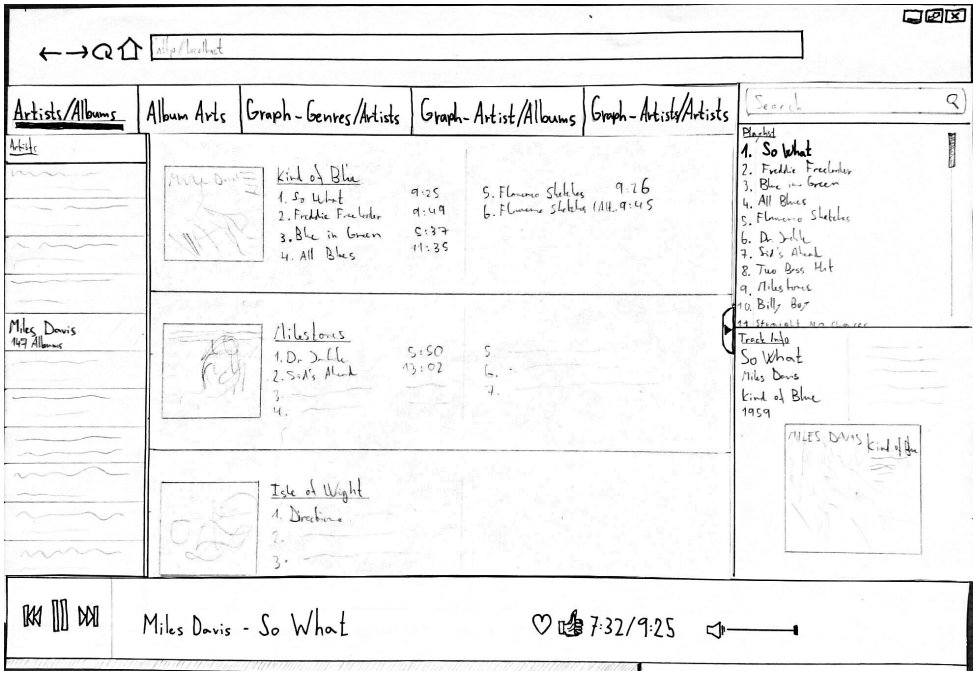
Pro účely lepší představy o podobě aplikace se vytváří tzv. lo-fi model aplikace, který je vhodný také k ověření funkcionality před implementací. [15] Vytvořil jsem tedy wireframe obrazovky uživatelského rozhraní, jedná se o základní prototyp, který slouží jako předloha pro funkční („hi-fi“) prototyp. Model vychází z požadavků a vlastní představy o podobě grafových pohledů. Obrazovky jsem vytvořil v rámci semestrální práce z předmětu MI-NUR (Návrh uživatelského rozhraní) v podobě papírového modelu.

Wireframe na obrázku 3.9 znázorňuje seznamový pohled pro zobrazení detailů alb zvoleného umělce. Panel umělců je na levé straně, vybrán je umělec se jménem „Miles Davis“ a zobrazena jsou pod sebou jeho alba. Jednotlivý detail alba obsahuje název alba, obrázek a seznam skladeb. Zde i u ostatních wireframes se vyskytují panely, jejichž funkcionalita je popsána detailně v sekci 3.5.2.

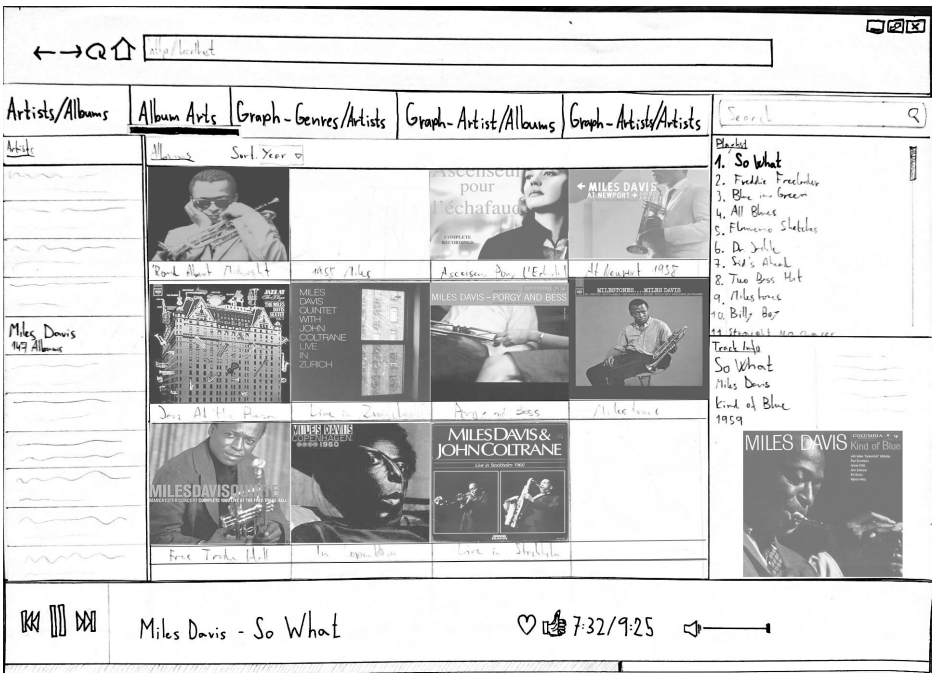
Obrázek 3.10 obsahuje wireframe s pohledem, kde jsou v hlavní části také znázorněna alba, rozdílem je však jejich vizuální reprezentace - použity jsou jen obrázky alb, což šetří užitným místem a poskytuje možnost rychlé orientace v knihovně.

Wireframe na obrázku 3.11 odpovídá případu užití UC12 (2.3) - obsahuje grafový pohled s pracovním názvem „Artist/Albums“, tedy aktuálně přehrávaného umělce, jeho alb a dalších umělců, kteří se na albech podílejí (tedy mají k němu v systému pevnou vazbu). Hlavní umělec je uzel ve středu pohledu ve tvaru kruhu, na něj jsou napojena alba jako uzly ve tvaru čtverce a na ně v poslední vrstvě přispívající umělci.

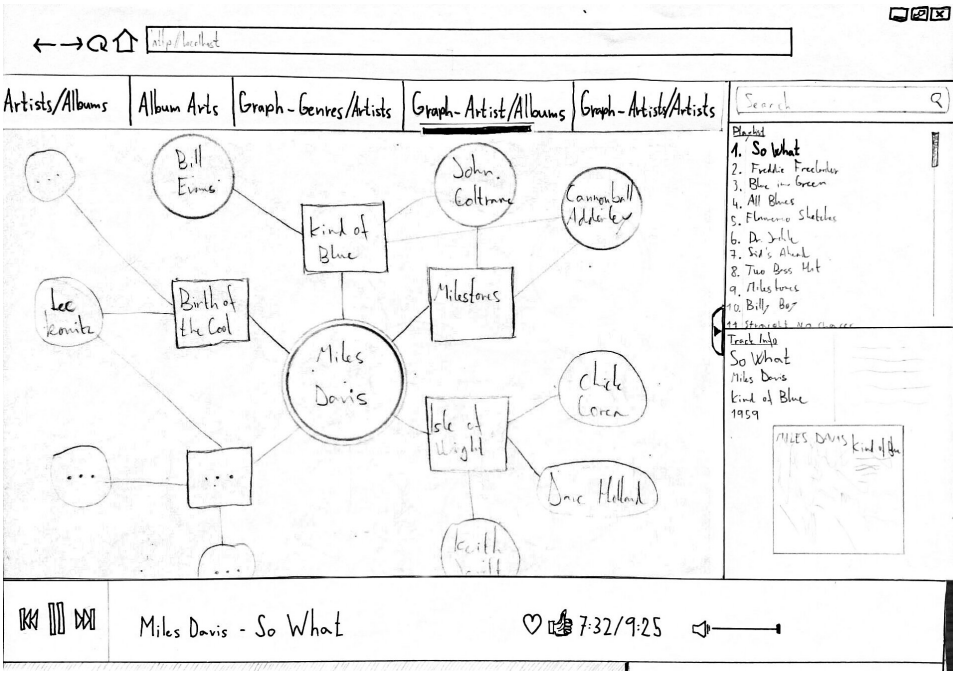
Poslední wireframe na obrázku 3.12 znázorňuje grafový pohled pro prohlížení a editaci vztahů podobnosti umělců. Znázorněna je také funkcionalita zvýraznění sousedních uzlů a hran po kliknutí na uzel, která bude implementována.



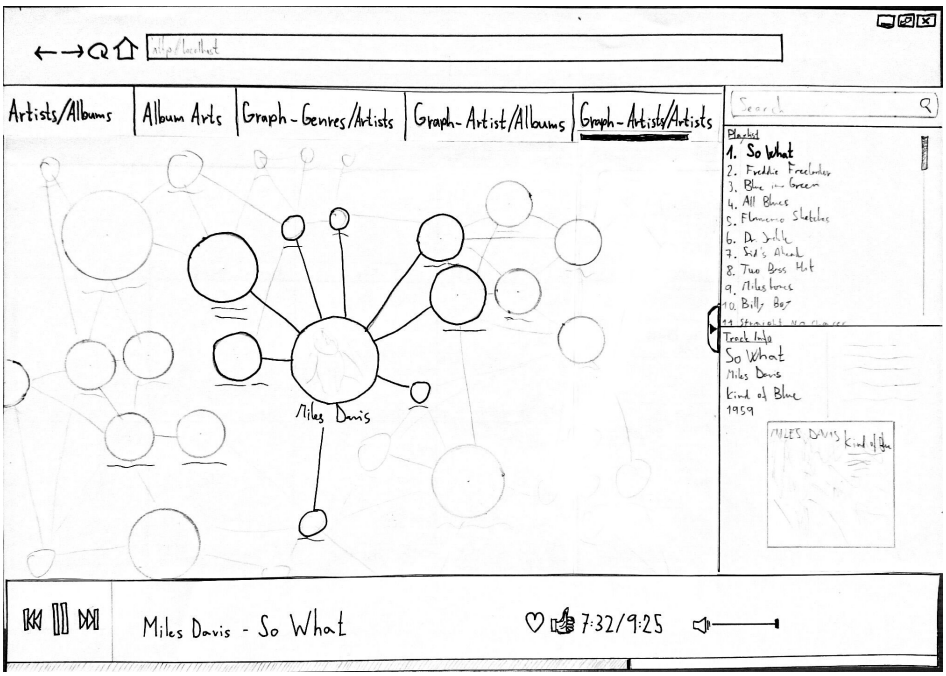
Obrázek 3.9: Wireframe - Pohled Album details



Obrázek 3.10: Wireframe - Pohled Album arts



Obrázek 3.11: Wireframe - Grafový pohled umělec-alba-umělci



Obrázek 3.12: Wireframe - Grafový pohled propojení umělců se zvýrazněním sousedních uzlů

3.5.2 Struktura komponent

Na vytvořených wireframe obrazovkách lze pozorovat základní strukturu jednotlivých komponent některých stěžejních částí uživatelského rozhraní. Rozhraní využívá koncept, který se nazývá „single-page application“ a je splněním nefunkčního požadavku N2. Pro zachování pocitu používání klasické desktopové aplikace jsou využity technologie, kdy nedochází k aktualizaci celé stránky po kliknutí na nějaký odkazový element. Aplikace je tak vysoce responsivní a je jednodušší reprezentovat její stav.

Aplikace má 4 hlavní panely - horní, hlavní, pravý a spodní. Následuje stručný popis jejich funkcionality.

Horní panel zajišťuje rychlé přepínání jednotlivých pohledů v hlavním panelu. Toto přepínání má podobu klasických „tabů“ (v jiné terminologii se používá termín karty), které jsou osvědčenou metodou pro přepínání pohledů v rámci aplikace - tradičně ji využívají např. webové prohlížeče. Aktivní panel je vizuálně odlišen od ostatních.

Pravý panel obsahuje vždy viditelný hlavní playlist a informace o aktuálně hrající skladbě. Permanentní viditelnost těchto komponent je důležitá, protože tyto funkce jsou v systému nejčastěji používané.

Hlavní panel plní funkci kontejneru pro aktivně zvolený pohled.

Spodní panel zajišťuje ovládání přehrávání pomocí ovládacích prvků a dále informuje uživatele pomocí tzv. „progress baru“ (klikacího posuvného měřítka) o aktuální pozici přehrávání.

Dalším důležitým prvkem, který v prototypu není znázorněn, je **modální vyskakovací okno**, které plní funkci nepermanentního zobrazení vyžádaných informací nebo např. editačních obrazovek - tedy funkcí, které po vykonání uživatele navrátí zpět na původní obrazovku. Pro účely zachování rozpracovaného stavu v aplikaci jsem zvolil tento koncept, kdy se základní pohled aplikace překryje tímto dočasným oknem, které je možné zavřít a navrátit se tak jednoduše k původní práci.

3.5.3 Navržené pohledy

Následuje výpis hlavních navržených pohledů. Aplikace bude zahrnovat i další pohledy, které jsou z pohledu návrhu triviální a nebudu je zde popisovat.

Grafový pohled „Artist/Albums“

Pohled znázorní v rámci grafové struktury jako centrální entitu hlavního umělce právě hrající skladby. Na hlavního umělce budou napojena všechna alba, se kterými je propojen ve vztahu hlavního nebo přispívajícího umělce. Poslední vrstvou budou další umělci, kteří se na albech zúčastnili (v systému existuje vazba umělec-album).

Uživateli tedy budou přehledně zobrazena související hudební díla z pohledu pevných vazeb, což je jeden ze způsobů, jak objevit další hudbu pro pře-

hrání. Tradiční přehrávače umožňují zobrazit alba nějakého umělce ve formě seznamu (podobně jako v pohledu „Detaily alba“), chybí však přehled dalších souvisejících umělců. Za předpokladu, že jsou vyplněny tyto vazby, aplikace umožňuje lepší navigaci v knihovně.

Obdobný pohled by v budoucnu mohlo být zobrazení skladatele a jeho děl, s případným zahrnutím umělců. Toto by bylo vhodné zejména pro klasickou hudbu, kde uživatel může chtít rozlišovat různé interpretace nějakého hudebního díla, které má jednoho specifikovaného skladatele.

Grafový pohled „Genres/Albums“

Součástí pohledu budou entity Genre a Album. Žánry budou spojovat alba, která jsou na žánry navázána. Pohled také bude znázorňovat hierarchii samotných žánrů. Vznikne tedy pohled, který vizuálně rozliší skupiny alb podle žánru.

Žánry jsou barevně rozlišeny, což může být základem pro další vizuální oddělení entit. Nabízí se možnost barevně označit i jednotlivá alba výpočtem průměrné barvy z napojených žánrů.

Grafový pohled „Artists/Artists“

Tento pohled zobrazí umělce a podobnostní vazby mezi nimi. Zároveň umožní vazby vytvářet způsobem popsaným v případu užití UC6.

Grafový pohled „Labels/Albums“

Pohled je ekvivalentní pohledu „Genres/Albums“, entity však jsou Label a Album.

Složený grafový pohled

Pro účely větší přizpůsobitelnosti by aplikace měla obsahovat pohled, který umožní uživateli přímo nastavit zobrazené entity a vztahy. Vzhledem k vysoké výpočetní náročnosti grafové vizualizace a omezeným prostředkům na cílové platformě je nutné kontrolovat počet zobrazených entit. Na druhé straně přílišné zjednodušení zobrazovaného modelu může způsobit, že pohled přestane plnit svoji hlavní funkci přehledné orientace v datech.

Tento problém lze částečně řešit omezením maximálního počtu zvolených entit a vztahů (obecně zvolených, zde se nejedná o omezení počtu vykreslených uzlů a hran). Není však jasné, kde nastavit tuto hranici. Pro úspěšnou implementaci tohoto pohledu proto bude nutné experimentálně ověřit různá nastavení tohoto parametru.

Další pomocnou technikou je shlukování uzlů a hran podle společné entity spolu se zamezením jejich vykreslení do doby, než jsou uzly vyžádány uživatelskou akcí (např. dvojklik na sdružující uzel). Příkladem může být shluknutí alb

podle společného žánru. V případě silně propojených dat může tato metoda být neefektivní.

Seznamový pohled „Album Arts“

Pohled bude obsahovat filtrovatelný seznam umělců spolu s obrázky alb seřazenými do mřížky. Po vybrání umělce se zobrazí pouze alba, která jsou s ním asociována. Album bude možné přetáhnout do hlavního playlistu, kde budou posléze přidány všechny jeho skladby. Dvojklik na album zobrazí v modálním okně další detaily včetně seznamu skladeb.

Seznamový pohled „Album details“

Pohled zobrazí stejný filtrovatelný seznam umělců, jako u „Album Arts“, místo mřížky obrázků zde budou podrobnější pohledy na alba - zobrazí se metadata pro dané album, vazby a seznam skladeb.

Editační pohledy

Editační pohledy budou zobrazeny v modálním okně ve formě formuláře. V případě odkázání na další editaci (např. editace skladeb u alba) bude možné se v modálním okně vrátit k původnímu pohledu, aby uživatel nabídku nemusel znovu vyvolávat.

Pohled „Settings“

Pohled zobrazí všechna dostupná nastavení na jednom místě v podobě seznamového pohledu.

3.5.4 Grafové vizualizace

Návrhu pohledů využívajících interaktivní grafové vizualizace předcházel průzkum knihoven, které takovou funkcionalitu na zvolené platformě poskytují. Hlavním důvodem byla snaha minimalizovat případné nutné úpravy lo-fi modelu při zjištění, že navržené řešení nelze implementovat. Po zjištění schopností jednotlivých možných knihoven (zejména Vis.js a D3.js) jsem verifikoval možnost jejich využití v modulární struktuře rozhraní. Dále již následoval samotný návrh těchto pohledů.

Z průzkumu vyplynulo, že knihovny poskytují koncovému uživateli možnost přesouvat uzly, přibližovat, oddalovat a posouvat celou grafovou strukturu, v některých případech i shlukovat skupiny uzlů. Knihovny zpravidla využívají fyzikální simulace pro pozicování uzlů. Iniciální polohy jsou určeny tímto mechanismem a uživatel má poté možnost uzly v pohledu manuálně přesouvat. [16]

Kapitola 4

Implementace

V této kapitole je popsán proces vývoje a technické parametry zvolené při implementaci navrženého projektu. Popis je rozdělen podle příslušných částí aplikace - frontend část zajišťuje uživatelské rozhraní, backend část zajišťuje zpracování dat a poskytuje metody dostupné přes definované API.

4.1 Frontend část aplikace

4.1.1 Framework Vue.js a související části implementace

Při plánování implementace jsem se rozhodl využít nějaký z frontendových webových frameworků podporujících reaktivní prostředí. Aplikace má strukturu „single-page“ aplikace (popsáno v předchozích kapitolách), je tedy žádoucí, aby při uživatelských akcích nedocházelo k opakovanému znovunačítání stránek. Tyto frameworky dále využívají centrální datový model, kde jsou ukládány změny hodnot proměnných vyvolané uživatelskou akcí. Všechny výskyty proměnných v rozhraní jsou na tento model napojeny, případné změny se tedy projeví ihned a na všech místech v rozhraní zároveň. To dále napomáhá pocitu, že uživatel používá aplikaci s tradičním desktopovým rozhraním. [17]

V současné době existuje poměrně velké množství těchto frameworků [20], které jsou zpravidla opensource a mají rozsáhlou vývojářskou komunitu. Vývojář má tedy na výběr z dostatečného množství variant.

Pro realizaci projektu jsem zvolil framework Vue.js [21]. Tato knihovna je principem funkce velmi podobná oblíbenému frameworku React, nabízí však rychlejší zpracování operací a její architektura umožňuje využít modulárně pouze ty části, které vývojář přímo potřebuje. Dále Vue.js umožňuje používat klasické validní HTML s rozšířením o speciální atributy, které umožňují integraci frameworku. Oproti dalšímu populárnímu frameworku Angular nabízí

[20] Základní přehled např. zde - `https://github.com/showcases/front-end-javascript-frameworks`

[21] `https://vuejs.org/`

Vue větší jednoduchost (avšak ne na úkor možností), což považuji za důležité pro nově příchozí vývojáře k projektu - naučit se Vue.js by mělo být dle autorů významně rychlejší. [18]

Vue.js šablona s Webpack

Projekt jsem založil pomocí Vue.js komunitou doporučované základní šablony [22]. Šablona zahrnuje konfiguraci knihovny Webpack, která se pomocí dalších částí stará o transpilaci (překlad kódu jednoho jazyka do jiného) ES6 [23] standardu do čistého JavaScriptu, který podporuje většina prohlížečů, včetně zpracování Vue souborů. Pokoušel jsem se tedy při implementaci využívat funkce zmíněného standardu. Webpack umožňuje především rozdělit závislosti na separátní části, které se načítají až když je potřeba, snížit počáteční čas načítání a integrovat knihovny třetích stran jako moduly. [24]

Komponenty a jejich rozdělení

Frontend část aplikace je rozdělena do komponent, což je tradiční přístup při implementaci v použitém frameworku. Jednotlivé komponenty jsou umístěny v souborech s příponou „Vue“ v adresáři „src/components“. Soubory jsou pro přehlednost kategorizovány podle umístění v hlavních panelech.

Každá komponenta sestává ze 3 hlavních částí: **template** - HTML s rozšířenými atributy, **script** - skriptová část, **style** - kaskádové styly. Důležité součásti jsou tedy seskupeny podle zobrazovaných dat, skripty se týkají pouze lokálně použitých prvků, styly lze dle volby aplikovat pouze na vybranou komponentu nebo na komponenty použité jako součást nadřazené.

V případě grafových pohledů je použita nadřazená komponenta, která obsahuje kód pro inicializaci grafové vizualizace. Tato komponenta v sobě obsahuje jednotlivé verze grafových pohledů, které zprostředkovávají data a nastavení pro daný graf.

Z důvodu náročnosti inicializace grafových pohledů a zachování zobrazeného stavu u ostatních komponent při přepínání „tabů“ v horním panelu jsem použil pro hlavní panel direktivu „keep-alive“, která dané pod-komponenty uloží do paměti spolu s jejich stavem. Pokud uživatel přepíná mezi pohledy, jejich rozpracovaný stav tedy zůstane zachován.

Následuje výpis a vysvětlení obsahu komponent, které jsem implementoval, a jsou tedy v různých kombinacích součástí aplikace.

Komponenty použité ve spodním panelu se nachází ve složce „bottom“.

[22] `https://github.com/vuejs-templates/webpack`
[23] `http://es6-features.org/`
[24] `https://webpack.github.io/docs/what-is-webpack.html`

Název	**Popis**
PlayerControls	Ovládací prvky přehrávače
PlayingTrack	Pole pro zobrazení názvu umělce a názvu přehrávané skladby
ProgressBar	Lišta zobrazující aktuální pozici v přehrávané skladbě, kliknutím na místo na liště se přehrávání přesune na danou pozici

Tabulka 4.1: Seznam komponent pro spodní panel

Komponenty použité v hlavním panelu se nachází ve složce „main“.

Název	**Popis**
AlbumArts	Seznamový pohled zobrazující obrázky alb
AlbumDetailsList	Seznamový pohled zobrazující detailní pohledy na alba
ArtistAlbumsGraph	Grafový pohled Artist/Albums
ArtistsAlbumArts	Seznamový pohled seskupující panel ArtistsPanel s pohledem AlbumArts
ArtistsAlbumDetails	Seznamový pohled seskupující panel ArtistsPanel s pohledem AlbumDetailsList
ArtistsArtistsGraph	Grafový pohled Artists/Artists
ArtistsPanel	Seznamový pohled zobrazující názvy umělců v klikatelném seznamu
GenresAlbumsGraph	Grafový pohled Genres/Albums
GraphCanvas	Základní grafový pohled sloužící pro vykreslení plátna a vizualizace
MultiGraph	Grafový pohled kombinující Artists, Albums, Labels a Genres
Playlists	Seznamový pohled pro procházení uložených playlistů

Tabulka 4.2: Seznam komponent pro hlavní panel

Komponenty použité v pravém panelu se nachází ve složce „right“.

Název	**Popis**
DefaultRight	Seskupení menších panelů Playlist a TrackInfo
Playlist	Hlavní playlist
TrackInfo	Zobrazení detailních informací o právě přehrávané skladbě a jejím albu

Tabulka 4.3: Seznam komponent pro pravý panel

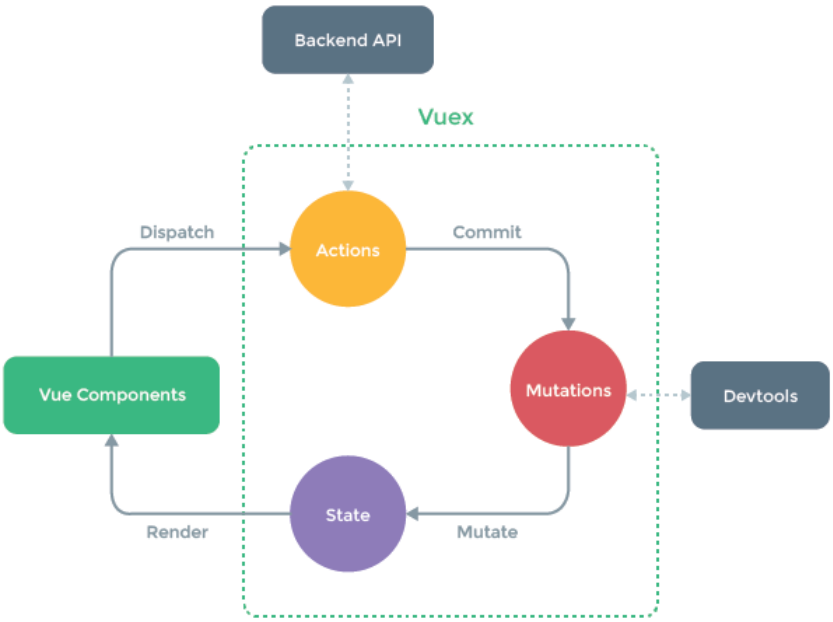
Ostatní komponenty se nachází ve složce „misc“.

Název	**Popis**
AlbumDetails	Pohled s detaily a seznamem skladeb pro jedno album
EditAlbum	Editační formulář pro album
EditAlbumTracks	Editační formulář pro skladby alba
EditEntityLabels	Editační formulář pro změnu štítků u entity (používané pro alba a skladby)
EditGenre	Editační formulář pro žánr
EditPlaylist	Editační formulář pro playlist
FiltersArtistsAlbums	Pohled s filtry pro pohledy s kombinací entit Artist a Album
FiltersArtistsAlbumsMore	Pohled s dalšími poli rozšiřující FiltersArtistsAlbums
Labels	Komponenta pro zobrazení formátovaných štítků, použito např. v AlbumDetails
Modal	Modální komponenta pro dialogové pohledy
ModalActions	Součást komponenty Modal, která umožňuje jednotlivým akcím přiřadit různá akční tlačítka
PlaylistActions	Součást komponenty Playlist, která jej rozšiřuje o akční tlačítka
PlaylistDetails	Zobrazení detailů uloženého playlistu
Settings	Pohled s nastaveními aplikace

Tabulka 4.4: Seznam ostatních komponent

Vuex

V praxi lze často narazit na problém komunikace jednotlivých komponent a udržení společného stavu. Pro tyto účely poskytuje Vue.js volitelně součást Vuex, která zajišťuje správu stavu aplikace. Na následujícím diagramu je znázorněna struktura fungování Vue.js aplikace při použití knihovny Vuex. Komponenty volají akce, které volitelně komunikují s backend API. Po získání potřebných dat je zavolána mutace - atomická operace, která změní stav ve Vuex modelu. Tyto změny jsou poté renderovány zpět do komponent. [19]



Obrázek 4.1: Struktura Vuex

Uvedený diagram je převzat z dokumentace [19].

Jako pomocný mechanismus jsem zvolil vytvoření další instance Vue, která je importována do komponent, kde je potřeba komunikovat, a použití Vuex Store by bylo zbytečně komplikované. Zde lze zaregistrovat události pomocí direktiv „$on(eventName)“ a „$emit(eventName)“, které lze následně využít v daných komponentách.

4.1.2 Ukázky implementovaného rozhraní

V příloze C jsou uvedeny základní ukázky implementovaného uživatelského rozhraní.

4.1.3 Realizace grafových vizualizací

Velmi důležitou součástí aplikace jsou grafové vizualizace, jejich implementaci proto bylo věnováno větší množství času.

Prvním krokem byla volba správné vizualizační knihovny. Aktuálně je k dispozici několik opensource javascriptových vizualizačních knihoven podpo-

rující grafové vizualizace [25]. Prozkoumal jsem dokumentace jednotlivých často používaných variant a zvolil jsem knihovnu Vis.js z důvodu dobrého výkonu, škálovatelnosti, možnosti přizpůsobit vzhled grafové struktury a rozvinuté opensource komunity. [16] U renderovaných uzlů je možné nastavit jejich tvar, barvu, obrázek a velikost. Toho je využito pro odlišení různých typů uzlů.

Problematickou částí bylo nastavení fyziky simulace celého grafu. Na jedné straně jsem se snažil zachovat přehlednost při větších přiblíženích (např. zamezit překryvům uzlů), na druhé straně je zde požadavek na čitelný pohled při oddálení (v grafu by např. měly být viditelné jednotlivé skupiny uzlů). Pro konkrétní nastavení není k dispozici shrnující literatura nebo „jedna nejlepší" metoda, parametry tedy bylo třeba experimentálně ověřovat na testovací hudební knihovně. V rámci implementace jsem tedy ladil příslušné parametry (s postupným zafixováním všech a změnami jednoho), dokud jsem nedošel k uspokojivému výsledku.

Výpis obsahuje objekt použitý při konfiguraci grafové knihovny. Důležitá část je „solver", tedy model výpočtu fyzikálních vlastností uzlů a hran. Zvolil jsem model „repulsion" jako ideální kompromis mezi malou výpočetní náročností a reprodukovatelně správným chováním vzhledem k podstatě dat. Správné chování definuji tak, že po dokončení stabilizace jsou související uzly zobrazeny v podobě jednotlivých komponent, uzly mají minimální vizuální překryv (zde jsou klíčové parametry „nodeDistance" a „springLength") a graf skončí ve stabilizovaném stavu v „rozumné" době (řádově jednotky sekund). Rychlost ukončení ovlivňuje parametr „iterations", který přestane počítat pozice uzlů po daném počtu iterací. Parametr „updateInterval" určuje, po kolika iteracích se vykreslí aktuální podoba grafu. Menší číslo zajistí plynulejší vykreslování, větší číslo menší výpočetní zátěž vykreslování.

```
physics: {
  enabled: true,
  repulsion: {
    nodeDistance: 200, centralGravity: 0.05,
    springLength: 120, springConstant: 0.2,
    damping: 0.05
  },
  solver: 'repulsion', maxVelocity: 30,
  minVelocity: 5, timestep: 0.88,
  adaptiveTimestep: true,
  stabilization: {
    enabled: true, iterations: 20,
    updateInterval: 3
  }
}
```

Výpis 4.1: Výsledné nastavení fyziky ve Vis.js

[25] Aktuální přehled knihoven - `https://github.com/anvaka/graph-drawing-libraries`

Renderování složitých grafů lze urychlit uložením pozic stabilizované sítě do databáze a následným znovunačtením při načítání. Implementaci tohoto řešení však komplikuje nestálá povaha dat, rozložení by bylo nutné znovu spočítat při každé změně. Testované sítě nezpůsobily zpoždění znemožňující pohodlné použití aplikace, proto jsem se při implementaci k tomuto řešení neuchýlil. Je však možné, že v budoucnu se bude jednat o nutný krok.

4.1.4 Hlavní playlist

Další implementačně složitější částí byl hlavní playlist. Snažil jsem se poskytnout podobné chování, které běžně přehrávače a jiné desktopové aplikace poskytují. Jmenovitě zejména výběr více skladeb v seznamu (ne nutně po sobě jdoucích), možnost přetahovat do playlistu na určenou pozici různé entity (album, skladba) s různým chováním a řazení skladeb v playlistu.

Pro tyto účely jsem zvolil oblíbené knihovny jQuery [26] a jQuery-UI [27], které nabízí rozšířenou platformu pro implementaci interaktivních seznamů (a další). Použil jsem komponenty „Sortable“, „Draggable“ a „Selectable“, jejichž kombinací jsem docílil požadovaného efektu. Oproti chování seznamů např. v OS Windows jsou zde jisté modifikace (zejména pomocné „úchytky“ sloužící pro přetahování objektů), bez kterých nebylo možné problém vyřešit, jedná se však ve výsledku o dobře funkční řešení.

Přetažení alba do playlistu na určenou pozici přidá všechny skladby v tomto albu. Dále lze vybrat v detailním pohledu alba nebo playlistu jednotlivé skladby (stejným způsobem, jako v hlavním playlistu) a následně je stejně přetáhnout na cílovou pozici. Po přidání nových skladeb playlist automaticky vytvoří nové objekty v DOM a aktualizuje pořadí skladeb. Skladby je také možné mazat označením a zmáčknutím tlačítka Delete, což usnadňuje rychlost používání.

4.2 Backend část aplikace

4.2.1 Node.js a Express.js

Pro realizaci backend části aplikace jsem zvažoval různé varianty jazyků a frameworků, z důvodu zachování jazykové konzistence s frontend částí (výhodné při vývoji i nasazení) jsem zvolil prostředí Node.js s frameworkem Express.js. Tento framework umožňuje vytvořit komplexní API a backend aplikace bez nutnosti používat komplexní konstrukty. To poskytuje dostatečnou flexibilitu, vysoký výkon a nízkou dobu pro naučení. [28]

[26] `https://jquery.com/`
[27] `https://jqueryui.com/`
[28] `https://expressjs.com/`

Jednotlivým zdrojům v API jsou přiřazeny tzv. „routes“ - funkce s mapováním na příslušné URL, které zpracovávají požadavek a vrací odpověď. V kódu jsou tyto funkce rozděleny podle jednotlivých entit do vlastních souborů.

Synchronize

Node.js využívá neblokující I/O model. [29] Pokud tedy např. v jedné funkci voláme externí API několikrát po sobě a s vrácenými daty vykonáme akci, není zajištěno pořadí vykonání těchto akcí, podobně je tomu např. u čtení souborů. V aplikaci se vyskytuje řada složitějších funkcí tohoto typu, např. komunikace s Neo4j API, což je detailněji popsáno v následující sekci. Pokud chceme zaručit návaznost akcí, je nutné jednotlivé funkce strukturovaně vkládat do tzv. „callbacks“ - funkcí, které jsou zavolány po dokončení nadřazené funkce. U složitějších akcí toto může vést k nepřehlednému kódu, což se v komunitě nazývá „callback hell“ [30].

Z tohoto důvodu jsem využil knihovnu Synchronize.js, která umožňuje transformovat funkce na synchronní, přičemž díky využití technologie „fibers“ nedochází k blokování vlákna celé aplikace [31].

4.2.2 Grafová databáze Neo4j

Pro ukládání dat jsem zvolil grafovou databázi Neo4j, která je velmi populární [32], opensource a poskytuje množství užitečných vlastností.

Databáze nabízí použití přes nainstalovanou službu a RESTful API nebo jako embedovaný soubor. Z důvodu oddělení komponent a kompatibility s použitými knihovnami pro přístup k DB jsem zvolil variantu s API. Součástí instalace aplikace tedy bude i závislost Neo4j, která bude běžet jako separátní služba, typicky na portu 7474. [20]

Cypher

Cypher je deklarativní jazyk pro komunikaci s databází, umožňuje vykonávat komplexní dotazy v přehledné syntaxi. Struktura dotazů je podobná jako u SQL, dotazy jsou složeny z jednotlivých klauzulí. [20]

Dotaz má obecně podobu odpovídající následujícímu výpisu. Hledáme všechny trojice (část „MATCH“), kde z uzlu „n1“ typu „Label1“ vychází hrana „rel“ typu „TYPE“ končící v uzlu „n2“ typu „Label2“. Na trojice je aplikována podmínka „WHERE“, kde přistupujeme k vlastnosti hrany a použijeme ji ve filtrovací podmínce. Zde může být jakákoli podmínka zahrnující vlastnosti odpovídajících uzlů i hran. Nakonec se pomocí části „RETURN“

[29] `https://nodejs.org/en/`

[30] `http://callbackhell.com/`

[31] Podrobnější popis např. na `https://mixmax.com/blog/node-fibers-using-synchronize-js`

[32] `https://db-engines.com/en/ranking/graph+dbms`

vrátí u všech získaných trojic požadovaná vlastnost hrany, její typ a vlastnost uzlu „n1“.

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property > {value}
RETURN rel.property, type(rel)
```

Výpis 4.2: Ukázka Cypher dotazu

V práci je použito mnoho dalších konstruktů tohoto jazyku, včetně dotazů pro vytvoření nových uzlů a hran. Popis by byl vyčerpávající a jazyk je dobře popsán v dokumentaci, proto čtenáře odkazuji do dokumentace [20].

Seraph

Pro komunikaci s Neo4j API používám knihovny Seraph [33] a částečně Seraph-model [34]. Seraph umožňuje volat dotazy pomocí definovaného rozhraní s odpovědí v tradičním callbacku ve stylu Node.js.

Následující ukázka převzatá z dokumentace uloží do databáze uzel s vlastnostmi „name“ a „age“ a v callbacku jej pomocí reference vymaže. V případě chyby je vyvolána chyba, kterou lze tradičně ošetřit pomocí try-catch mechanismu.

```
var db = require("seraph")("http://localhost:7474");
db.save({ name: "Test-Man", age: 40 }, function(err, node)
    {
  if (err) throw err;
  console.log("Test-Man inserted.");
  db.delete(node, function(err) {
    if (err) throw err;
    console.log("Test-Man away!");
  });
});
```

Výpis 4.3: Ukázka Seraph

Seraph-model je rozšířením knihovny seraph, kde lze definovat modely, což se projeví větší čitelností a objektově orientovanou zaměřeností kódu. Modely zároveň umožňují definovat strukturu jednotlivých entit včetně požadovaných parametrů. Tuto funkcionalitu však v aplikaci nevyužívám z důvodu nekompatibility s PUT metodou, kde může být obdržena jen část reprezentace objektu a není následně možné objekt uložit bez dalších zbytečných operací. Využity jsou metody pro jednoduché čtení a tvorbu entit, v případě složitějších dotazů je použita přímo knihovna Seraph v kombinaci s příslušným Cypher dotazem.

[33] https://github.com/brikteknologier/seraph

[34] https://github.com/brikteknologier/seraph-model

4.2.3 Čtení metadat

Pro čtení základních metadat z hudebních souborů je použita knihovna musicmetadata [35], která podporuje většinu často používaných formátů (mp3 (1.1, 2.2, 2.3, 2.4), m4a (mp4), vorbis (ogg, flac), asf (wma, wmv)). V rámci skenování nových souborů jsou extrahovány základní společné vlastnosti `title`, `duration`, `trackNr`, `diskNr`, `year` a název alba a umělce. V případě absence základních vlastností jsem implementoval metodu pro extrakci vlastností z cesty k souboru. Pokud ani tato metoda nezíská žádná data, může uživatel metadata doplnit ručně v knihovně.

Funkce pro skenování nových souborů nejdříve vytvoří interní reprezentaci nových entit, které poté přidá do databáze jako uzly a hrany příslušných typů.

Prostorem pro vylepšení této funkcionality je robustní podpora dalších formátů a souborů s nečitelnými metadaty.

4.3 Dokumentace a příprava projektu pro opensource

Projekt obsahuje dokumentaci pro programátory, která je dostupná na GitHub stránce projektu spolu s popisem RESTful API (konkrétní odkazy jsou seskupeny v následující sekci). Do této dokumentace je možné po konzultaci přispívat stejně, jako do celého projektu. Součástí dokumentace je i podrobný instalační manuál. Uživatelská dokumentace bude doplněna ve chvíli, kdy vznikne verze vhodná k produkčnímu užití aplikace. V tuto chvíli je projekt ve stádiu pokročile rozpracovaného prototypu, který se může výrazně měnit, nemá tedy smysl tvořit dokumentaci pro uživatele.

V kódu se dále vyskytují komentáře vysvětlující složitější metody a volbu konkrétních mechanismů pro jejich řešení. U jednoduchých metod se komentáře nevyskytují. V případě ustálení vyvíjeného modelu a spolupráce více vývojářů bude vhodné komentáře doplnit i zde.

Nejdůležitější navržená vylepšení a zjištěné chyby jsem přidal standardním způsobem jako součást projektu jako tzv. issues na GitHub - každý issue má svoji stránku, kde je možné změny komentovat a diskutovat. [21]

V rámci GitHub projektů lze pozorovat i celou historii vývoje projektu, která je složena z jednotlivých „commits“ - částí kódu publikovaných v rámci jednotlivých úprav. [21] Obsažena je i specifikovaná licence (sekce 1.5).

Projekt je tedy připraven pro vývoj v opensource komunitě, což bylo vytyčeno jako jeden z cílů práce. Projekt splňuje i oficiální tzv. opensource definici [22]. Dalším krokem bude také propagace projektu na různých sítích, kde se vyskytují vývojáři, kteří by mohli mít zájem se na projektu podílet.

[35] `https://github.com/leetreveil/musicmetadata`

4.3.1 Odkazy na online části práce

Následují odkazy na jednotlivé online části projektu. Repozitáře jsou rozděleny na 2 části tak, jak to bylo vytyčeno při navrhování projektu. Vše je publikováno na službách, které podporují opensource vývoj, a je tak umožněno komunitně přispívat do všech částí projektu.
Frontend část aplikace: `https://github.com/krystofspl/relplay-client`
Backend část aplikace: `https://github.com/krystofspl/relplay-server`
Dokumentace a popis instalace: `https://www.gitbook.com/book/krystofspl/relplay-docs/`
Specifikace RESTful API: `https://app.swaggerhub.com/apis/krystofspl/relplay-api/1.0.0`

Kapitola 5

Testování aplikace

Z důvodu ověření správnosti a vhodnosti řešení jsem provedl testování aplikace. Otestoval jsem se skupinou participantů uživatelské rozhraní aplikace.

Vhodné by byly také např. unit testy, integrační testy nebo zátěžové testy, kvůli časové náročnosti implementace jsem je však nestihl realizovat.

Testování s participanty podává užitečnou zpětnou vazbu a ověřuje, že aplikace se chová tak, jak má. [23] Aplikace poskytuje zejména v grafových pohledech netradiční rozhraní, je tedy přínosné zkoumat právě interakce uživatelů s aplikací.

5.1 Metoda testování kognitivním průchodem

Principem této metody je odpovědět u jednotlivých akcí na základní otázky, které se ptají na smysluplnost procesu a výsledku vykonání dané akce. Hlavním cílem je identifikovat případné problémy v použitelnosti aplikace z pohledu uživatele. Testují se jednotlivé případy užití postupným průchodem definovaných kroků a odpovídáním na dané otázky. [23] Následují zmíněné otázky.

Q0 Čeho chce uživatel dosáhnout?

Q1 Bude uživateli zřejmé, co má nyní udělat?

Q2 Spojí si uživatel popisek se svým cílem?

Q3 Dostane uživatel dostatečnou a srozumitelnou reakci na svoji akci?

V případě negativní odpovědi na některou z otázek 1 až 3 se k záznamu testování přiloží popis problému v podobě stručného komentáře. [23]

5.1.1 Výběr účastníků testování

Pro testování jsem vybral 4 participanty, kteří mají zkušenost s testováním uživatelských rozhraní (v rámci studia nebo pracovní zkušenosti) i s přehráváním hudby na PC. Věk ani pohlaví nebyly pro testování relevantní. Počet participantů vychází z doporučení 3-5 jedinců pro objevení většiny důležitých chyb. [23] Testování bylo provedeno na notebooku v klidném prostředí. K interakci s aplikací jsem participantům poskytl tradiční myš.

5.2 Výsledky testování

Následuje tabulka se souhrnem výsledků testování, detailní výsledky jsou v příloze D. U jednotlivých nálezů uvádím rozbor možného řešení problému. Pro klasifikaci závažnosti nálezů používám tři priority, které pomohou identifikovat důležitost jednotlivých problémů. [23]

1. **Priorita 1** - Závažný nedostatek nebo chyba, která zapříčiňuje obtížné používání systému.

2. **Priorita 2** - Nedostatek nebo chyba, která je pro uživatele nepříjemná, ale ne závažná.

3. **Priorita 3** - Nedostatek nebo chyba, která je spíše kosmetickou chybou, není závažné.

Nález	**UC**	**Pr.**	**Popis**	**Rozbor řešení**
N1	UC1	3	Systém při zvýrazňování nezobrazuje plochu pro výběr, jako např. na ploše Windows.	Je obtížné doplnit element tak, aby byla zachována přehlednost a funkčnost celku.
N2	UC1	2	Systém zobrazuje tlačítka na všech řádcích, není zřejmé, které z nich podržet.	Zobrazit tlačítko pouze u prvního vybraného elementu nebo vedle všech.
N3	UC1	3	Z podoby tlačítka není zřejmý jeho účel.	Změnit podobu tlačítka nebo přidat popisek akce.
N4	UC1	2	Po přidání skladeb není rozlišeno, které skladby byly právě přidány, což je nepřehledné u delšího playlistu.	Po přidání zvýraznit přidané skladby, např. jejich vybráním v playlistu.

Tabulka 5.1: Sourhn výsledků testování - 1. část

Nález	UC	Pr.	Popis	Rozbor řešení
N5	UC3	3	Pole mají netradiční vzhled, není zřejmé, že jsou editovatelná.	Změnit podobu polí, zesvětlit jejich pozadí, ponechat tradiční rámečky.
N6	UC4	2	Tlačítko pro editaci má nelogické umístění a je moc malé.	Akci vyvolávat namísto tlačítka přes kliknutí na popisek štítků.
N7	UC4	2	Tlačítko pro uložení je překryto nabídkou pro zvolení štítku, viditelné je až po kliknutí mimo.	Schovat automaticky nabídku po zadání hodnoty, problematické by pak však byla nutnost nabídku otevírat při zadávání více hodnot.
N8	UC5	1	Generování by mělo fungovat i s prázdným playlistem.	Doplnit funkcionalitu pro generování s náhodnou počáteční množinou.
N9	UC6	2	Hrana se musí přetáhnout přímo na uzel, nefunguje přetažení na popisek.	Použitá knihovna toto bohužel nepodporuje.
N10	UC6	1	V případe vetšího množství umělců je velmi složité najít konkrétního umělce.	Problém spočívá v podstatě pohledu, bude tedy vhodné implementovat zadávání podobnosti ještě na úrovni editačních formulářů.
N11	UC9	3	Zvýraznění aktuálně vybraného playlistu není dostatečně jasné.	Použít výraznější barvy pro zvýrazněný řádek.
N12	UC11	3	Podoba doporučení jako klasické odkazy evokuje přesun na jinou stránku, uživatel si není jistý, co kliknutí zpusobí.	Změnit podobu z odkazů na tlačítka.
N13	UC11	3	Zpracování doporučených hodnot není přehledné.	Problém souvisí s N12, řešení by mělo postihnout i tento problém.
N14	UC11	2	Chybí potvrzení o použití hodnot.	Vizuálně zvýraznit pole, které právě bylo upraveno novou hodnotou.

Tabulka 5.2: Sourhn výsledků testování - 2. část

5.3 Zhodnocení testování

Testování uživatelského rozhraní odhalilo některé problémy, které jsem nepostihl v rámci implementace a návrhu. Problémy jsou většinou kosmetického charakteru, nezapříčiňují nemožnost používat aplikaci a budou podle priority zpracovány v dalším vývoji projektu. Převaha odpovědí „ano“ na základní otázky indikuje dobrou průchodnost uživatelským rozhraním.

V budoucnu bude vhodné provést další a podrobnější testování v závislosti na další implementované funkcionalitě.

Závěr

V práci jsem se seznámil s principy fungování tradičních hudebních přehrávačů a organizérů hudebních knihoven. Po získání požadavků na systém jsem provedl jejich analýzu a formuloval uživatelské scénáře. Dále jsem navrhl a implementoval požadovanou funkcionalitu a připravil projekt pro další opensource vývoj. Provedl jsem testování uživatelského rozhraní s několika participanty a výsledky začlenil do projektu pro budoucí zpracování.

Implementovaná funkcionalita dobře pokrývá vytyčené cíle, vznikla aplikace umožňující správu existující hudební knihovny využívající grafovou strukturu pro vizualizace vztahů a generování playlistů, implementována je také agregace metadat z veřejného zdroje.

Práce se vedle originální funkcionality detailně zabývá i zpracováním té standardní, která je již poskytována tradičními přehrávači. V průběhu zpracovávání vyšlo najevo, že pro úspěšnou implementaci originální funkcionality je nutné detailně pochopit významy a způsoby používání základních entit, spolu s implementací základních funkcí, které jsou již implementovány jinde. Tento fakt způsobil částečné zpomalení vývoje rozšiřující funkcionality, cíle se však přesto podařilo splnit a aplikaci považuji za dobře připravenou pro další vývoj.

Jako hlavní negativum aktuálního stavu vnímám absenci testů na úrovni kódu, které jsem z časových důvodů nevytvořil, a bude jistě nutné je v rámci dalšího vývoje implementovat. Testování uživatelského rozhraní s participanty naopak ověřilo dobrou průchodnost rozhraní a poukázalo na některé nedostatky, které bude nutné opravit.

Vzniklá aplikace je funkční platformou připravenou na přidání dalších souvisejících funkcí. V práci jsem nastínil řadu dalších možných směrů vývoje práce, aktuální problémy a nápady jsou zahrnuty také v dokumentaci projektu. Mezi další navrhnuté směry, u kterých jsem poskytl i návrh řešení, náleží zejména automatická klasifikace hudebních souborů podle jejich obsahu s využitím strojového učení, centrální server pro sdílení uživatelských vazeb, rozšíření zdrojů pro agregaci metadat a další vizualizační pohledy. V této fázi

však bude kladen důraz na dokončení základních částí uživatelského rozhraní, které napomohou snadnějšímu používání aplikace. V aplikaci chybí např. kontextové menu pro jednotlivé položky po kliknutí pravým tlačítkem myši nebo úpravy více entit najednou.

Literatura

[1] Ricci, F.; Rokach, L.; Shapira, B.: *Introduction to recommender systems handbook*. Springer, 2011.

[2] MusicBrainz: *MusicBrainz documentation and model specification*. [cit. 2017-03-15]. Dostupné z: `https://musicbrainz.org/doc/`

[3] Gravell, D.: *Audio file formats: a comparison*. 2013, [cit. 2017-03-22]. Dostupné z: `https://www.blisshq.com/music-library-management-blog/2013/08/27/audio-file-formats-comparison/`

[4] UKCHARTS: Rules for chart eligibility - albums. leden 2007, [cit. 2017-03-15]. Dostupné z: `https://web.archive.org/web/20070627231755/http://www.theofficialcharts.com/docs/NEW_Album_Chart_Rules_2007_2.pdf`

[5] Google: *YouTube Help - Playlists*. 2017, [cit. 2017-03-21]. Dostupné z: `https://support.google.com/youtube/answer/57792`

[6] Přispěvatelé Wikipedie: *ID3*. 2017, [cit. 2017-03-18]. Dostupné z: `https://en.wikipedia.org/wiki/ID3`

[7] ID3.org: *Popis ID3v1 tagů*. 2000, [cit. 2017-03-22]. Dostupné z: `http://id3.org/ID3v1`

[8] Nilsson, M.: *ID3 tag version 2.4.0 - Main Structure*. 2000, [cit. 2017-03-18]. Dostupné z: `http://id3.org/id3v2.4.0-structure`

[9] Arlow, J.; Neustadt, I.: *UML 2 and the unified process: practical object-oriented analysis and design, kap. 3.6, 3.7*. Pearson Education, 2005.

[10] Bogdanov, D.: *From music similarity to music recommendation: Computational approaches based on audio features and metadata*. Dizertační práce, Universitat Pompeu Fabra, Barcelona, Spain, 2013.

[11] McKay, C.; Fujinaga, I.: Automatic Genre Classification Using Large High-Level Musical Feature Sets. In *ISMIR*, Citeseer, 2004.

[12] MusicBrainz: *LinkedBrainz documentation*. [cit. 2017-04-20]. Dostupné z: `https://wiki.musicbrainz.org/LinkedBrainz`

[13] MusicBrainz: *MusicBrainz API documentation*. [cit. 2017-04-04]. Dostupné z: `https://wiki.musicbrainz.org/Development/XML_Web_Service/Version_2/Search`

[14] W3C: *SPARQL Query Language for RDF*. 2008, [cit. 2017-04-05]. Dostupné z: `https://www.w3.org/TR/rdf-sparql-query/`

[15] Žikovský, P.: *2. Přednáška MI-NUR - Návrh UI, prototypy*. ČVUT v Praze, 2016, [cit. 2017-03-27]. Dostupné z: `https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x02-navh_a_prototyping.pdf`

[16] vis.js: *Dokumentace Network části knihovny*. 2017, [cit. 2017-03-30]. Dostupné z: `http://visjs.org/docs/network/`

[17] Vue.js: *Dokumentace knihovny*. 2017, [cit. 2017-04-12]. Dostupné z: `https://vuejs.org/v2/guide/`

[18] Vue.js: *Porovnání podobných frontend frameworků*. 2017, [cit. 2017-04-12]. Dostupné z: `https://vuejs.org/v2/guide/comparison.html`

[19] Vue.js: *Dokumentace knihovny Vuex*. 2017, [cit. 2017-04-12]. Dostupné z: `https://vuex.vuejs.org/en/intro.html`

[20] Neo Technology Inc.: *Dokumentace databáze Neo4j*. 2017, [cit. 2017-04-13]. Dostupné z: `https://neo4j.com/docs/`

[21] GitHub Inc.: *Dokumentace GitHub*. 2017, [cit. 2017-05-01]. Dostupné z: `https://developer.github.com/v3/git/`

[22] Open Source Initiative: *The Open Source Definition*. 2017, [cit. 2017-05-01]. Dostupné z: `https://opensource.org/osd`

[23] Sporka, A.: *2. Přednáška TUR na ČVUT FEL - Testování bez uživatelů*. ČVUT v Praze, 2015, [cit. 2017-04-27]. Dostupné z: `https://cent.felk.cvut.cz/courses/Y39TUR/?page=slides`

Příloha **A**

Seznam použitých zkratek

GUI Graphical User Interface

API Application Programming Interface

JSON JavaScript Object Notation

HTML HyperText Markup Language

DOM Document Object Model

SPARQL SPARQL Protocol and RDF Query Language

Příloha B

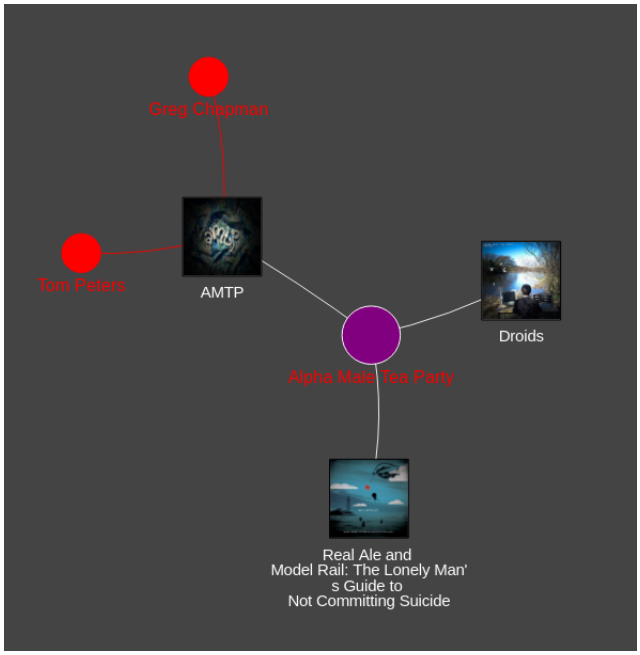
Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `relplay-client` ..... zdrojové kódy frontend části aplikace
  - `relplay-server` ..... zdrojové kódy backend části aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

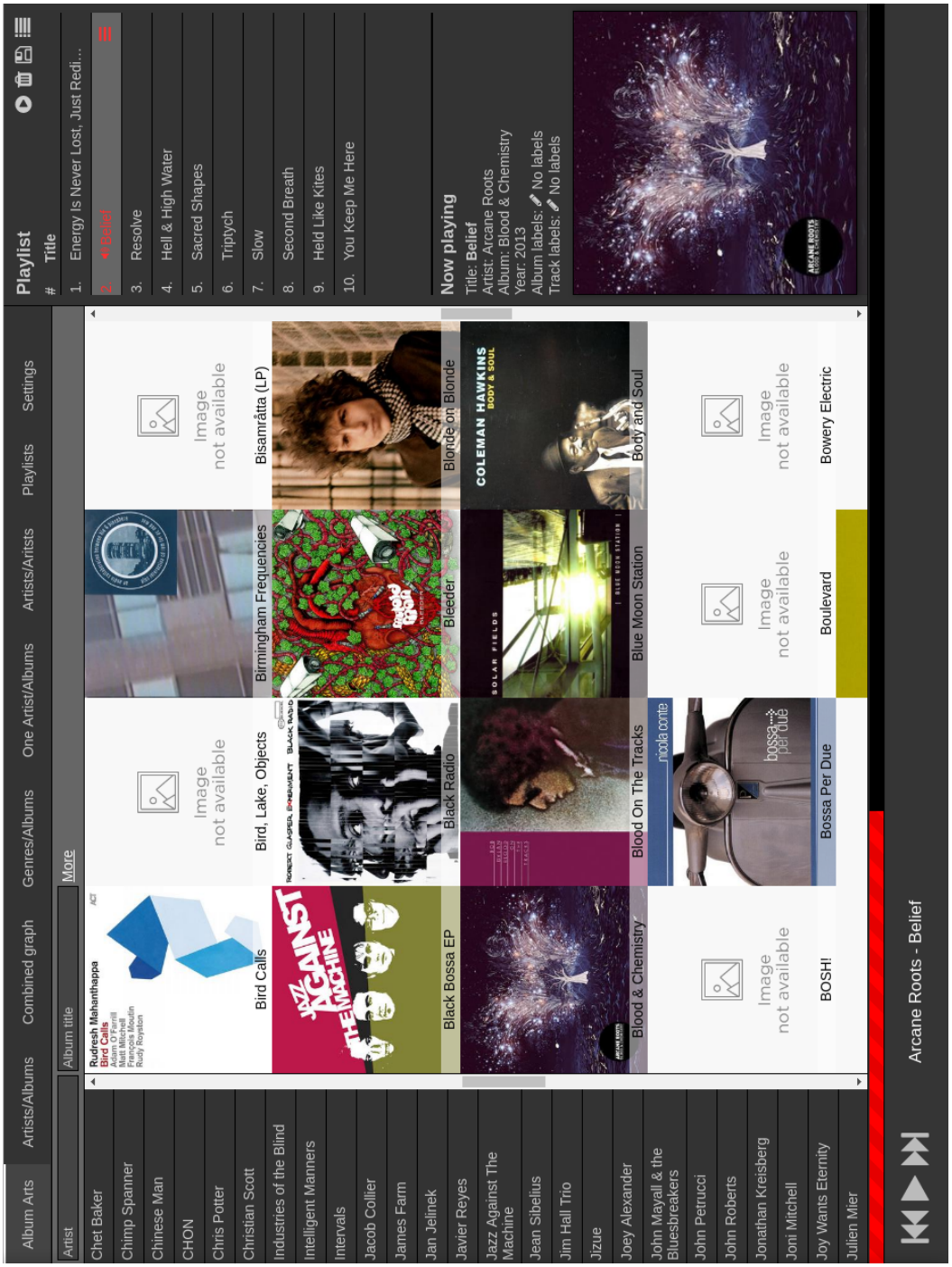
Příloha C

Ukázky uživatelského rozhraní

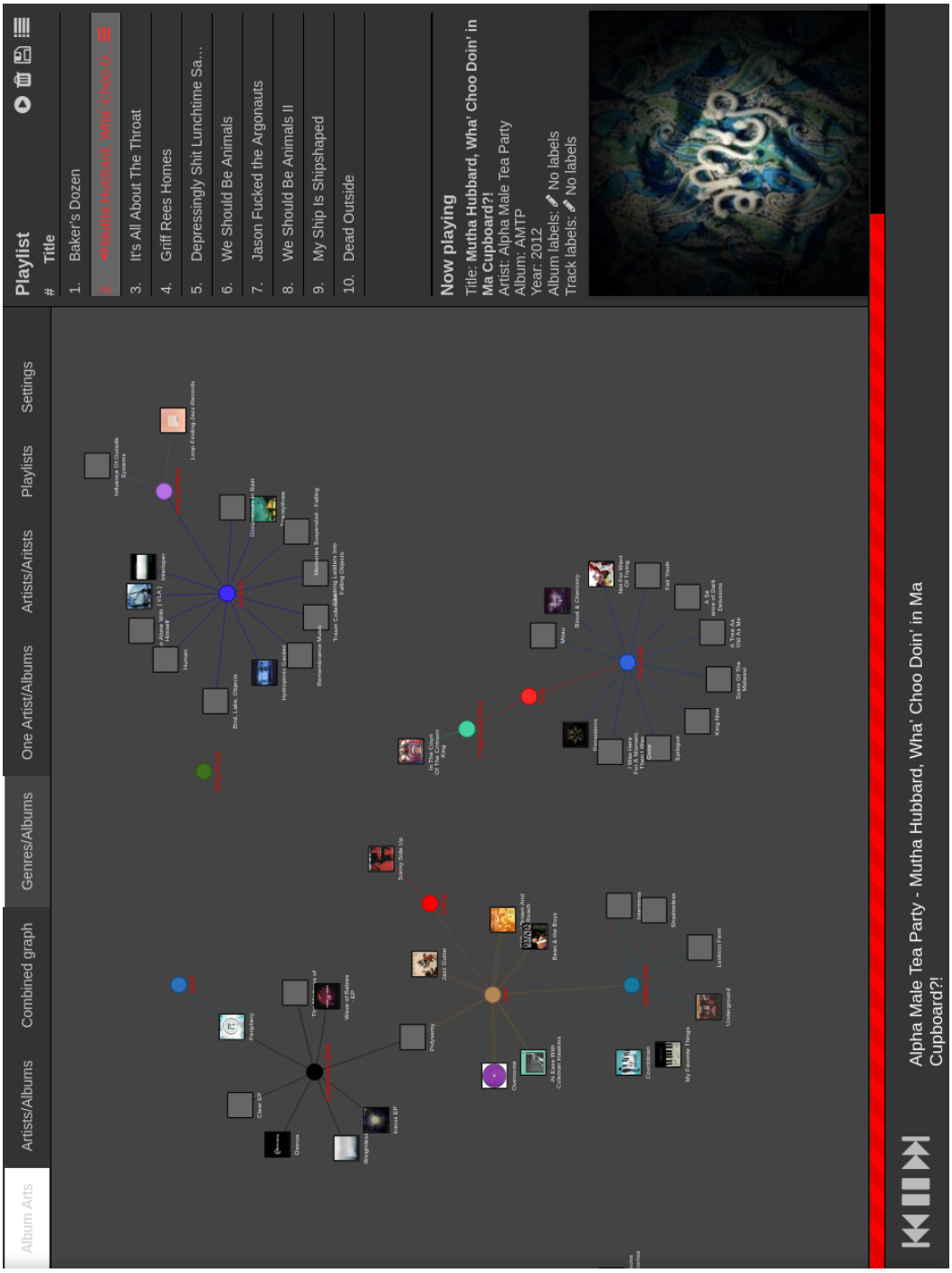
Následující obrázky ukazují popisovanou strukturu a jednotlivé vybrané pohledy aplikace.



Obrázek C.1: Grafový pohled Artist/Albums - detail

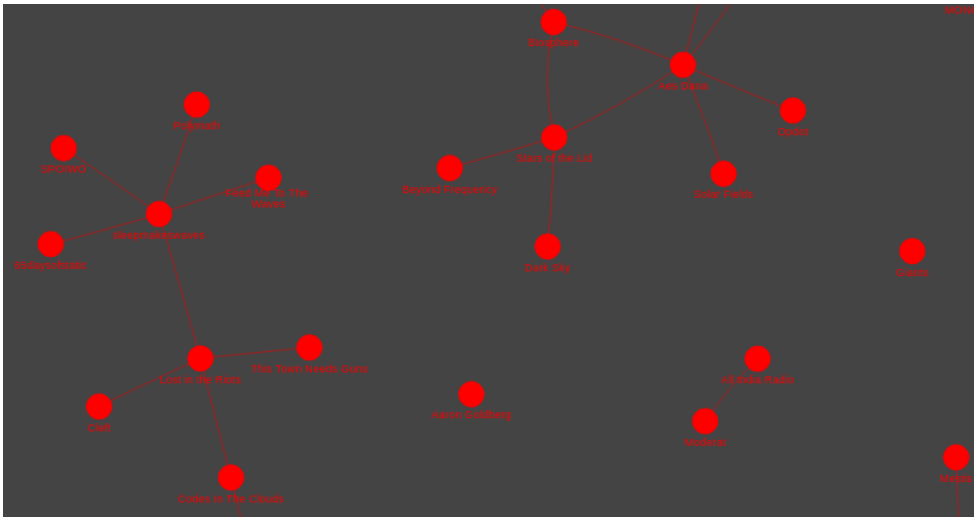


Obrázek C.2: Pohled na aplikační rozhraní s otevřeným pohledem Album Arts

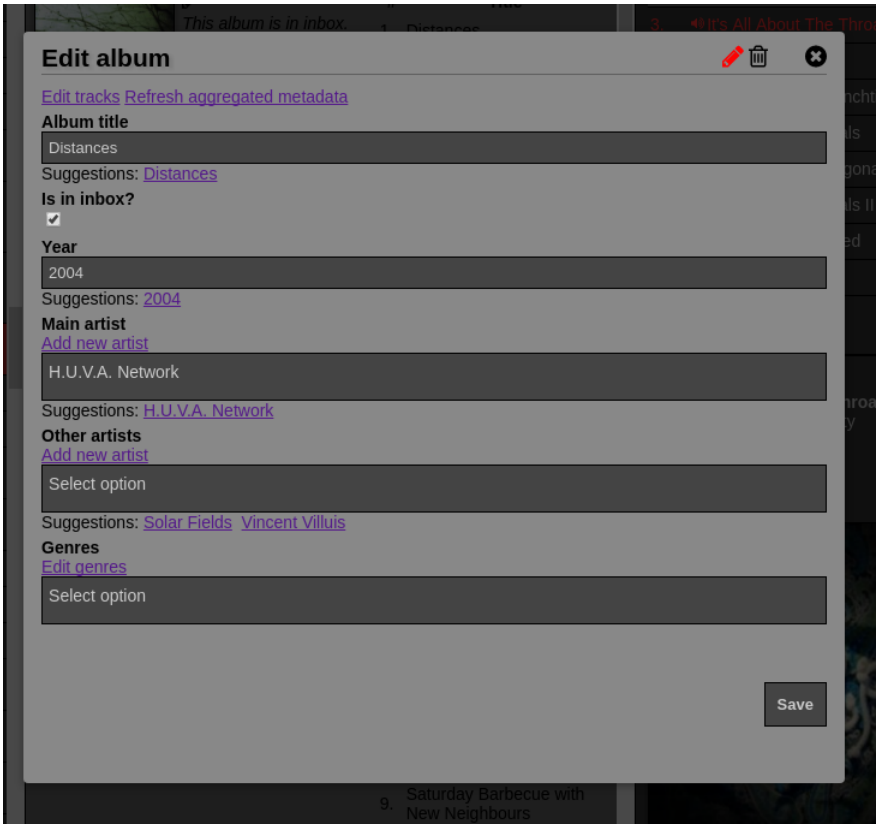


Obrázek C.3: Pohled na aplikační rozhraní s otevřeným grafovým pohledem Genres/Albums

Obrázek C.4: Grafový pohled Multigraph



Obrázek C.5: Grafový pohled Artists/Artists - detail



Obrázek C.6: Modální pohled s formulářem pro editaci alba - detail

Příloha D

Výsledky testování uživatelského rozhraní

Výsledky byly zapisovány do tabulky při interakci ze strany participanta testování. Některé kroky jsou seskupeny z důvodu logické návaznosti a společné reakce systému. Čtenáře odkazuji na seznam případů užití v sekci 2.3, pro zápis jsou použity identifikátory jednotlivých případů a čísla jednotlivých kroků v hlavním scénáři.

Jednotliví participanti jsou označeni zkratkami P1 až P4. V případě negativní odpovědi na nějakou otázku u některého participanta je uveden souhrnný popis problému. Nálezy jsou označeny písmenem N a příslušným pořadovým číslem. V případě kladné odpovědi a žádných připomínek pro daný krok záznam pro stručnost neuvádím.

D.1 UC1 - Přidat skladby do playlistu

Q0 - Uživatel chce přidat existující skladby do hlavního playlistu.

1. Kroky 1 až 4

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q3	Ne (P2,P3)	Systém při zvýrazňování nezobrazuje plochu pro výběr, jako např. na ploše Windows.	N1

2. Kroky 5, 6

Otázka	Odpověď (participant)	Popis	Nález
Q1	Ne (P1)	Systém zobrazuje tlačítka na všech řádcích, není zřejmé, které z nich podržet.	N2
Q2	Ne (P1)	Z podoby tlačítka není zřejmý jeho účel.	N3

3. Kroky 7, 8

Otázka	Odpověď (participant)	Popis	Nález
Q3	Ne (P1, P4)	Po přidání skladeb není rozlišeno, které skladby byly právě přidány, což je nepřehledné u delšího playlistu.	N4

D.2 UC2 - Přehrát skladbu

Q0 - Uživatel chce přehrát nějakou skladbu z hlavního playlistu.

1. Kroky 1 až 5
   Nebyly zaznamenány žádné problémy.

D.3 UC3 - Editovat metadata alba

Q0 - Uživatel chce editovat metadata existujícího alba.

1. Kroky 1, 2
   Nebyly zaznamenány žádné problémy.

2. Krok 3

Otázka	Odpověď (participant)	Popis	Nález
Q3	Ne (P2)	Pole mají netradiční vzhled, není zřejmé, že jsou editovatelná.	N5

3. Kroky 4 až 6
   Nebyly zaznamenány žádné problémy.

D.4 UC4 - Přiřadit štítek k albu

Q0 - Uživatel chce přiřadit štítek k existujícímu albu.

1. Kroky 1 až 3

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q2	Ne (P4)	Tlačítko pro editaci má nelogické umístění a je moc malé.	N6

2. Kroky 4 až 8

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q3	Ne (P2, P3)	Tlačítko pro uložení je překryto nabídkou pro zvolení štítku, viditelné je až po kliknutí mimo.	N7

D.5 UC5 - Použít generování playlistu

Q0 - Uživatel chce využít systémové generování playlistu.

1. Kroky 1 až 6

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q1	Ne (P1)	Generování by mělo fungovat i s prázdným playlistem.	N8

D.6 UC6 - Zadat podobné entity

Q0 - Uživatel chce zadat podobnost u dvou entit v grafovém pohledu.

1. Kroky 1 až 4
   Nebyly zaznamenány žádné problémy

2. Kroky 5 až 8

Otázka	Odpověď (participant)	Popis	Nález
Q1	Ne (P2)	Hrana se musí přetáhnout přímo na uzel, nefunguje přetažení na popisek.	N9
Q1	Ne (P3,P4)	V případě většího množství umělců je velmi složité najít konkrétního umělce.	N10

D.7 UC7 - Zkontrolovat nové skladby

Q0 - Uživatel chce v aplikaci zkontrolovat změny ve složce s hudbou a importovat nové skladby.

1. Kroky 1 až 5
   Nebyly zaznamenány žádné problémy.

D.8 UC8 - Uložit aktuální playlist

Q0 - Uživatel chce uložit hlavní playlist pro pozdější využití.

1. Kroky 1 až 4
   Nebyly zaznamenány žádné problémy.

D.9 UC9 - Přehrát uložený playlist

Q0 - Uživatel chce přehrát uložený playlist.

1. Kroky 1 až 4

Otázka	Odpověď (participant)	Popis	Nález
Q3	Ne (P1)	Zvýraznění aktuálně vybraného playlistu není dostatečně jasné.	N11

2. Kroky 5 až 7
   Nebyly zaznamenány žádné problémy.

D.10 UC10 - Přiřadit hudební žánr

Q0 - Uživatel chce přiřadit ke zvolenému albu existující hudební žánr.

1. Kroky 1 až 5
   Nebyly zaznamenány žádné problémy.

D.11 UC11 - Použít k editaci metadat agregaci z internetu

Q0 - Uživatel chce použít pro anotaci alba metadata z internetu.

1. Kroky 1 až 3
   Nebyly zaznamenány žádné problémy.

2. Kroky 4, 5

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q1	Ne (P4)	Podoba doporučení jako klasické odkazy evokuje přesun na jinou stránku, uživatel si není jistý, co kliknutí způsobí.	N12
Q3	Ne (P1)	Zpracování doporučených hodnot není přehledné.	N13

3. Kroky 6 až 8

Otázka	**Odpověď (participant)**	**Popis**	**Nález**
Q3	Ne (P2)	Chybí potvrzení o použití hodnot.	N14

D.12 UC12 - Zobrazit grafový pohled pro umělce a alba

Q0 - Uživatel chce zobrazit a procházet grafový pohled.

1. Kroky 1, 2
   Nebyly zaznamenány žádné problémy