Sem vložte zadání Vaší práce.

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

Diplomová práce

Sdílený prioritní webový playlist

Bc. Martin Horský

Vedoucí práce: Ing. Martin Půlpitel

5. května 2015

Poděkování

Děkuji především své rodině za podporu během studia i při psaní této práce. Dále děkuji zaměstnancům a samotné společnosti Ackee, s. r. o. za umožnění testování použití výsledné aplikace v reálném prostředí. V neposlední řadě děkuji Ing. Martinovi Půlpitlovi za vedení této práce.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 5. května 2015 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Horský, Martin. *Sdílený prioritní webový playlist.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Obsahem práce je návrh webové aplikace, která slouží pro přehrávání hudby formou sdíleného webového playlistu. Výsledná aplikace řeší prioritu přehrávání v uzavřené skupině uživatelů. Podstatnou částí práce je řešení architektury aplikace jako tzv. single-page aplikace s využitím komunikace technikou REST.

Klíčová slova aplikace, hudba, web, sdílení, skupina, priorita, přehrávání, API, Javascript, PHP, REST

Abstract

The content of this thesis is design of the web application that is used for playing music through shared web playlist. The application solves playback priority in a closed user group. Substantial part of this work is designing an application architecture as single-page application using REST communication technique.

Keywords application, music, web, sharing, group, priority, playback, API, Javascript, PHP, REST

Obsah

Seznam obrázků

Seznam zdrojových kódů

Úvod

Poslech hudby je pro mnohé lidi nedílnou součástí každodenního života. Hudbu poslouchá snad každý a to buď cíleně například při odpočinku, nebo jako doplněk k jiným činnostem, mezi které patří sport, studium, práce nebo oslavy.

Hudba je součástí kultury a někdy také životního stylu. Existují lidé, kteří mají poslech hudby jako svůj hlavní koníček. Určitě by se daly nalézt další příklady potvrzující velký význam hudby.

Poslech hudby je obecně individuální záležitostí. To především kvůli rozdílným preferencím lidí z hlediska žánru. Pro individuální poslech hudby existuje celá řada řešení počínaje domácí Hi-Fi soustavou a končeje kapesním přehrávačem.

Skupinový poslech velkého množství lidí definují koncerty, festivaly, společenské události, přehlídky, sportovní události nebo rádio. Význam hudby, v každé z jmenovaných akcí je jiný, platí však, že reprodukce hudby je většinou v režii jednotlivce nebo vybrané skupiny a návštěvník – individuální posluchač – nemůže pochopitelně do produkce zasáhnout. Rozdílná situace nastává u poslechu hudby v uzavřené skupině. Členové chtějí mít kontrolu nad tím „co zrovna hraje“.

Zaměřením této práce je vyvinout řešení pro pohodnější poslech hudby uzavřené skupiny uživatelů. Dalším předmětem je prozkoumat a popsat možnosti, které taková skupina posluchačů v současnosti používá a má k dispozici.

Další částí práce je analýza problémů, které se mohou vyskytnout při poslechu hudby ve skupině, v čele s prioritou přehrávání.

Cílem práce je vytvoření webové aplikace, která umožňuje uzavřené skupině uživatelů zařadit do společného seznamu hudbu a tu následně přehrávat.

Kapitola 1

Rešerše a průzkum

Pro budoucí správný návrh celé aplikace je vhodné znát způsoby poslechu hudby obecně. Chování a zvyklosti uživatelů mohou posloužit pro následný správný návrh aplikace. První kapitola se proto zabývá popisem historických a moderních způsobů poslechu hudby.

Pochopení uživatele je důležité především při návrhu rozhraní a způsobu ovládání výsledné aplikace. Pokud aplikace nefunguje, jak uživatel očekává, je velká pravděpodobnost, že ji nebude používat. Cílem je samozřejmě se tomu vyhnout.

Vhodnou inspirací z „reálného“ světa mohou být nejrůznější zařízení pro reprodukci hudby od rádií až po kapesní hudební přehrávače.

1.1 Historie a současnost poslechu hudby

Předmětem práce není bádání v minulosti a nebylo by tak na místě detailně popisovat historii hudby jako samotné. Nicméně je vhodné pozastavit se nad zajímavými fakty, které ve sledovaném tématu ovlivnily lidstvo a mohou tak posloužit pro budoucí popis požadavků na aplikaci z hlediska uživatelských potřeb a zvyklostí a které by tak měly mít vliv na vzhled výsledné webové aplikace a její ovládání.

1.1.1 Vývoj poslechu hudby

V dřívějších dobách lidé mohli poslouchat hudbu pouze živou. Nejstarším „nástrojem“ je pochopitelně lidský hlas. Následují nálezy z pravěku, jako jsou trubky z rohů, paličky z kostí nebo jednoduché flétny, které dokazují význam hudby již z dávných věků.

1.1.1.1 Fonograf a gramofon

Pravým začátkem pro moderní způsob poslechu hudby byl však až roku 1877 Fonograf, který sestrojil vynálezce Thomas Alva Edison. Jedná se o první přístroj pro nahrávání a reprodukci hlasu, který po svém rozšíření po Americe a Evropě dominoval až do roku 1910. V článku „Historie záznamu zvuku“ [1] je uvedeno, že lidé považovali hlas znějící z fonografu za tajemný a často si mysleli, že osoba je skutečně schována někde v místnosti. Nevěřili tomu, že by voskový váleček s připojenou plechovou „troubou“ mohl sám mluvit.

Fonograf byl poději nahrazen gramofonem. Který, na rozdíl od fonografu nezaznamenával zvuk na váleček, ale na desky. V obou případech se jednalo o analogový záznam zvuku. Zatímco fonograf používal k záznamu změnu hloubky drážky na válečku tak gramofonová deska má hloubku konstantní a záznam se provádí vychýlením drážky do stran. To umožnilo snadno vyrábět kopie gramofonových desek. U fonografu se záznam kopírovat nezdařilo. Plochý tvar desky navíc později umožnil gramofonové desky již se záznamem ve velkých sériích zhotovovat lisováním.

Souboj mezi fonografem a gramofonem trval od 90. let 19. století až do 20. let 20. století, a byl ukončen zánikem Edisonovy továrny v roce 1929 [2]. Od té doby měl gramofon jakýsi monopol mezi nástroji pro přehrávání hudby.

Rok 1952 dokonce vznikla americké společnost Record Industry American Association (česky Americké asociace producentů desek). Ta standardizovala u svých členů jednu korekční křivku dnes známou pod zkratkou RIAA.

Díky celosvětovému uznání se mohl pod stát gramofon po druhé světové válce doslova fenoménem. Rozvoj materiálů umožnil výrobu desek z „vinylu“ neboli plastu, který má i díky obsahu sazí dobré kluzné vlastnosti (neopotřebovává tolik hrot přenosky jako původní tvrdý šelak) a zároveň je mnohem odolnější proti rozbití [3].

Gramofon byl až na konci 20. století částečně vytlačen digitálními přehrávači CD či DVD. Ještě před tím zaznamenalo lidstvo několik důležitých milníků, které je vhodné v krátkosti zmínit.

1.1.1.2 Rádio

Jedním z nich bylo masivní rozšíření rádia. Samotný přenos radiových vln byl znám již z dřívějších dob, ale až kolem roku 1920 začalo vysílat první komerční stanice, která začala pravidelně vysílat oznámení týkající se prezidentských voleb. Kvůli špatné kvalitě reprodukce nemohlo rádio v té době konkurovat deskám, ale i přes to nahrávací společnosti přispěchaly se smlouvami, které znemožnily největším umělcům s rádiem spolupráci. Toto úsilí omezovat obsah produkce v rádiích vedlo k rychlému rozvoji rádia z hlediska kvality zvuku a reprodukce. Prodeje nahrávek začaly klesat [4].

Do rádií se hudba jako hlavní obsah dostala až v 50. letech 20. století. Do té doby rádia vysílala živé vystoupení, komedie, pořady pro děti a zprávy

včetně zpráv o počasí. V té době vznikl termín pro lidi, kteří v rádiových stanicích uváděli a přehrávali populární hudbu z gramofonových desek – DJs (Disc jockey, česky Diskžokej).

1.1.1.3 Audiokazeta

Během doby úspěchu rádia, v polovině 30. let, němečtí inženýři vynalezli prakticky použitelnou magnetickou pásku pro záznam a byl představen první použitelný magnetofon. Brzo však vypukla válka, takže do USA se dostali až po ni.

Když v roce 1964 firma Philips představila svůj 30 minutový formát pro audiokazetu, založenou právě na magnetických páskách, a povolila kopírovat specifikaci ostatním továrnám. Umožnila tak vytvoření trhu s levným a přenosným řešením. Tento trh posílilo v roce 1979 představení zařízení Sony Walkman. Nastala tak revoluce, kdy se audiokazeta používala jako jediný formát dostupný v domácnosti, v autě a jako médium pro kapesní přehrávání.

1.1.1.4 Kompaktní disk

V té době rozmachu Walkmanu firmy Philips and Sony oznámily práci standardu kompaktního disku (angl. compact disc, zkratka CD). Ten po jeho uvedení v roce 1982 zažil raketový vzestup. Alba se vydávali na všech třech typech nosičů (gramofonová deska, ozvučená kazeta a kompaktní disk), ale po roce 1988, kdy kompaktní disky překonaly v prodejích gramofonové desky, byla na začátku 90. let výroba gramofonových desek definitivně v celosvětovém měřítku ukončena.

1.1.1.5 MP3 a internet

Rok 1990 přinesl vznik formátu MP3 (Moving Picture Experts Group-1, Layer-3), jež se spolu v kombinaci digitálního zvuku, internetu a pozdějších MP3 přehrávačů stal fenoménem v oblasti přehrávání hudby.

Zajímavým rokem z pohledu této práce byl rok 1995. V té době společnost RealNetworks úspěšně spustila první možnost streamování hudby založenou na formátu RealAudio. V porovnání v té době dlouhého stahování souboru s písničkou se streamované stahování stalo velmi populární i naproti počáteční špatné kvalitě.

První populární peer-to-peer (P2P), která vznikla v roce 1999 pro sdílení hudby mezi uživateli, síť Napster, byla po dvou letech i přes svojí značnou popularitu kvůli úspěšné žalobě organizace RIAA zrušena.

Na přelomu tisíciletí vzniky kapesní přehrávače nazývané jako MP3 přehrávače, které v té době nahradily do té doby používané Walkmany užívané pro přehrávání audiokazet a novější Diskmany pro přehrávající CD. Nevýznamějším zařízením v této oblasti byl v roce 2001 představený přehrávač iPod od společnosti Apple.

1.1.1.6 Prodej hudby přes internet

Byla to znovu firma Apple, která se zapsala do historie v oblasti hudby, když v roce 2003 spustila do dnešního dne nejúspěšnější on-line obchod s hudbou – iTunes Store. Během prvního roku prodal Apple 70 milionů písniček [5] za cenu 0,99 dolarů jedné písničky.

Tento trend prodeje a následného poslechu zakoupených skladeb vydržel až do současnosti. Koupenou hudbu si každý uživatel může pro své účely uložit do každého ze svých zařízeních (hudební přehrávač, mobilní telefon, osobní počítač) a poslouchat.

V současné době existuje několik serverů, kde je možné hudbu nakupovat počínaje průkopnickým iTunes Store od Apple, přes populární Google Play, Amazon Music, Xbox Music od Microsoftu až po menší služby jako je eMusic nebo Rhapsody.

1.1.1.7 Streamování hudby přes internet

Současným trendem v oblasti poslechu hudby jsou služby zaměřené na tzv. streamování hudby. Tyto služby dovolují za určitý měsíční poplatek nebo dokonce v kombinaci s reklamou zdarma přístup k obrovské knihovně hudby. Je tak možné poslouchat hudbu kdykoli a kdekoli.

Streamovaná hudba je dalším krokem k očekávané „cloud budoucnosti“, kdy v uložišti disku počítače nebo v mobilu nemusí být nic. Stačí aplikace a přístup k internetu. Přehrávání potom probíhá pomocí webového prohlížeče a není za potřebí specializovaný program nebo software na přehrávání hudby.

Kromě nabídky na vyžádání (angl. on demand) služby často nabízejí playlisty doporučených nebo často přehrávaných skladeb a nejrůznější žebříčky.

Výhodou je již jednou zmiňovaná dostupnost neomezeného uložiště bez přímého rizika ztráty své sbírky, které může nastat v případě hudby uložené na disku počítače.

Existují samozřejmě nevýhody streamovacích služeb. Jednou z nich přinášejí samotní umělci. Některé kapely se digitální distribuci zkrátka vyhýbají, jiné mají naopak příliš slabé zastoupení. Často se u těchto služeb projevuje regionální omezení. Z tohoto důvodu je spousta alb dostupných jen v některých zemích. Nemluvě o situaci, kdy je album dostupné jen v určité službě. Další nevýhodou může být fakt, že předplacení služby neznamená, že hudbu vlastníte.

Díky popularitě streamování hudby přes internet existuje mnoho služeb a další vznikají. Jmenovitě se jedná o Spotify, Deezer, Grooveshark, Rdio, Pandora nebo Last.fm.

1.2 Poslech hudby ve skupině

Do této chvíle práce nerozlišovala, zda je hudba „konzumována“ jednotlivcem či nikoli. Poslech hudby ve skupině s sebou přináší především různé problémy, kterými se mimo jiné tato část práce zabývá.

Obsahem následujících odstavců je také popis aktuálních možných řešení skupinového poslechu.

1.2.1 Prostředí a uživatelé

Následující odstavce popisují konkrétní uživatelské skupiny, jejich chování z pohledu poslechu hudby. Dále jsou popsány problémy, které nastávají při poslechu hudby v uzavřené skupině uživatelů. Text se zaměřuje konkrétně na dvě prostředí.

Kancelář V kanceláři pobývají lidé, které spojuje především pracovní činnost. Tato skupina lidí poslouchá společnou hudbu jako doplněk při práci. Vhodně vybraná hudba může mít povzbuzující nebo uklidňující vliv a může sloužit jako clona nežádoucího hluku. Vynikajícím příkladem prostředí může být otevřený kancelářský prostor, ve kterém spolupracují členové projektového týmu.

Soukromá párty Oslava narozenin, společenské události, posezení s přáteli jsou výborným popisem prostředí uzavřené skupiny lidí, jejichž společná chvíle je doplněná poslechem hudby. Význam hudby v závislosti na průběhu večera u takové skupiny může být měněn. Samotný poslech se spolu s tancem může stát primární činností dané skupiny.

1.2.2 Problémy poslechu

Uzavřené skupiny uživatelů při poslechu narážejí na několik problémů, které mohou sloužit jako prostor pro zlepšení v navrhované aplikaci.

Žánr Každý uživatel ve skupině má jiný hudební vkus. Někteří lidé mají v tomto směru jasné preference a nejsou tolerantní k jiným žánrům. Jiní uživatelé nemají vyhraněný žádný oblíbený žánr a jejich preference se mění v závislosti na aktuální popularitě hudebních skladeb.

Tento problém je pozorován především ve skupině z kancelářského prostředí. Členy skupiny jsou lidé, jež spojuje především práce, a každý má jiný pohled na hudbu a zároveň jiný životní styl, který tento pohled může přímo ovlivňovat.

Přepínání Jeden z největších pozorovaných problémů bylo „přepínání“ právě přehrávané skladby. Jedná se o situaci, kdy se vypne nedokončené přehrávání a pustí se nová skladba. Tento efekt má za následek rozhořčení posluchačů.

Přeskakování Jeden z podobných problému, mající stejný efekt jako předchozí, je „přeskakování“ skladeb. Tedy změna nebo nedodržení pořadí plánovaného přehrávání. Velmi negativní ohlas má situace přeskočení skladeb zkombinované s přepnutím aktuální skladby.

Priorita Především v prostředí párty spolu v kombinaci s alkoholem je velkým problémem priorita výběru hudby. Kdo a kdy bude pouštět „svoji“ skladbu. Změna osoby, která se stará o aktuální playlist – v oblasti přehrávání hudby zvané jako DJ – může mít za následek všechny předchozí vyjmenované problémy.

1.2.3 Možnosti poslechu ve skupině

Tato část práce popisuje některé pozorované způsoby, jakými uživatelé ve skupině pouštějí a poslouchají hudbu. Předmětem popisu jsou výhody a nevýhody takového řešení.

Tyto metody se také mohou lišit v rozdílném schématu připojení výstupního zařízení (reproduktory) ke zdroji přehrávání.

První kategorií jsou metody, které fungují na bázi soukromé databáze hudby, druhou kategorii zahrnují možnosti poslechu s využitím internetových služeb jako zdroje pro písničky.

1.2.3.1 Přehrávač

V obou sledovaných prostředích (v kanceláři a na párty) je k reproduktorům připojen jeden počítač, který ze své vlastní databáze pouští hudbu. Databází je pevný disk se soukromou sbírkou omezeného počtu dostupných hudebních souborů.

Jako software pro přehrávání je použit program přehrávače dostupného na dané platformě. Skladby vybírá a postupně řadí seznamu jedna osoba na žádosti ostatních, případně sám DJ určuje, co se bude hrát.

Tento způsob poslechu má nevýhodu v omezené dostupnosti skladeb. Určitá nevýhoda je v omezených možnostech režie posluchačů.

1.2.3.2 YouTube

YouTube, internetový server primárně určený pro sdílení videosouborů všeho druhu, díky svému obsáhlému množství hudebních klipů bývá využíván jako zdroj pro hudební soubory. Tento přístup je pozorován hlavně v prostředí soukromých oslav, kdy je k reproduktorům připojen jeden počítač, který přehrává hudbu z webového prohlížeče.

Výhodou je prakticky neomezený počet dostupných písniček. Dalším plusem je obecná známost služby, jejíž obsah uživatelé znají. Tyto výhody spolu s přítomností vyhledávání dělají z YouTube silný nástroj pro nalezení a následné přehrání požadovaných hudebních skladeb.

Nevýhodou je chybějící snadná organizace jednotlivých skladeb do seznamů přehrávání. Tato možnost je dostupná po přihlášení, což bohužel v případě neomezeného přístupu k počítači není ideální.

1.2.3.3 AirPlay

AirPlay je protokol od společnosti Apple pro bezdrátové sdílení multimédií mezi zařízeními. Takto může kdokoli, jehož zařízení protokol podporuje (jedná se především o mobilní telefony iPhone a počítače Mac), posílat zvuk do jiného zařízení, které je připojeno k reproduktorům.

Na rozdíl od předchozích dvou způsobů se jedná spíše o schéma zapojení, které umožňuje reprodukci skupině uživatelů. Zdrojem pro hudbu tak může jak osobní hudební sbírka uživatele, tak internetová služba.

Tento přístup byl pozorován v kanceláři vývojářského týmu. Jako zařízení připojené k reproduktorům byl připojen mikropočítač Raspberry Pi, který sloužil jako přijímač protokolu AirPlay.

I přesto, že tento přístup vypadá jako řešení skupinového poslechu, opak je pravdou. Připojeno ke vzdálenému přijímači může být vždy pouze jeden klient. Ten ani nezabrání připojení jiného zařízení a tím přerušení případného přehrávání.

Nevýhodou také zůstává nemožnost přímého zásahu do seznamu přehrávání ostatními posluchači.

1.2.3.4 plug.dj

Jedná se o službu, která skupinový poslech definuje jako svůj primární účel. Samotná stránka se definuje jako „služba věnována pozitivně smýšlejícím mezinárodním komunitám pro sdílení a objevování hudby“ [6]. Plug.dj má 4,7 miliónů registrovaných uživatelů a je přeložen do 47 jazyků [7].

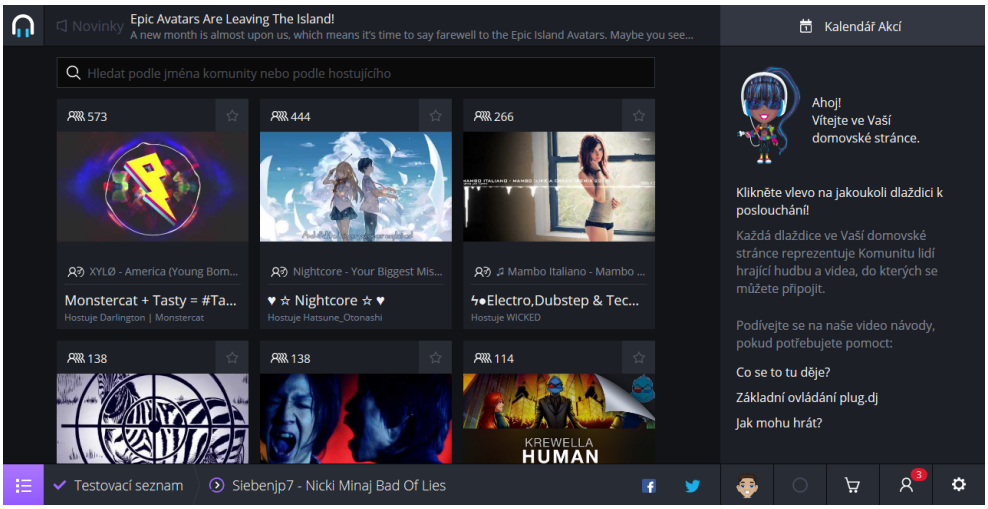
Plug.dj obsahuje obrovské množství místností, nazvané jako komunity, do kterých se lze připojit. Uvnitř uživatel může poslouchat, kterou přehrávají ostatní uživatelé. Sám se může aktivně podílet na přehrávání zařazením se na frontu DJ. Až na uživatele přijde řada, může přehrávat vybranou hudbu z YouTube nebo SoundCloud.

Během přehrávání je možné pozitivně nebo negativně hlasovat pro aktuální písničku. Za tyto aktivity a zároveň za samotné využívání služby dostává uživatel body, které může využít ke změně svého avatara nebo k odemčení pokročilých funkcí chatu, který je v každé komunitě přítomný.

Celá stránka má charakter interaktivní hry o čemž svědčí velké množství již zmíněných avatarů (postaviček připojených uživatelů), které během pře-

hrávání tancují před postavičkou aktuálního DJ. Dostupný chat je ve větší komunitě plný odkazů na spam a míjí se s účelem.

Mezi další nevýhody patří průběžné odemykání funkcí, což má za následek značně omezenou použitelnost pro nově příchozího uživatele. Služba se zdá být také značně nepřehledná (obrázek 1.1). K dispozici jsou pro nové uživatele vysvětlující videa o ovládání a účelu jednotlivých součástí služby, což je určitě vítané, avšak potvrzuje to fakt o celkové složitosti a nepřehlednosti celého procesu poslechu hudby.



Obrázek 1.1: Webová aplikace plug.dj

Plug.dj je vhodnou službou pro **otevřenou** skupinu uživatelů. Pro uzavřenou skupinu lidí nabízí zbytečné funkce, které soupeří s primárním účelem o uživatelovu pozornost

1.3 Shrnutí

Z předchozího textu vyplývá, že v minulosti existovaly pro poslech hudby omezené možnosti. Problematický byl nejen záznam, který byl po dlouhá léta omezován technologickou náročností a tudíž vysokou cenou, ale také samotné přehrávání. Zařízení pro poslech se měnila.

Ukázalo se, že způsoby poslechu hudby se měnily a mění spolu s vývojem technologií. Velké změny zaznamenal prodej a poslech především s příchodem internetu. Vznikl nový trh, který má za následek převahu digitální hudby bez existence fyzických nosičů.

Samotný poslech hudby řeší uzavřené skupiny uživatelů formou režie jednotlivce, který hudbu vybírá a řadí do playlistu. Tak je tomu se ve všech prozkoumaných možnostech skupinového sdílení hudby.

Kapitola 2

Analýza požadavků

Tato kapitola se zabývá popisem tzv. případů užití (angl. use cases) aplikace. Některé požadavky vycházejí ze zadání, jiné z přechozí kapitoly. Poslední kategorií jsou požadavky na systém z důvodu budoucího rozšíření a použití. Druhá část této kapitoly popisuje funkční a nefunkční požadavky na implementovaný systém.

2.1 Základní otázky a popis aplikace

Před samotným popisem požadavků je vhodné odpovědět si na základní otázky, které pomohou definovat aplikaci jako celek. Odpovědi na otázky se opírají o zadání této práce a pochopení ze strany autora práce. Některé otázky mohou být diskutovány později v návrhu.

Jaký je základní popis aplikace? Webová aplikace, pomocí které může skupina uživatelů přidávat hudbu do sdíleného playlistu a tu pak přehrávat.

Co znamená, že se bude jednat o webovou aplikaci? Aplikace bude zobrazitelná pomocí webového prohlížeče přes internetovou síť. Bude tedy dostupná odkudkoli. Další výhody tohoto řešení jsou diskutovány v nadcházejících kapitolách.

Kdo bude aplikaci používat? Aplikaci budou používat uzavřené skupiny lidí. Příkladem mohou být spolupracovníci v kanceláři, účastníci oslav a jiných společenských událostí nebo spolubydlící.

Co bude aplikace umět? Hlavním účelem aplikace bude možnost zařazení vybrané písně z nějakého zdroje do společného playlistu a přehrávání této fronty písní.

Jak bude řešena priorita uživatelů? Priorita v řazení písní bude řešena variabilním bodovým hodnocením. Uživatel dané skupiny, který má více bodů, může zařadit svou písničku před ostatní. Řazení písní probíhá podle bodů uživatelů, které uživatelé získají z různých zdrojů. Způsoby získání bodů aplikace podrobně neřeší.

Pro jaký účel aplikace nebude určena? Aplikace nebude přehrávačem hudby. Nebude možno ručně přeskakovat písničky zařazené do fronty přehrávání. Dále nebude umožněno pozastavení přehrávání.

Pro koho není aplikace určena? Uživatel, kterému by scházely funkce jako je opakování písní, náhodné přehrávání nebo již zmíněné přetáčení a pozastavení využije standardního přehrávače.

Má aplikace nějaký název? Zadání práce název aplikace neobashuje. Nicméně název bude diskutován a vymyšlen před návrhem aplikace jako tzv. *kódové jméno*, jehož vznik bývá častý při vývoji softwaru.

2.2 Případy užití

Případy užití popisují interakci uživatele s aplikací. Případy užití vždy popisují cíle a chování systému z pohledu uživatele. Jedná se o abstraktní popis který nezahrnuje technické řešení ani popis uživatelského rozhraní.

1. **Volba fronty přehrávání**

   Aktér:
   Přihlášený uživatel

   Popis:
   Uživatel zvolí frontu přehrávání, kterou chce poslouchat.

   Scénář:

   1. Systém poskytne obrazovku pro volbu fronty přehrávání.
   2. Uživatel zvolí frontu přehrávání.
   3. Systém nastaví frontu přehrávání jako aktuální a zobrazí její seznam písní.

2. **Přidání písničky do seznamu**

   Aktér:
   Přihlášený uživatel

Popis:
Uživatel zařadí píseň do aktuálního seznamu.

Scénář:

1. Systém poskytne obrazovku, na které bude možné přidat píseň.
2. Uživatel vyhledá nebo zadá adresu písně, kterou chce přidat.
3. Systém zobrazí výsledek dotazu s možností potvrdit přidání.
4. Uživatel potvrdí přidání písně.
5. Systém provede přidání písně a zobrazí uživateli potvrzení o proběhlé akci.
6. Systém zobrazí seznam písní spolu s nově přidanou písní.

3. **Poslech seznamu přehrávání**

Aktér:
Přihlášený uživatel

Popis:
Uživatel poslouchá hudbu z aktuálně zvolené fronty.

Scénář:
Scénář navazuje na případ užití 1. Volba fronty přehrávání

1. Systém zobrazí informaci o aktuálně přehrávané písni.
2. Systém reprodukuje hudbu.

4. **Zesílení nebo zeslabení přehrávání**

Aktér:
Přihlášený uživatel

Popis:
Uživatel změní hlasitost aktuálně přehrávané hudby.

Scénář:

1. Systém poskytuje možnost změnit hlasitost hudby.
2. Uživatel změní hlasitost přehrávání.
3. Systém změní hlasitost reprodukované hudby.

5. **Ohodnocení přehrávané písně**

Aktér:
Přihlášený uživatel

Popis:
Uživatel ohodnotí aktuální píseň, která je přehrávána.

Scénář:

1. Uživatel ohodnotí aktuálně přehrávanou píseň.
2. Systém zareaguje přičtením hlasu k dané písničce.

2.3 Funkční požadavky

Funkční požadavky vycházejí z případů užití. Detailněji popisují požadavky na fungování systému po provedení akcí sepsaných v textu případů užití.

Zahrnují technické detaily a požadavky na systém z pohledu vývojáře. Cílem je upřesnit očekávané chování systému a usnadnit tak budoucí implementaci.

Následující soupis požadavků je rozdělen do logických spolu souvisejících funkčních celků.

1. **Přihlášení**

   Formulář pro přihlášení

   Popis:
   Systém umožní přihlášení uživatele. Požadováno bude zadání uživatelského jména a heslo. Pro tyto účely budou k dispozici dvě formulářové pole s popisem.

   Po odeslání formuláře systém zkontroluje, zda zadané údaje souhlasí se záznamy v databázi. V případě úspěchu je uživatel přihlášen a přesměrován na úvodní obrazovku. V případě neúspěchu je uživatel o té situaci informován a je mu nabídnuta možnost opravy údajů.

   Zapamatování přihlášení

   Popis:
   Dalším prvkem přihlašovacího formuláře bude zaškrtávací políčko (checkbox) pro možnost zapamatování přihlášení uživatele. Je-li pole během přihlášení zaškrtnuté, systém uživatele neodhlásí po ukončení sezení prohlížeče. Pole bude ve výchozím nastavení nezaškrtnuté.

2. **Výběr fronty přehrávání**

Seznam front

Popis:
Systém zobrazí úvodní stránku se seznamem front přehrávání, které jsou dostupné pro aktuálně přihlášeného uživatele. Každý prvek seznamu obsahuje název fronty a aktuální přehrávanou písničkou. Kliknutím na dostupné tlačítko se uživatel dostane na stránku přehrávání.

3. **Seznam přehrávání**

Přehrávání

Popis:
Po načtení stránky systém začne přehrávat hudbu. K dispozici je informace o aktuálně přehrávané skladbě – název skladby, ovládání, průběh přehrávání – a další informace – jméno uživatele, který zařadil píseň do seznamu a zdroj písničky.

Fronta písniček

Popis:
Systém zobrazuje uživateli písničky zařazené ve frontě. První skladba v seznamu je následující přehrávaná skladba. Písničky jsou řazeny podle priority uživatelů.

4. **Přidání písničky do seznamu**

Popis:
Systém umožní zadat do formulářového pole název písničky nebo interpreta. K dispozici bude tlačítko pro odeslání dotazu. Uživatel dotaz odešle. Systém zobrazí výsledek hledání vyhovující danému dotazu jako seznam písniček. Uživatel zvolí jeden z výsledků kliknutím na k tomu určené tlačítko. Skladba je zařazena do seznamu přehrávání a uživatel je přesměrován na stránku aktuálního přehrávání.

5. **Změna hlasitosti**

Popis:
Systém umožní změnit hlasitost přehrávání kliknutím na ovládací prvky k tomu určené. Aplikace bude podporovat dva režimy hlasitosti – úplné ztlumení a zesílení na maximum.

6. **Ohodnocení přehrávané písně**

   Popis:
   Systém umožní dát hlas za aktuálně přehrávanou píseň. Tento hlas je přičten uživateli jako bod a zvýší se mu priorita.

7. **Odhlášení**

   Popis:
   Systém umožní odhlášení uživatele. Odstraněna budou všechna data sezení. Uživatel bude o této skutečnosti informován a přesměrován na stránku přihlášení.

2.4 Nefunkční požadavky

Stejně jako případy užití a funkční požadavky také nefunkční požadavky vycházejí především ze zadání práce.

1. **Webová aplikace**

   Aplikace bude zobrazitelná ve webovém prohlížeči. Jednat se bude o „Web 2.0“ aplikaci, jejíž hlavní definicí je obsah tvořený uživatelem a dynamické fungování.

2. **Jednoduché a přehledné rozhraní**

   Rozhraní aplikace bude jednoduché, přehledné a soustředěné na hlavní účel – přehrávání hudby. Důraz by měl být kladen na použitelnost.

3. **Mobilní zobrazení**

   Aplikace bude zobrazitelná a funkční na mobilních zařízeních s menším displejem a nižším výkonem.

4. **Rozšiřitelnost**

   Aplikace bude navržena tak, aby byla snadno rozšiřitelná o nové funkce.

5. **Komunikace pomocí technologie REST**

   Ke komunikaci s ostatními službami bude aplikace využívat technologie REST.

6. **Využití jazyka PHP**

   Logika aplikace bude naprogramována v jazyce PHP. Příkladem může být využití pro výpočty, řazení nebo samotné získávání dat z databáze.

   Podporována bude poslední stabilní verze PHP – 5.6.

7. **Podpora moderních prohlížečů**

   Podporovány budou všechny hlavní desktopové webové prohlížeče včetně vybraných mobilních ve své aktuální verzi. Konkrétní verze jsou sepsány v následujícím seznamu.

   - Google Chrome – 42
   - Mozilla Firefox – 37
   - Safari – 8
   - Opera – 29
   - Internet Explorer – 11
   - Opera – 29
   - iOS Safari – 8.3
   - Android Browser – 40

Kapitola 3

Výběr streamovací služby

Výsledná aplikace by měl používat jako zdroj písní existující hudební službu, která poskytuje veřejné rozhraní pro získání hudebních skladeb.

Její výběr závisí na několika faktorech, nad kterými je vhodné se pozastavit, popsat a brát v úvahu u samotného výběru.

Přítomnost API Přítomnost rozhraní s možností dotazování skrz webové dotazy je zásadní pro využití dané služby jako zdroje pro aplikaci sdíleného prioritního playlistu. Přes API volání by měly být dostupné informace jako je název nebo délka písničky.

Možnost streamování Služba musí poskytovat způsob, jakým je možné přehrávat písničky ve výsledné webové aplikaci. K dispozici by mělo být rozhraní pro ovládání streamu nebo přehrávače vloženého na stránku.

Plná délka Není možné zvolit službu, která nabízí pouze krátké ukázky písniček a nepodporuje možnost přehrání v plné délce.

Množství písniček Služba by měla poskytovat velké množství výběru písní různých žánrů od umělců, jež tvořili v minulosti až po skladby od umělců současných.

Nově vydané písně Nově uvolněné alba a samostatně vydané hudební nahrávky (tzv. singly), by měly být co nejdříve v dané službě dostupné. Požadavek je především na písničky, které jsou v určité době oblíbené a vyskytují se v hitparádách.

Vysoká kvalita Přehrávání by mělo být dostupné v co nejlepší poslouchatelné kvalitě a nahrávka by neměla trpět příliš velkou kompresí.

Originální verze Písničky by měly být k dispozici ve své originální verzi nahrané přímo umělcem nebo vydavatelem.

Služba zdarma V ideálním případě by bylo vhodné využít služby, která nabízí poslech respektive dostupnost písní zdarma.

Nepřítomnost reklam Vhodná služba by neměla spouštět reklamy na začátku nebo během přehrávání.

Oblíbenost služby Služba by měla být obecně známa pro uživatele aplikace. V ideálním případě by uživatel měl službu jako samotnou používat, aby znal obsah, který mu poskytuje.

3.1 Analýza služeb

Následující odstavce popisují nejznámější služby, jejich výhody a nevýhody. Zároveň každá služba prozkoumána zda a v jaké míře podporuje požadované funkce, nebo zda má výše popsané vlastnosti. Zásadním aspektem je podpora nějakého způsobu streamování nebo přehrávaní skladeb v plné délce.

3.1.1 Spotify

Spotify je internetová služba, která nabízí poslech hudby. V současnosti nabízí přes 30 milionů skladeb [8]. Se svými 60 milióny aktivních uživatelů patří mezi nepoužívanější služby k poslechu hudby vůbec.

Spotify poskytuje pro vývojáře webové API. Toto API je popsáno na adrese `https://developer.spotify.com/web-api/`. Založeno je na principech REST, data odpovědí jsou dostupná ve formátu JSON. API umožňuje získat informace o umělcích, albech, hudebních stopách přímo z databáze (katalogu) Spotify. Po autorizaci zároveň umožňuje dotazy na uživatelská data zahrnující například vytvořené playlisty nebo hudbu uloženou v knihovně „Moje hudba“.

Spotify bohužel pomocí API **nepodporuje** žádný ze způsobů, jak hudbu streamovat [1]. Po dotazu na koncovou URL písničky je v odpovědi k dispozici pouze klíč `preview_url`, jehož hodnota je odkazem na 30 sekundovou ukázku hudební stopy. Není proto možné využít Spotify jako zdrojovou službu.

3.1.2 Deezer

Deezer je přímým konkurentem Spotify. V počtu aktivních uživatelů strádá (16 milionů) avšak co do počtu nabízených písniček mu patří první místo s 35 milióny dostupných skladeb [9].

[1] Provdou je, že Spotify tuto skutečnost na oficiálních stránkách skrývá, ale je možné se dohledat alespoň odpovědi na internetu `http://stackoverflow.com/questions/24705253/play-full-spotify-track-inside-my-own-website-using-spotify-web-api`

Deezer poskytuje jednoduché API pro procházení katalogu písní. Toto API umožňuje prohlížení, vyhledávání a další operace nad velkým množstvím objektů jako jsou například alba, umělci, písničky nebo uživatelé. API umí komunikovat ve třech formátech – JSON, XML, PHP. Celé API je opravdu jednoduché a dokumentace dostatečná.

Pro přehrávání je dostupné Javascript SDK, které vloží do stránky přehrávač. Do tohoto přehrávače je možné načíst hudební stopu, celé album nebo řadit písně do fronty. Tento přehrávač lze programově ovládat (pozastavit, přeskočit, nastavit hlasitost).

Deezer rozlišuje několik uživatelských úrovní, od kterých se odvíjí práva na přehrávání, které jsou skrz API a SDK poskytována. Pravidla znázorňuje tabulka na obrázku 3.1 [10].

UNLOGGED USERS	FREEMIUM USERS	PREMIUM USERS	PREMIUM+ USERS
API : 30s. extract **Plugins** : 30s. extract	**API**: 30s. extract **Plugins** : 30s. extract	**API** : 30s. extract **Plugins** : 30s. extract	**API/Plugins** : Full Track / Unlimited

Obrázek 3.1: Pravidla přístupu k obsahu služby Deezer

Plná délka skladby je dostupná pro uživatele (vývojáře) s předplacenou službou PREMIUM+, která v době psaní této práce stojí 165 Kč měsíčně. V ostatních případech je k dispozici pouze část skladby jako 30 sekundová ukázka.

3.1.3 Grooveshark

Grooveshark je stejně jako předchozí služby určen pro online streamování hudby. V roce 2013 měl 30 miliónů registrovaných uživatelů a k poslechu k dispozici katalog o 15 miliónech skladeb.

Grooveshark již několik let čelí stížnostem mnoha předních vydavatelských společností mezi něž patří EMI, Universal Music Group, Sony Music Entertainment a Warner Music Groups pro porušování autorských práv a licenčních poplatků [11].

Grooveshark funguje na principu nahrávání hudby uživateli, na než je předána odpovědnost za tyto skladby. Výhodou je rychlá dostupnost nové hudby, kterou uživatelé nasdílejí. Bohužel není garantováno, že se jedná o hudbu z originálního zdroje a proto katalog Groovesharku obsahuje mnoho uživatelsky upravených, zkrácených hudebních stop různé kvality.

K dispozici pro vývojáře je robustní API, které podporuje prohledávání obsahu a práci s uživatelským účtem. K dispozici je oficiální PHP knihovna.

Pro integraci s vlastní službou poskytuje Grooveshark Flash přehrávač streamu, který lze ovládat Javascriptem pomocí požadavků na Groovshark API.

Pro využití Grooveshark API je nutné zažádat o přístup zadáním osobních údajů do formuláře s popisem použití. Žádost je přezkoumána zaměstnanci Groovesharku. V případě vyhovění je poskytnut přístupový klíč, který vývojář zasílá při každém požadavku na API.

3.1.4 YouTube

YouTube, na rozdíl od předchozích služeb, není primárně určen pouze pro poslech hudby. Služba YouTube vznikla jako portál pro sdílení videí. Jedná se o jednu z nejnavštěvovanějších stránek na internetu [2] a vůbec největší internetovou stránku, kde mohou uživatelé nahrávat, přehrávat a sdílet videosoubory. V současné době je vlastněna společností Google, která službu koupila v roce 2006.

YouTube, i přesto, že není primárně zaměřen na poslech hudby, obsahuje milióny hudebních videí a videoklipů. Tato skutečnost je podpořena tím, že na YouTube lze nalézt oficiální videoklipy písní nahraných přímo hudebními vydavatelstvími.

Za pozornost stojí zmínit společnosti Vevo, kterou v roce 2009 založily vydavatelství Universal Music Group a Sony Music Entertainment [12]. Ti na něm nabízejí obsah ze své produkce. Současně je většina obsahu na základě dohody se společností Google dostupná i prostřednictvím speciálního kanálu na videoserveru YouTube. Pod značkou Vevo nahrávají své videoklipy také samotní umělci.

Pro vývojáře poskytuje YouTube rozhraní pro vyhledávání, prohlížení, vytváření úpravu a mazání videí a playlistů. Dále také poskytuje potřebné nástroje pro vložení přehrávače na stránky jež lze ovládat voláním z kódu. Jedná se tak o způsob, jakým je možné bez omezení přehrávat hudební videa ve vyvíjené aplikaci sdíleného prioritního webového playlistu.

3.1.5 Ostatní

Existuje nepřeberné množství dalších služeb, které by bylo možné použít jako zdroj hudebních skladeb. Naprostá většina z nich ale žádné aplikační rozhraní (API) neposkytuje nebo v jejich použití zabraňuje nedostatek dostupných skladeb, jejichž malé množství by zcela jistě nepokrylo nároky uživatelů vyvíjené aplikace.

3.2 Vyhodnocení

Spotify, jakožto nepoužívanější službu zaměřená na online poslech hudby, z důvodu chybějícímu způsobu streamování hudby není možné využít. Omezením

[2] `http://www.alexa.com/topsites`

se se na služby, které nabízí vyžadované vlastnosti zdarma, se vyřazením Deezeru zúží výběr na dva potenciální zdroje – Grooveshark nebo YouTube.

Oba servery fungují na shodném principu, dodržující požadavky uvedené v americkém zákoně Digital Millennium Copyright Act (zkratka DMCA), které nařizují odstranění obsahu podléhajícímu autorskému právu v případně nahlášení takového obsahu.

YouTube narozdíl od Groovesharku obsahuje také licencovaný obsah, ať už díky společnosti Vevo, tak díky samotným umělcům, kteří nahrávají své hudební klipy na server.

Pro vývoj a využití API je nutné u serveru Grooveshark o přístupové údaje zažádat, kdežto Google (YouTube) umožňuje klíče vygenerovat. Je tak možné začít s vývojem bez zařizování přístupu.

Vzhledem k výhodám popsaným v předchozích odstavcích je YouTube nejlepší možnou platformou, která může sloužit jako zdroj písniček pro aplikaci, která je předmětem této práce.

Kapitola 4

Koncept a návrh aplikace

Kapitola popisuje požadavky na fungování systému. Upřesňuje zadání a stanovuje požadavky na výslednou aplikaci. Součástí je také návrh uživatelského rozhraní.

4.1 Název aplikace

Většina softwarových projektů dostává na začátku nebo během vývoje nějaké kódové jméno. Tento kódový název slouží jako odkaz na aplikaci během vývoje, využíván je v dokumentaci a nápomocný je pro běžnou komunikaci mezi členy vývojového týmu.

Často se stává, že kódové jméno zůstane výslednému projektu také po konci vývoje – ve chvíli, kdy je produkt vydán. Takový název by měl být jedinečný, jednoduchý, snadno zapamatovatelný, ale také originální a nezaměnitelný s jinou službou nebo aplikací.

Na název aplikace vyvíjenou v rámci této diplomové práce byl kladen důraz především na originalitu, zajímavost a jednoduchost. Možný budoucí rozvoj a podpora jiných jazykových mutací aplikace přidala požadavek na mezinárodní chápání a tedy nepřímé použití anglického jazyka.

Pro název „Sdíleného prioritního webového playlistu“ se nabízelo využít jedno nebo dvě anglická slova z množiny slov, které nějak souvistí s účelem aplikace – music (hudba), queue (fronta), list (seznam), you (ty), share (sdílet), play (hrát), priority (priorita) a song (píseň). Slovo *share* bylo vyřazeno hned na začátku, protože jeho význam ve spojení s hudbou by mohlo vyvolat milnou představu o podpoře internetového pirátství.

Kandidáty na pojmenování byly kombinace předchozích slov, z nichž převládal význam „hudební fronty“ a jeho různé varianty: SongList, MusicPlaylist, MusicQueue, myousicue, musiQ, mYousiQ, YouQue, musicQ, musiQU, musicYou, musicU, UQ

Pro aplikaci byl nakonec zvolen název **MusicQ**, který je spojením slov *music* a *queue* a využívá faktu, že *queue* se v angličtině vyslovuje stejně jako

samotné písmeno *Q*. Docíleno tak bylo požadované originality a pochopení významu.

4.1.1 Logo aplikace

Logo aplikace je logotypem názvu MusicQ s počátečním malým písmenem. Používá font Segoe UI ve své variantě Black Italic. Zajímavým prvkem loga je symbol ▶ ukrývající se uvnitř písmena Q, který se používá jako piktogram pro spuštění přehrávání hudby.

Obrázek 4.1: Logo aplikace MusicQ

4.2 Uživatelské rozhraní

Návrhu uživatelského rozhraní nebude z kapacitních důvodu věnován velký prostor. Obsahem této kapitoly nebude prototypování rozmístění prvků ve stránce a řešení bude ponecháno až na fázi implementace. To však neznamená nedodržování pravidel použitelnosti, která jsou kladena na správné fungování uživatelského rozhraní.

Tyto pravidla, nazývána jako Nielsenova heuristická analýza, si zaslouží pozornost a je vhodné je zmínit [13].

1. **Viditelnost stavu systému**

   Systém by měl informovat uživatele o tom, co dělá, a to pomocí zpětné vazby v rozumném čase.

2. **Shoda mezi systémem a realitou**

   Systém by měl komunikovat uživateli známými slovy, frázemi a pojmy. Měl by se držet konvencí reálného světa a zobrazovat informace v přirozeném a logickém pořadí.

3. **Uživatelská svoboda**

   Uživatelé často zkouší funkce metodou pokus–omyl a potřebují tak snadno bez složitého postupu možnost východiska z nechtěného stavu. Např. podporou akce zpět.

4. **Shoda s použitou platformou a obecnými standardy**

   Systém by neměl používat odlišná slova, chování nebo akce než je tomu zvykem u dané platformy. Je nutné dodržovat konvence, tak aby uživatel nebyl překvapen.

5. **Prevence chyb**

   Je lepší vyhnout se zobrazení chybové hlášky pomocí eliminace složitějších postupů, pomocí vysvětlení situace nebo za pomoci výběru volby před potvrzením uživatelské akce.

6. **Rozpoznání**

   Je vhodné nezatěžovat uživatelovu paměť a poskytnout mu viditelné objekty, akce a volby stejně jako instrukce provázející procesem v aplikaci.

7. **Přizpůsobení a efektivita**

   Systém by měl dokázat uspokojit oba typy uživatelů, nezkušeného i zkušeného, umožnit tak uživatelům přizpůsobit systém častým operacím.

8. **Minimalistický návrh**

   Systém by neměl zobrazovat informace, které jsou nepodstatné. Každá informace navíc konkuruje aktuálně důležité informaci a snižuje její viditelnost.

9. **Smysluplné chybové hlášky**

   Chybové hlášky by měly být vyjádřeny v běžném jazyce (bez chybových kódů), přesně popsat problém a navrhnout řešení.

10. **Nápověda a dokumentace**

    Systém by měl být použitelný bez dokumentace, nicméně i tak je nezbytné poskytnout nápovědu. Jakákoli informace by mela být dohledatelná. Nápověda by ale neměla být příliš obsáhlá.

Během implementace bude kladen důraz na dodržování výše sepsaných pravidel.

Po dokončení funkční aplikace je v budoucnu předpokládán rozvoj aplikace jehož předmětem bude změna uživatelského rozhraní založeném na grafickém návrhu profesionálního grafika. Také z tohoto důvodu bude současné rozhraní spíše prototypem, který bude vytvářen až během samotné implementace.

4.3 Architektura

Pravidla uživatelského rozhraní popisují aplikaci jako uživatelsky přívětivou, reagující na uživatelské akce, měnící se. Podstatnější je však podpora pro poslech hudby **bez přerušení** uvnitř prohlížeče, která definuje nestandardní požadavky na architekturu webové aplikace. Běžná webová interakce, kdy uživatelská akce vyvolá požadavek na server a prohlížeč reaguje překreslením celé stránky je nevhodná.

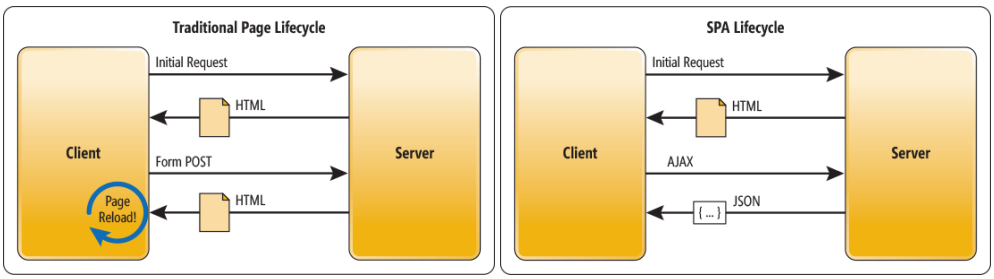
Webová aplikace, která nevyžaduje překreslení stránky po uživatelem vykonané akci využívá ke svému fungování skriptovací jazyk Javascript. Takové

aplikace se obecně nazývají klientskými (anglicky client-side) webovými aplikacemi. Tyto aplikace běží uvnitř webového prohlížeče a ke svému fungování z velké části server nepotřebují. Mezi front-end vývojáři se lze také setkat s pojmem HTML5 webové aplikace. Nazývají tak stránky, které jsou z velké části řízeny a vykreslovány Javascriptem a u kterých je využíváno nových technologií, které poskytuje právě HTML5. Mezi tyto technologie patří podpora pro multimediální obsah, canvas, web storage, history management a další.

4.3.1 Single-page application

Pro klientské webové aplikace se používá anglický termín Single-page application (zkratka SPA), který v češtině znamená „jednostránková aplikace“. Tento přístup tvorby webových aplikací stává v současnosti stále více oblíbeným.

Při úvodním požadavku webového prohlížeče je staženo kompletní uživatelské rozhraní v HTML, CSS a logika aplikace v Javascriptu. Další komunikace se serverem probíhá skrz AJAX požadavky přes definované rozhraní nejčastěji ve formátu JSON. Díky tomu není potřeba překreslovat celý dokument, ale pouze jeho části v závislosti na příchozích datech. Komunikaci znázorňuje obrázek 4.2 [14].



Obrázek 4.2: Komunikace se serverem tradiční webové stránky a Single-page aplikace

Řešení SPA oproti tradičnímu přístupu přináší několik **výhod**:

- **Uživatelský požitek** Aplikace reaguje na uživatelské akce plynule bez překreslení a s kratší odezvou. Uživatel získá dojmu práce s desktopovou aplikací. Interakce a ovládání je jednoduší. Chyby serveru jsou „zamaskovány“ - uživatel neuvidí stránku s chybovým kódem 500, ale aplikace mu poskytne při takovém požadavku poskytne informační hlášku.
- **Snazší vývoj a údržba** Aplikace má striktně oddělené uživatelské rozhraní a datový model aplikace. Server je odstíněn od znalosti a vytváření prezentační logiky ve formě HTML.

Samozřejmě existují také **nevýhody** SPA.

- **Úvodní načtení** Stažení celého uživatelského rozhraní při prvním dotazu může být pomalejší než u tradičního přístupu webových stránek.
- **Požadován Javascript** SPA vyžaduje povolený Javascript. Tato nevýhoda se zdá v současnosti nepodstatným přežitkem. Problém nastává, pokud by bylo nutné podporovat čtečky pro nevidomé.
- **Navigace a SEO** Navigace na stránce stejně jako překreslení stránky je řízeno pomocí Javascriptu. To může vytvářet problémy týkající se navigace a optimalizace pro vyhledávače.
- **Zabezpečení** Logika aplikace je dostupná v Javascriptu, jehož zdrojový kód je dostupný pro případné útočníky. To přináší další požadavky na autentizaci a autorizaci uživatelských akcí.

I přesto, že nevýhody převládají, práce vyžaduje nepřerušení přehrávání hudby uvnitř prohlížeče, které by nastalo přenačtením stránky po provedení uživatelské akce. Nevýhody SPA jsou tak brány jako ústupky, které nemají vliv na fungování aplikace. Případné další problémy budou diskutovány během implementace.

4.3.2 Komunikace se serverem

Navrhovaná architektura předpokládá dostupnost dat přes nezávislé rozhraní. Toto rozhraní, které bude poskytovat server, se obecně nazývá Web API. V kontextu webového vývoje je možné rozhraní nazývat pouze jako API. Komunikace bude probíhat skrz protokol HTTP v prostředí internetu. API bude sloužit především k dotazování na data a jejich trvalému ukládání.

4.4 Návrh API

Jak bylo v návrhu architektury nastíněno, aplikace bude pro získávání, práci a ukládání dat používat volání na webové aplikační rozhraní (API). Tato volání představují požadavky na server v konkrétním formátu a schématu. Návrh schématu, definice URL zdrojů, popis metod volání a výběr řešení konkrétních problému týkajících se návrhu API je součástí této kapitoly.

4.4.1 REST

Pro komunikaci se serverem bude použit koncept softwarové architektury REST. Jedná se o sadu způsobů a pravidel návrhu, při jejíž dodržování je docíleno rozšiřitelnosti takového rozhraní.

Při dodržení architektury REST je webová služba nebo aplikace označována jako RESTful.

Základem pro RESTful API je přístup k tzv. zdrojům za pomoci URL a standardních HTTP metod (GET, PUT, POST, DELETE atd.). Zdroj je objekt nebo entita nějakého typu, s vlastními daty, vztahy k ostatním zdrojům a množinou metod, pomocí níž je možné se zdrojem pracovat.

Speciálním zdrojem, označovaným jako kolekce, je skupina prvků stejného typu. Operace nad zdrojem kolekce se liší od operací nad je jedním prvkem.

4.4.1.1 Metody HTTP

Jak bylo v předchozích odstavcích řečeno, v architektuře REST probíhá komunikace za pomoci HTTP metod. Metoda je součástí požadavku odesílaného na server, vyjádřená je slovesem a jedná se o hlavní část „jednotného rozhraní“. Spolu se zdrojem ve formě podstatného jména poskytuje informaci o vykonávané akci.

Základní a také nejčastěji požívané HTTP metody jsou POST, GET, PUT a DELETE, které v tomto pořadí korespondují s operacemi create (vytvoření), read (čtení), update (editace), delete (smazání), známé pod zkratkou CRUD a používané u trvalých (perzistentních) uložišť. Dále existují další metody, které ale nejsou tak používané. Mezi ně patří například OPTIONS a HEAD.

4.4.2 Verze API

Již při návrhu API by měla být brána v úvahu možnost budoucí nekompatibilní změny či rozšíření API. Tato změna zapříčiní vznik nové verze. Z tohoto důvodu je nutné rozlišovat verzi API již od začátku i přesto, že žádná jiná verze neexistuje.

Způsobů, jak uvést verzi API ve volání, je několik. Prakticky se používají dvě metody a to uvedení verze v hlavičce dotazu nebo v přímo v URL. Třetí, méně používanou možností je využití hlavičky Accept. Její použití je zmíněno v článku Using the Accept Header to version your API [15].

Pro lepší představu je vhodné uvést příklady možností.

1. Součást URL dotazu

```
/api/v1/users/123
```

2. Vlastní hlavička dotazu

```
Api-Version: 1
```

3. Součást Accept hlavičky dotazu

```
Accept: application/vnd.example.v1+json
```

Proti uvedení verze API v URL hraje fakt, že verze API znamená pouze způsob, jakým získat data daného zdroje. Nejedná se o verzi tohoto zdroje a

sématicky to není správné. Je však možné se dívat na Naopak předostí tohoto řešení je jasná identifikace použité verze a jednoduchost řešení a implementace.

Druhou používanou možností je uvedení vlastní hlavičky v dotazu. Pojmenování hlavičky a její formát je jednou z konvencí určenou při návrhu a je nezbytné ji tak uvést v dokumentaci. Dále je vhodné zmínit, že předpona "X-" používaná k pojmenování vlastních či nestandardních hlaviček je v RFC 6648 [16] označena jako zastaralá, proto nic nebrání k použití názvu, který je uveden v příkladu.

Poslední výše uvedenou metodou verzování API je modifikace hlavičky Accept podle RFC 4288 - 3.2 Vendor Tree[17]. Tato varianta není tak často používaná jako předchozí a navíc je její použití náročnější pro testování.

Ve výsledku není důležité, která metoda bude zvolena, ale skutečnost, že některá bude použita a řádně zdokumentována. Příkladem může být verzování, které u svého API používá služba pro příjem plateb přes internet Stripe [18], jež označuje verzi dnem, kdy byla vydána. (například 2015-02-18). Na první pohled se zdá být verzování nejasné, avšak dokumentace změn [19] je více než dostatečná.

V aplikaci MusicQ bude použito verzování uvedené v URL a to vzhledem k jednoduchosti použití a převládajícímu využití u populárních webových služeb [20] využívajících REST architektury..

4.4.3 Konvence pojmenování a vzhled URL

4.4.3.1 Koncová lomítka

Jednou z otázek, které je nutné při návrhu REST API řešit, je vzhled samotných URL. Konkrétní problém představují koncová lomítka (angl. trailing slashes) v URL.

K problému koncových lomítek je vhodné si na začátku uvést příklady.

1. URL s koncovými lomítky:
   a) `/users/` *(kolekce)*
   b) `/users/123/`
2. URL bez koncových lomítek:
   a) `/users` *(kolekce)*
   b) `/users/123`

První řádek každého z příkladů je zdrojem pro kolekci. Druhý míří na zdroj jednoho objektu.

Pokud by se na URL nahlíželo podle zvyku ze systému Unix, tak jak je zmíněno v článku „Why trailing slashes are important“ [21], koncové lomítko v cestě znamená adresář (složku). Chybí-li v cestě koncové lomítko, jedná se

o adresu souboru. Adresář je v kolekce souborů. Pokud by se převzala tato konvence, URL zdroje kolekce by obsahovala koncové lomítko a URL zdroje jednoho objektu by byla bez lomítka na konci.

Článek také popisuje význam lomítka, pokud data zdroje obsahují relativní URL. V tomto případě opravdu záleží zda je lomítko přítomno nebo nikoli. Tomuto problému se lze snadno vyvarovat a aplikačně zajistit absolutní URL v atributech a datech zdroje.

Výše zmíněná pravidla by však definovala zbytečnou konvenci pro vzhled schématu API, proto je vhodné návrh zjednodušit. Reálně se tak nabízí tři možnosti jak řešit koncová lomítka v URL zdroje:

1. Všechny zdroje budou dostupné pouze skrz URL bez koncových lomítek

2. Všechny zdroje budou dostupné pouze skrz URL s koncovými lomítky

3. Všechny zdroje budou dostupné přes obě varianty

Už anglický překlad zkratky URL (Uniform Resource Locator) - „jednotná adresa zdroje“ - spolu s architekturou REST popisuje požadavek na jedinečnost adresy zdroje. Jinými slovy by stejný zdroj neměl být k dispozici skrze různá URL, což se u dvou adres lišících se pouze přítomností respektive nepřítomností koncového lomítka může stát. Proto není třeba nad třetí možností přemýšlet.

Nehledě na předchozí srovnání s významem koncového lomítka v Unixu a nehledě na problémy, které přináší relativní adresy v datech zdroje, jsou první dvě možnosti totožné. Proto nezáleží, která možnost bude zvolena. Důležité je připomenout si, že by stejně jako verzování API měl být návrh a následná dokumentace konzistentní.

Aplikace MusicQ bude pro adresy zdroje používat možnost bez koncového lomítka. Požadavek na zdroj lišící se pouze v přítomnosti koncového lomítka bude přesměrován stavovým kódem 301 [22] na podporovanou adresu.

4.4.3.2 Množné číslo

Další otázka, která vzniká během návrhu schématu API, se týká použití množného čísla v názvu zdroje oproti použití čísla jednotného.

Tento problém je ztotožnitelný se stejnou otázkou v oblasti relačních databází, který nemá jednoznačné řešení [23].

Při rozhodování v použití jednotného nebo množného čísla může hrát roli typ zdroje dostupný přes danou URL. Sémanticky by dávalo smysl, kdyby se zdroj kolekce nazýval množným číslem a pod zdrojem pojmenovaným jednotným číslem by se skrýval jeden prvek (objekt, entita).

V článku „All those sses: what pluralization method do you use for your REST APIs?“ [24] jsou popsány čtyři možnosti řešení:

1. **Smíšený** – Množné číslo je založeno na pohledu serveru

   `GET` `/members` *(čtení kolekce)*

   `POST` `/members` *(vytvoření **prvku**)*

   `PUT` `/members` *(editace kolekce)*

   `DELETE` `/members` *(odstranění kolekce)*

   `GET` `/member/1` *(čtení prvku)*

   `PUT` `/member/1` *(editace prvku)*

   `DELETE` `/member/1` *(odstranění prvku)*

2. **Smíšený** – Množné číslo je založeno na pohledu klienta

   Jediný rozdíl oproti předchozí možnosti je přidání prvku voláním POST operace na zdroj s názvem jednotného čísla. Tento přístup dává větší smysl klientovi (přidává jeden prvek).

   `GET` `/members` *(čtení kolekce)*

   `PUT` `/members` *(editace kolekce)*

   `DELETE` `/members` *(odstranění kolekce)*

   `GET` `/member/1` *(čtení prvku)*

   `POST` `/member` *(vytvoření prvku)*

   `PUT` `/member/1` *(editace prvku)*

   `DELETE` `/member/1` *(odstranění prvku)*

3. **Pouze jednotné číslo**

   Některé metody jako mazání a editace kolekce přestanou dávat smysl. Jejich použití však není tak časté. Otázkou je, zda by pro tyto operace neměla mít kolekce svůj vlastní identifikátor, aby s ní bylo zacházeno jako s prvkem.

   `GET` `/member` *(čtení kolekce)*

   `POST` `/member` *(vytvoření prvku)*

   `PUT` `/member` *(editace kolekce)*

   `DELETE` `/member` *(odstranění kolekce)*

   `GET` `/member/1` *(čtení prvku)*

   `PUT` `/member/1` *(editace prvku)*

   `DELETE` `/member/1` *(odstranění prvku)*

4. **Pouze množné číslo**

   Významově se jedná o lepší řešení než v předchozím případě. Všechny operace zní v angličtině dobře.

   `GET` `/members` *(čtení kolekce)*

   `POST` `/members` *(vytvoření prvku)*

   `PUT` `/members` *(editace kolekce)*

   `DELETE` `/members` *(odstranění kolekce)*

   `GET` `/members/1` *(čtení prvku)*

   `PUT` `/members/1` *(editace prvku)*

   `DELETE` `/members/1` *(odstranění prvku)*

Aplikace MusicQ si zakládá na jednoduchosti návrhu, proto ani jedna z prvních dvou možností řešení nebude brána v úvahu. Čtvrtá možnost vypadá jako lepší ze zbývajících, avšak angličtina v má některé tvary podstatných jmen nepravidelné (person - people, child - children), na první pohled nejasné (category - categories) nebo množné číslo u některých slov neexistuje. Proto a také vzhledem k tomu, že převažují dotazy na zdroje prvků oproti dotazům na kolekce, bude zvolena 3. možnost a to použití jednotného čísla.

4.4.4 Formát dat

V současnosti je při návrhu API rozumné přemýšlet pouze nad dvěma formáty pro výměnu dat. Jedním z nich je XML (eXtensible Markup Language) druhým JSON (Javascript Object Notation). Popisy formátů spolu s ukázky formátů se zabývá článek „API Data Exchange: XML vs. JSON“ [25] z kterého je čerpáno v následujících odstavcích.

U XML spolu s XSD (XML Schema Definition), které definuje strukturu XML dokumentu, může být zkontrolována konzistence a úplnost dat, během výměny mezi dvěma systémy, což snižuje nároky na kód, který data zpracovává.

Příklad reprezentace objektu v XML formátu znázorňuje kód 4.1

JSON je oproti XML jednodušší formát. Většina programovacích jazyků zvládá jeho zpracování nativně. Pokud je použit Javascript, deserializací JSON vzniknou přímo objekty, s kterými lze rovnou pracovat.

Jednoduchost JSON formátu umožnila stát se hlavním formátem pro výměnu dat v mnoha rychle se rozvíjejících programovacích jazycích v čele s Ruby. Také díky odlehčené struktuře JSON je oproti XML jednodušší formát parsovat. Kód 4.2 zobrazuje předchozí příklad, tak jak je objekt reprezentován v JSON.

Největší nevýhodou JSON formátu je jeho nedefinovaná struktura. Další nevýhodou je použití pouze pro jednoduché datové struktury.

```
<product>
     <id>15</id>
     <name>Widgets</name>
     <description>These widgets are the finest widgets ever made by
     ↪  anyone.</description>
     <options type="color">
          <item>Purple</item>
          <item>Green</item>
          <item>Orange</item>
     </options>
</product>
```

Zdrojový kód 4.1: Data ve formátu XML

```
{
"product" : {
     "id" : 15,
     "name" : "Widgets",
     "description" : "These widgets are the finest widgets ever made by
     ↪  anyone.",
     "options" : [
          {
               "type" : "color",
               "items" : [
                    "Purple",
                    "Green",
                    "Orange"
               ]
          }
       ]
   }
}
```

Zdrojový kód 4.2: Data ve formátu JSON

RESTful API by mělo být navržené jako rychlé, spolehlivé a jednoduché pro použití. Formát JSON je právě takovým formátem, který toto splňuje. Proto v aplikaci MusicQ bude použit.

Při samotném dotazu je vyžádání formátu docíleno hlavičkami Accept a Content-type. Accept říká, v jakém formátu je požadována odpověď. Může definovat jeden nebo více formátů. K označení formátu dat se používá textový popis „typ internetového média“ [26]. V tomto popisu může být použit znak hvězdičky „∗“ umožňující definovat zároveň více typů. Content-type naopak označuje formát dat posílaných při požadavku.

```
Accept: audio/*; q=0.2, audio/basic
```

```
Content-Type: text/html; charset=UTF-8
```

4.4.5 Autentizace

Autentizace (někdy také autentikace nebo autentifikace) je ověření, zda uživatel nebo aplikace může se zdrojem pracovat. Práce se zdrojem znamená jeho vytvoření, čtení, úpravu nebo smazání.

Vzhledem k návrhu aplikace využívající architekturu REST bylo nutné navrhnout bezpečný způsob jakým se klientské volání API bude ověřovat. Předpokládat budeme použití pro samotnou webovou stránku, tak pro budoucí mobilní aplikaci.

Způsobů a technik, jak autentizaci řešit, existuje několik. Mezi nejpoužívanější patří **HTTP Basic**, **HTTP Digest**, **OAuth 1.0** a **OAuth 2.0**. Jejich základní rozdělení je na ty, které využívají spojení zabezpečené (HTTPS, „HTTP over SSL“) nebo nezabezpečené (HTTP). Existuje také celá řada vlastních řešení, které však narážejí na problémy s bezpečností.

4.4.5.1 Basic access authentication

Je nejzákladnější způsob pro ověření uživatele na webu. Přenášeno je jméno a heslo ve formě „jmeno:heslo“ v kódování Base64. Každý požadavek na API provází hlavička Authorization v následujícím formátu:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Jméno a heslo klienta je zakódováno pomocí Base64, které díky tomu lze na serveru, zpětně rozkódovat a ověřit správnost údajů. Je zcela jasné, že tento způsob neposkytuje žádný způsob šifrování přenášeného uživatelského jména a hesla a jeho použití je z hlediska bezpečnosti podmíněno použitím HTTPS jako způsob přenosu dat.

4.4.5.2 Digest access authentication

Při použití Digest access authentication je server tou stranou, která musí učinit první krok. Přistupuje-li uživatel k důvěrnému zdroji, server reaguje odpovědí s hlavičkou `WWW-Authenticate` spolu s hodnotami, které musí poskytnout.

Server vygeneruje hodnotu nazvanou jako *nonce*, která by měla být unikátní pro každý požadavek. Druhou hodnotou je tzv. *opaque*. Jedná se o druhý unikátní řetzec vygenerovaný serverem u kterého je taktéž požadována odpověď v nezměněné podobě. Poslední hodnotou je obecně známý *realm*, což je řetězec s informací zobrazený uživateli.

Klient s pomocí funkce MD5 vytvoří hash v jasně definovaných krocích [27]:

1. Spočítej `A1` jako `MD5("username:realm:password")`
2. Spočítej `A2` jako `MD5("requestMethod:requestURI")`
3. Spočítej konečný hash jako `MD5("A1:nonce:A2")`.

Příklad odpovědi zaslané klientem obsahující původní hodnoty spolu s konečným hashem v hodnotě *response*:

```
Authorization: Digest username="Mufasa",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      response="e966c932a9242554e42c8ee200cec7f6",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Výhoda oproti *Basic access authentication* je, že *Digest access authentication* přenáší pouze hash místo samotného hesla. Vypočítaný hash se také mění spolu s každým dotazem díky unikátnímu *nonce*, který je také použit k výpočtu.

Nevýhodou se použití MD5 jako hashovací funkce, která je v současnosti považována za slabou. Další nevýhodou je možnost útoku zvaného Man in the middle, který vede k tomu, že klient není schopen rozeznat zda daný server je opravdu tím za co se vydává.

4.4.5.3 OAuth

OAuth byl navržen jako standard popsaný v RFC 5849 [28] především jako způsob jakým aplikace poskytne přístup jiné aplikaci bez vyžadování zadání uživatelského hesla.

OAuth 1.0

Jak popisuje článek „Top Differences between OAuth 1.0 and OAuth 2.0 for API Calls“ [29], OAuth 1.0 řeší problém poskytnutí přístupu chytrým způsobem za pomoci zabezpečené formy spojení tzv. handshake skrz API volání mezi dvěma aplikacemi. Toto umožňuje použití API pro způsoby, které dříve nebyly možné.

OAuth 1.0 funguje na základě skutečnosti, že klient i server sdílejí token (ve většině případů nazývaný) který je srovnatelný s uživatelským jménem a tzv. secret token (tajný) jež nahrazuje uživatelské heslo. Klient musí vytvořit „podpis“ pro každé API volání zašifrováním několika unikátních informací s použitím zmíněného tajného tokenu (klíče). Server musí vygenerovat stejný podpis a povolit přístup pouze pokud podpisy sedí.

Výhodou tohoto postupu je skutečnost, že není možné zjistit tajný token, protože vždy, když putuje přes síť, je šifrován. Pouze klient a server mají klíč.

Celý proces OAuth 1.0 je možný bez použití SSL, které může zpomalit komunikaci.

OAuth 2.0

OAuth 2.0 je evolucí předchozího. Přináší především zjednodušení celého protokolu. Mezi hlavní rozdíly patří body týkající se podpisu požadavku:

1. Zabezpečené připojení je vyžadováno při požadavku na generování tokenu. Jedná se o obrovské snížení složitosti celého protokolu, protože podpis požadavku není potřeba.

2. Stejně tak není třeba podpis při samotných API voláních. Jeho roli zde zastává vygenerovaný token. I přesto je zde doporučováno SSL.

3. Jakmile je token ověřen, OAuth 1.0 požaduje po klientovi posílat spolu s API voláním dva bezpečnostní tokeny, z nichž je generován podpis. OAuth 2.0 má jeden ověřovací token a žádný podpis není potřeba.

4. Je zřejmé, jaká část protokolu je implementována „vlastníkem zdroje“, tedy serverem, který implementuje API, a která část je implementována na odděleném „autorizačním serveru“. Toto umožňuje využití třetích stran pro autentizaci.

Jak je vidět z rozdílů, OAuth 2.0 nepodporuje podpis, šifrování nebo ověření klienta. V těchto věcech tak, aby byla zajištěna nepopiratelnost a ověření serveru, kompletně závisí na SSL.

4.4.5.4 Vlastní řešení

Návrh vlastního řešení zabezpečení API, které používá nezabezpečené spojení popisuje článek „Designing a Secure REST (Web) API without OAuth“ [30]. Jedná se o praktický stejný způsob zabezpečení, které používá u svých služeb firma Amazon [3]. Jedná se o bezpečné dotazování na API odolné proti známým typům útoků i bez použití HTTPS. Je založené na vytváření podpisu pomocí soukromého klíče, který zná pouze klient a server podobně, jak je tomu v případě OAuth.

4.4.5.5 JSON Web Tokens

Při použití Javascriptu jako programovacího jazyku na straně klienta je nemožné použití metod, které vyžadují uložení skrytého tokenu (soukromého klíče) na straně klienta. Tento údaj stejně jako algoritmus bude vždy dostupný pro případného útočníka.

Nabízí se tedy otázka jakým způsobem řešit tento problém? Východiskem je formát tokenu zvaný zkratkou JWT (JSON Web Tokens). Jedná se o relativně nový autentizační mechanizmus jehož specifikace je v současné době v návrhu [31].

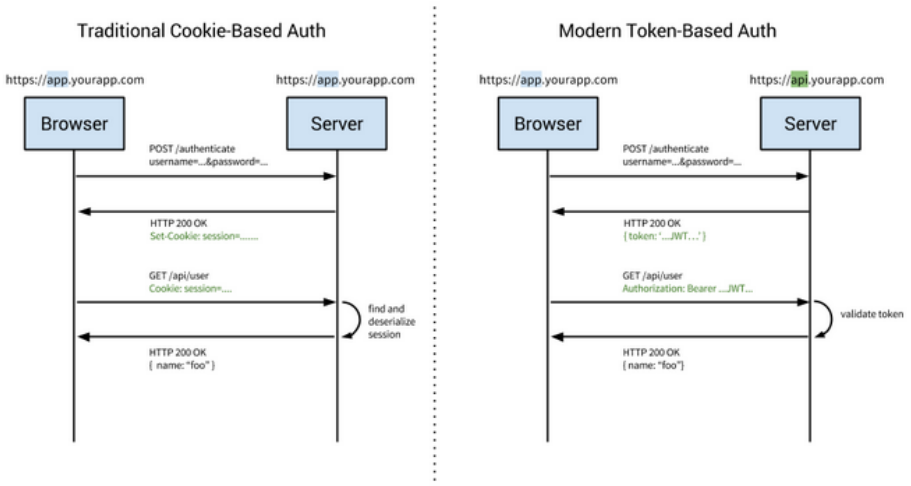
Tento token umožňuje navrhnout bezpečnou komunikaci mezi dvěma systémy. Je možné jej rozdělit na tři části – *hlavičku, přenášená data a podpis*:

1. Hlavička (angl. header) je částí, která obsahuje typ tokenu a způsob šifrování.

[3] `http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html`

2. Přenášená data (angl. payload) obsahují jakékoli další informace. Do této části je možné uložit například id uživatele, platnost tokenu a další. Celá část je převedena do Base64 kódování.

3. Podpis (angl. signature) se skládá z kombinace hlavičky, přenášených dat a tajného klíče. Tento klíč musí být uložen bezpečně na straně serveru.



Obrázek 4.3: Ukázka přístupu autentizace založeného na cookies a moderního přístupu s využitím JWT

Tento přístup autentizace založený na použití tokenu se hodně podobá tradičnímu přístupu založeného na ukládání cookies (porovnání komunikace je zobrazeno na obrázku 4.3), ale zároveň, jak je zmíněno v článku „Cookies vs Tokens. Getting auth right with Angular.JS“ [32], přináší mnoho výhod:

1. Cookies spolu s CORS (Cross-origin resource sharing, ve volném překladu *sdílené zdroje odjinud*) nefunguje dobře napříč rozdílnými doménami. Použití tokenu dovoluje vytvářet AJAXové volání na jakýkoli server s jakoukoli doménou, protože je použita hlavička pro přenos uživatelských dat.

2. Není nutné udržovat na serveru uložiště pro dané sezení a server tak funguje bezstavově (angl. stateless). Všechen stav obsahuje samotný token. Ostatní stav je uložen na straně klienta.

3. Na server lze nahlížet jako službu poskytující pouze API. Server se nemusí poskytovat generování tokenu.

4. Pro použití mobilními zařízeními nejsou cookies ideální. Při dotazování na zabezpečené API je nutné vytvářet „přepravky“ cookies. JWT toto značně zjednodušuje.

5. Není-li autentizace závislá na cookies, není třeba chránit aplikaci před útokem CSRF (Cross-site Request Forgery).

6. V neposlední řadě má přístup JWT pozitivní vliv na výkon serveru. Není potřeba hledat sezení uživatele v perzistentním uložišti. Token je ověřován v paměti.

I přesto, že není možné modifikovat token odchycený při přenosu po síti (podpis toto znemožňuje), samotná data obsahují informace, které není vhodné veřejně vystavit (kódování Base64 nešifruje data). Proto a také z důvodu úvodního odeslání uživatelského jména a hesla pro vygenerování tokenu, je nutné celý proces chránit zabezpečeným přenosem HTTPS.

4.5 Technologie

Následující odstavce se věnují technologiím, knihovnám a frameworkům, které budou použity pro implementaci aplikace.

4.5.1 React

React je Javascriptová knihovna pro tvorbu uživatelského rozhraní za jejíž vývojem stojí Facebook. V softwarové architektuře MVC (Model–view–controller) zastává React funkci *view*. Jedná se o knihovnu, která umožňuje vytvářet uživatelské rozhraní a reagovat na změny dat jeho překreslením.

React přistupuje k vytváření uživatelského rozhraní jeho dělením na tzv. **komponenty**. Každá tato komponenta má svůj stav a její vzhled je definován deklarativně v závislosti na aktuálním stavu. Změna stavu znamená překreslení komponenty. Aby toto překreslení nezpůsobovalo ztrátu výkonu, React využívá vlastní „odlehčenou“ implementaci DOMu – Virtuální DOM.

4.5.1.1 Virtuální DOM

DOM (Document Object Model) je objektově orientovaná reprezentace HTML dokumentu. Jedná se o jazykově nezávislé rozhraní umožňující přístup ke struktuře dokumentu nebo jeho částí a její následnou modifikaci. K této manipulaci se strukturou dokumentu v aplikacích je využíván především Javascript.

Virtuální DOM je abstrakcí nad DOMem umožňující jednodušší přístup k programování a lepší výkon. V tradičním způsobu manipulace s DOMem je nutné sledovat změny dat a imperativním způsobem provádět změny v DOMu tak, aby odpovídal aktuálnímu stavu. Jak již bylo zmíněno v přechozích odstavcích React s umožňuje definovat vzhled dokumentu respektive komponent deklarativním způsobem. Programátor tak neřeší, co se má stát, pokud nastane změna, ale pouze jak má daná komponenta s aktálním stavem dat vypadat. Pokud je tedy vyžadováno překreslení komponenty, React vytvoří reprezentaci nové podoby a porovná ji s tou předchozí. Následně vytvoří minimální

množinu změn, která je aplikována do reálného DOMu. React manipuluje s virtuálním DOMem v paměti, což je mnohem efektivnější než manipulace s reálným DOMem v prohlížeči.

4.5.1.2 Stav komponent a tok dat

Jak již bylo v úvodu zmíněno, React je založen na vytváření jednotlivých znovupoužitelných komponent, které společně reprezentují *view* aplikace.

Každá komponenta má svůj vlastní *stav* (angl. state), který si drží u sebe. Tento stav ovlivňuje vzhled dané komponenty. Při změně stavu bývá komponenta překreslena. Druhým vstupním parametrem komponenty jsou *vlastnosti* (angl. properties), které také mohou ovlivňovat výstup komponenty.

HTML kód komponenty vzniká ve funkci *render*. Jedná se o jedinou povinnou funkci React komponenty. Funkce *render* deklarativně popisuje vzhled komponenty v závislosti na jejím aktuálním *stavu* a *vlastnostech*. Výstup této metody není řetězec HTML, ale strom komponent. Tento výstup interně zpracovává React za pomoci technologie virtuálního DOMu popsaného v předchozích odstavcích.

Z komponent lze vytvářet jiné komponenty pomocí skládání. Příklad takové kompozice je ukázán ve zdrojovém kódu 4.3. Komponenta `CommentBox` se skládá z `CommentList` a `CommentForm`.

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList />
        <CommentForm />
      </div>
    );
  }
});
```

Zdrojový kód 4.3: Skládání React komponent

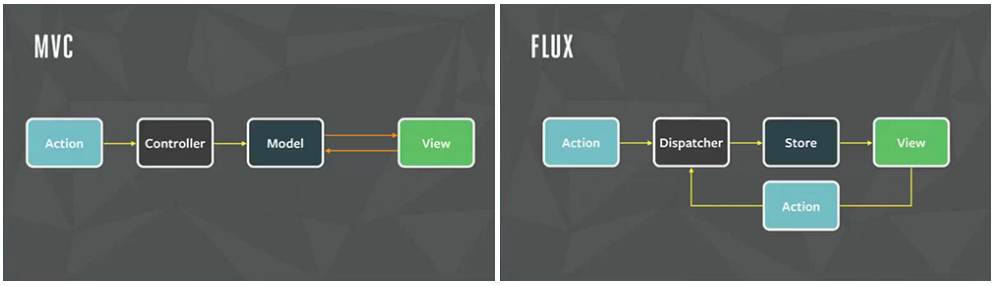
Rozhraní aplikace v Reactu je tak složeno ze stromu komponent. Při rozhodování o tom, co má být komponentou platí stejné obecně známé principy z objektově orientovaného programování. Jedním z nich je princip jedné odpovědnosti, což znamená, že komponenta by měla ideálně dělat pouze jednu věc. Pokud tomu tak není, měla by být rozložena na menší komponenty. Druhým pricipem je princip známý pod zkratkou DRY („Don't Repeat Yourself“) v češtině znamenající „Neopakuj se“, řešící jak je možné minimálně reprezentovat stav aplikace, který je potřebný k vykreslení všech komponent.

Dobře navržené komponenty mohou být například pouze se změnou konfigurace (předané jiné *vlastnosti*) využity na více místech aplikace. React tedy podporuje znovupoužitelnost komponent.

Z rodičovské komponenty je možné předávat data potomkům skrz atributy jako je zvykem v HTML. Hodnoty těchto atributů jsou v potomkovi k dispozici jako *vlastnosti* v proměnné `this.props`. Jedná se o odlišný způsob oproti konkurenčním knihovnám založených na obousměrném předávání dat tzv. *two-way data binding (two-way data flow)*. Způsob předávání dat mezi rodičem a jeho potomky používaný Reactem se nazývá *one-way data binding (one-way data flow)* – volně přeloženo jako *jednosměrné svázání dat (jednosměrný datový tok)*. Důvodem je, že tok dat v jednom směru je mnohem jasnější a je možné jej při vývoji aplikace sledovat od začátku do konce. Jednotlivé závislosti mezi komponentami lze určit mnohem snáz a předchází se tak neurčitým stavům a chybám uživatelského rozhraní rozsáhlé aplikace.

4.5.2 Flux

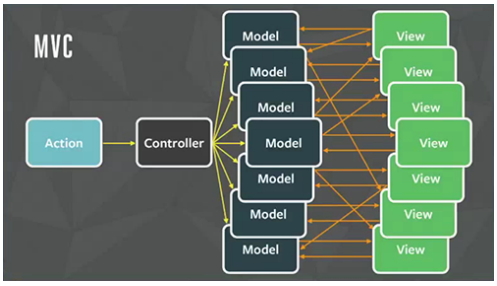
Flux je softwarová architektura, která je používána firmou Facebook pro tvorbu klientských webových aplikací psaných v Javascriptu. Flux má blízko a dá se místy přirovnat k jinému obecně známému návrhovému vzoru – MVC (obrázek 4.4 [33]).



Obrázek 4.4: Porovnání návrhových vzorů MVC a Flux

Rozdílem mezi vzory je přístup Fluxu k datovému toku. V reálné MVC aplikaci, kde existuje více *modelů* a *view* a jejichž datové závislosti budou provázané (obrázek 4.5), není snadné určit, zda nevznikne cyklus. Tedy situace, kdy změna dat některého z *modelů* má za následek změnu *view*, jež změní jiný *model* opakovaně tak, že vznikne nekonečný cyklus. Tento cyklus vede k snížení výkonu aplikace a je těžké ho najít, natož odstranit. U rozsáhlé MVC aplikace je toto opravdu těžké eliminovat. Flux je proto založen na jednosměrném datovém toku, tak jak znázorňují šipky na obrázku 4.4. Tento přístup dělá chování aplikace více předvídatelným.

Jak je vidět na obrázku 4.4, architekturu Fluxu tvoří několik částí. Dispatcher, store (uložiště), view (pohled) a action (akce).



Obrázek 4.5: Vzor MVC u reálné aplikace

4.5.2.1 Dispatcher

Překlad do češtiny by mohl být „odesílatel“ nebo jednoduše „dispečer“, avšak „Dispatcher“, stejně jako ostatní pojmy Fluxu není vhodné překládat z důvodu, že se jedná o relativně nové pojmy a nevhodným překladem by mohl být jejich význam posunut nebo zcela změněn.

Dispatcher je na rozdíl od ostatních částí v aplikaci pouze jeden a existuje jako tzv. singleton. Jedná se o jakýsi manažer, který přijímá akce, které následně rozesílá spolu s daty skrz registrované „zpětné volání“ (angl. callbacks) do stores. To znamená, že veškeré akce, které vznikají v aplikaci, musí procházet skrz Dispatcher. Chování aplikace tak lze snadno kontrolovat právě v této komponentě architektury Flux.

4.5.2.2 Stores

Stores (česky uložiště) obsahují stav a logiku aplikace. Pro každou doménu aplikace existuje jeden store. Každý store poskytuje zpětné volání, které registruje u dispatcheru. Toto zpětné volání přijímá akci společně s daty a v závislosti na typu akce zpracuje data – například uloží do interní proměnné. Nic mimo store neví, jak jsou data uložena. Store také neposkytuje žádné setter metody známé z modelových tříd MVC. Jediným způsobem, jak data do storu uložit, je skrz výše zmíněné zpětné volání. Po uložení dat následuje vyslání události informující o této skutečnosti.

Stores obsahují metody k získání jejich aktuálního stavu (obdoba getter metod). Dále poskytují metody pro registraci respektive zrušení registrace událostí, které store dokáže vysílat. Těchto metod využívají views.

4.5.2.3 Views a Controller-Views

Views reprezentuje uživatelské rozhraní aplikace. Flux v této součásti architektury předpokládá použití Reactu. View se skládá z komponent Reactu, které se pomocí metod některého Store registrují na události změn dat. Data získá komponenta pomocí tzv. Store Queries, veřejně dostupných metod jednotlivých Stores k tomu určených. Po přijetí dat komponenta uloží data do

svého *stavu*, případně pomocí atributů (*vlastností*) předá data komponentám níže v hierarchii, a sama je překreslena. Takové speciální komponenty – reagující na změnu dat *store* a předávající data příslušným potomkům – je správné nazývat *controller-views*.

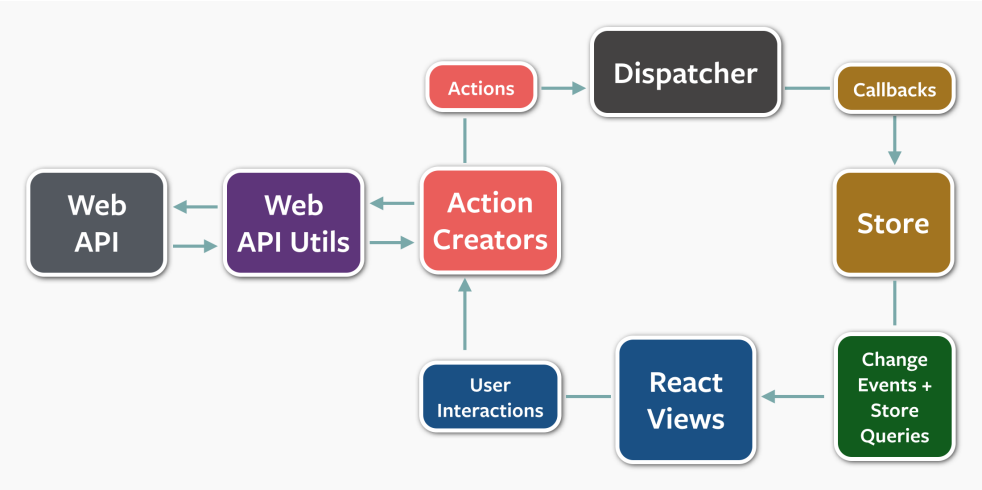
Vykoná-li uživatel akci na kterou má systém nějakým způsobem zareagovat, React komponenta za pomoci metod tzv. Action Creators vytvoří informaci o této skutečnosti.

4.5.2.4 Akce a Action Creators

Jak bylo v předchozím odstavci popsáno, v architektuře Flux existují pomocné metody dělené do tříd podle domény nazývané Action Creators. Ty jsou určeny pro vytváření konkrétních *akcí*. Každá tato akce má unikátní typ, který říká, co se má dále provést. Tato akce je spolu s novými daty vyslána pomocí dispatcheru do storů. Tím se celý okruh toku dat v architektuře návrhového vzoru Flux uzavírá.

Akce však nemusí přicházet pouze z aplikace (uživatelské akce), ale také z jiných míst, například ze serveru s využitím API. To se děje především během inicializace.

Celý diagram všech součástí architektury Flux znázorňuje obrázek 4.6.



Obrázek 4.6: Diagram součástí Flux aplikace

4.5.3 Silex

Silex je jednoduchý PHP framework, který je postavený na součástech mnohem robustnějšího frameworku Symfony2.

Silex je na oficiálních stránkách [4] popsán jako mikroframework s jádrem pro vytváření jednoduchých jednosouborových aplikací zaměřený na:

- **Stručnost** Silex nabízí intuitivní a stručné API, které je radost používat.
- **Rozšířitelnost** Silex má rozšířitelný systém založený na kontejneru pro služby nazvaném *Pimple*, který umožňuje snadným způsobem propojit framework s knihovnami třetích stran.
- **Testovatelnost** Silex používá `HttpKernel` ze Symfony2, který abstrahuje dotaz a odpověď. To vytváří velmi snadno testovatelné aplikace stejně jako framework samotný. Tato komponenta zároveň respektuje HTTP specifikaci a vybízí k jeho správnému použití.

Jak již bylo zmíněno, Silex patří do kategorie tzv. mikroframeworků, která je zaměřená na snadný způsob zpracování dotazu, který je zároveň rychlejší než při použití klasického frameworku. Příklad jednoduchého použití je uveden ve zdrojovém kódu 4.4. Díky svojí snadno čitelné struktuře spolu v kombinaci s anonymními funkcemi PHP dokonce umožňuje umístit veškerou logiku do jednoho souboru.

```php
require_once __DIR__.'/../vendor/autoload.php';

$app = new Silex\Application();

$app->get('/hello/{name}', function($name) use($app) {
    return 'Hello '.$app->escape($name);
});

$app->run();
```

Zdrojový kód 4.4: Zpracování dotazu frameworkem Silex

Silex je založen na jednoduché definici routování, poskytuje metody `get`, `post`, `put`, `delete` reflektující standardní HTTP metody a spolu s rychlostí zpracování a zabudovanou podporou odpovědi v JSON formátu je skvělou volbou pro zpracování REST požadavků na straně serveru.

4.5.4 Doctrine DBAL

Pro práci s databází na straně serveru bude použita databázová abstraktní vrstva Doctrine DBAL. Jedná se o jednoduchou nadstavbu okolo PDO (PHP Data Objects) umožňující objektový přístup k databázi – výběr a manipulaci s uloženými daty.

[4] `http://silex.sensiolabs.org/`

Součástí Doctrine DBAL je třída `QueryBuilder`, která nabízí vytváření dotazů SQL postupným skládáním.

Další výhodou je získání dat jako pole, které lze pro účely aplikace snadno konverovat do formátu JSON a použít v odpovědi na API dotaz. To vše v požadované rychlosti.

4.5.5 Bootstrap

Bootstrap je sada nástrojů HTML a CSS pro vytváření webových stránek a webových aplikací. Obsahuje připravené styly pro všechny standardní i nestandardní prvky, jmenovitě pro nadpisy, tabulky, formuláře, tlačítka, odkazy, obrázky nebo pro vyskakovací menu, záložky a navigační panely.

Nespornou výhodou celého Bootstrapu je připravenost pro mobilní zařízení v čemž spočívá podpora responzivního designu a možnost definice přizpůsobitelné HTML struktury.

Bootstrap bude využit pro implementaci klientské části aplikace, tedy pro její uživatelské rozhraní. Základní vzhled bude založen na šabloně Cover [5].

[5] `http://getbootstrap.com/examples/cover/`

Kapitola 5

Implementace

Implementace probíhala v cyklech srovnatelných s inkrementálním vývojem, kdy každý cyklus se skládal z detailního návrhu, implementace a testování funkčnosti.

Tento přístup umožňoval navázat na prvotní koncept, analýzu požadavků a návrh architektury. Zároveň bylo možné aplikaci spustit po jednom nebo více cyklech a otestovat funkčnost, kterou do té doby umožňovala.

Tato kapitola popisuje detaily implementace a zaměřuje se nad zajímavými částmi aplikace.

5.1 Výchozí implementace

Základním cílem výchozí implementace bylo vytvořit prototyp, který spustí přehrávání uvnitř prohlížeče. Jak bylo rozhodnuto v kapitole 3, jako zdroj hudby byla použita služba **YouTube**.

Dále bylo nutné implementovat získání informací o přehrávané písničce pro zobrazení jejího názvu na stránce během přehrávání.

5.1.1 Přehrávání

5.1.1.1 Popis přehrávače

YouTube poskytuje tři verze přehrávačů použitelné uvnitř webového prohlížeče, které je možné ovládat pomocí API. Dva z nich – ActionScript 3.0 (Flash) Player a Javascript Player – byly 27. ledna 2015 označeny jako zastaralé [34]. Poslední možnost na rozdíl od rozhraní API přehrávačů Flash a Javascript, které oba vkládají do stránky objekt Flash, rozhraní API přehrávače v prvku iframe odesílá obsah do tagu `<iframe>` stránky. Tento přístup je flexibilnější než předchozí rozhraní API, protože umožňuje službě YouTube předávat mobilním zařízením, která nepodporují Flash, přehrávač HTML5 namísto přehrávače Flash [35].

Pomocí funkcí Javascript rozhraní API je možné zařazovat videa k přehrání do fronty, přehrát, pozastavit, nebo zastavit přehrávání videí, upravit hlasitost přehrávače a získávat informace o přehrávaném videu.

V budoucím rozšíření aplikace MusicQ je plánována podpora přehrávání z více zdrojů. Při výchozí implementaci byl brán na tuto skutečnost zřetel a nebyla využita možnost použití fronty přehrávání, které je k dispozici přímo u samotného YouTube přehrávače.

5.1.1.2 Použití přehrávače

Načtení požadovaného videa do stránky probíhá v několika krocích.

1. Do stránky je načten kód rozhraní API přehrávače, tak jak znázorňuje kód 5.1. V příkladu je ke stažení kódu API použita modifikace DOM, což zajišťuje asynchronní vyvolání kódu. (Atribut async tagu <script>, který také umožňuje asynchronní stahování, není dosud podporován ve všech moderních prohlížečích) [6].

2. Po úplném stažení Javascript kódu rozhraní API je zavolána globální funkce `onYouTubeIframeAPIReady`, kde je vytvořen objekt `YT.Player`, který do stránky vloží videopřehrávač.

3. Při vytváření objektu `YT.Player` lze mimo jiných parametrů (velikost, ID videa) v poli `events` přidat posluchače událostí (funkce), které se budou spouštět na základě konkrétních událostí, jako je například změna stavu přehrávače nebo změna kvality přehrávání. Toho lze docílit také později pomocí funkce `addEventListener`.

```
var tag = document.createElement('script');

tag.src = "https://www.youtube.com/iframe_api";
var firstScriptTag = document.getElementsByTagName('script')[0];
firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);
```

Zdrojový kód 5.1: Asynchronní načtení kódu API přehrávače YouTube

Takto načtený přehrávač je Javascriptem ovládán pomocí dostupných method instance výše zmíněného objektu `YT.Player`.

Mezi jednu z hlavních funkcí použitou v aplikaci patří `loadVideoById`, jejíž syntaxe je následující:

```
player.loadVideoById(videoId:String, startSeconds:Number,
↪  suggestedQuality:String):Void
```

[6] http://stackoverflow.com/questions/1834077/which-browsers-support-script-async-async/1834129#1834129

- Povinný parametr `videoId` specifikuje ID videa YouTube, které se má přehrát. Tento parametr je konkrétně popsán v následujícím oddíle textu.
- Nepovinný parametr `startSeconds` přijímá celé číslo nebo číslo s plovoucí desetinnou čárkou. Pokud je zadaný, přehrávání videa bude zahájeno klíčovým snímkem, který je zadanému času nejbližší.
- Nepovinný parametr `endSeconds` přijímá celé číslo nebo číslo s plovoucí desetinnou čárkou. Pokud je zadaný, přehrávání videa skončí v zadaný čas. Tento parametr není v aplikaci používán. Jeho využití je diskutováno v kapitole 6
- Nepovinný parametr `suggestedQuality` určuje navrhovanou kvalitu přehrávání videa. Vzhledem k možnému použití aplikace na mobilních připojeních. Je jeho hodnota ponechána na výchozím `default`, což dá pokyn službě YouTube, aby vybrala nejvhodnější kvalitu přehrávání, která se liší pro různé uživatele, videa, systémy a další podmínky přehrávání.

Metoda pro spuštění přehrávání vyžaduje jeden povinný parametr – `videoId`. Jeho získání popisují následující odstavce.

5.1.2 Přidání hudby

Aby bylo možné začít samotné přehrávání hudby, je potřeba poskytnout uživateli možnost přidání požadované hudby respektive videa při použití YouTube jako zdroje.

Ve výchozí implementaci bylo pro zjednodušení vyžadováno vložení URL daného videa.

5.1.2.1 Ověření URL

Pro ověření správnosti vstupního URL zadaného uživatelem byla vytvořena třída PHP se statickými metodami. K ověření, zda URL vyhovuje formátu adresy URL YouTube videa, bylo využito regulárního výrazu. Implementaci těchto statických method znázorňuje zdrojový kód 5.2.

5.1.2.2 Získání informací o videu

Po ověření platného formátu URL odkazu na video aplikace uloží tuto skutečnost do databáze.

Mimo ID videa zjistí další potřebné informace potřebné pro správné fungování aplikace. K tomuto účelu využívá aplikace MusicQ rozhraní „YouTube Data API“, které poskytuje veškeré možné volání pro práci se zdroji videí. Jako formát dat se používá JSON.

Spolu s každým požadavkem musí být zasílán klíč API. Tento klíč je vygenerován pomocí Google Developers Console.

```
class YoutubeUtils
{
    // ...
    public static function isYoutubeUrl( $url )
    {
        return (bool) preg_match(
        ↪  '#(https?://)?(www\.)?(youtube\.com|youtu\.be)#', $url );
    }

    public static function getVideoIdFormUrl( $url )
    {
        preg_match(
        ↪  '#(?<=embed/|v=|v/|vi=|vi/|youtu\.be/)[a-zA-Z0-9_-]{11}#',
        ↪  $url, $matches );
        return $matches[0];
    }

    // ...
}
```

Zdrojový kód 5.2: Metody pro parsování ID videa z YouTube odkazu

MusicQ využívá pro dotazy na API s použitím oficiální Google knihovny v jazyce PHP. Jedná se o zjednodušení volání YouTube API.

Aplikace MusicQ pomocí volání metody `listVideos` nad instancí třídy `Google_Service_YouTube_Videos_Resource` s předáním parametru `id` získá informace o videu. Během volání je použito tzv. částečného zdroje, kde API umožňuje a zároveň také požaduje uvedení částí zdroje, které má vrátit. Tím je zabráněno přenosu, parsování a ukládání nepotřebných dat. Tento přístup také zajišťuje efektivního využití síťového provozu. Požadovaný název obsahuje část zdroje s klíčem `snippet` a délka trvání je k dispozici v části `contentDetails`. Použití knihovny a uložení těchto dat je uveden ve zdrojovém kódu 5.3.

5.1.3 Spuštění přehrávání

Během načtení stránky přehrávače byl ve výchozí implementaci pomocí funkce `setInterval` zaregistrován callback AJAXového volání na server, který se v formou tzv. long pollingu v pravidelných intervalech dotazuje na zdroj `/api/v1/queue/{queueId}/song/current`, kde `{queueId}` značí ID aktuální fronty.

Po uložení písničky do databáze, jak popisuje předchozí část 5.1.2.2, začne odpověď na dotaz vracet JSON s potřebnými informacemi o aktuální písničce. Tyto informace je možné použít pro spuštění přehrávání Javascriptem voláním metody `loadVideoById` objektu `YT.Player`.

```
$items          = $app['youtube']->videos
    ->listVideos( 'id,snippet,contentDetails', [
        'id' => $id
    ] )
    ->getItems();
$item           = reset( $items );
$snippet        = $item->getSnippet();
$contentDetails = $item->getContentDetails();

$song             = [ ];
$song['type']     = 'youtube';
$song['title']    = $snippet['title'];
$song['duration'] = YoutubeUtils::parseDuration(
↪   $contentDetails['duration'] );

$songType            = [ ];
$songType['videoId'] = $id;

$id = $app['repository']->saveYoutubeSong( $song, $songType );
```

Zdrojový kód 5.3: API dotaz na informace o videu YouTube a jeho zpracování

5.2 Uživatelé systému a autentizace

Systém ve své první verzi počítá s dvěma rolemi pro uživatele:

1. Nepřihlášený uživatel
2. Přihlášený uživatel

I přesto, že se nejedná o žádnou pokročilou strukturu, i v této fázi byl kladen důraz na budoucí rozšíření těchto rolí o uživatele s většími právy. Například pro správce systému nebo vlastníka a administrátora jednotlivých playlistů (front přehrávání).

Nepřihlášenému uživateli je dovoleno přistoupit pouze na úvodní obrazovku a samozřejmě na obrazovku pro přihlášení. Pokusí-li se zobrazit stránku, která vyžaduje přihlášeného uživatele je „přesměrován“ na přihlášení a zobrazena hláška o nutnosti zadání svých údajů. Po úspěšném přihlášení je mu zobrazena původně požadovaná stránka.

Celý výše zmíněný proces má na starost definice pravidel v samotném `react-router` (zdrojový kód 5.4), který registruje controller-view nazvaný `AuthenticatedHandler`. Ten má jednu statickou metodu `willTransitionTo`, která je volána před vykreslením dané komponenty (a jejích potomků) a dává možnost přesměrovat nebo přerušit přechod na komponentu (její vykreslení).

Autentizace na straně serveru (API) je řešena tzv. middleware. Framework Silex umožňuje zaregistrovat metodu k požadavku na daný zdroj, která je volána **před** vykonáním samotného controlleru. Tato metoda je zaregistrována pomocí metody `before` controlleru, která očekává funkci jako parametr. Tuto

```
var routes = (
    <Route name="home" handler={App} path="/">
        <DefaultRoute handler={HomeView} />
        <Route name="login" handler={LoginView} />
        <Route name="logout" handler={Logout} />
        <Route name="auth" handler={AuthenticatedHandler}>
            <Route handler={PlayerView} path="/queue/:queueId"
             ↪  ignoreScrollBehavior>
                <Route name="add-song">
                    <DefaultRoute handler={AddSongView} />
                    <Route name="play-song" path=":playSongId"
                     ↪  handler={PlaySongView} />
                </Route>
                <DefaultRoute name="queue" handler={QueueView} />
                <Route name="video" handler={VideoView} />
            </Route>
        </Route>
        <NotFoundRoute handler={NotFoundHandler}/>
    </Route>
);
```

Zdrojový kód 5.4: Deklarativní definice routování v JSX

funkci poskytuje servisní třída `JWTAuthenticator`. Funkce kontroluje požadavek na přítomnost hlavičky `Authorization` spolu s platným JWT tokenem. Selže-li ověření v této fázi, API vrací odpověď ve formátu JSON s popisem chyby a stavovým kódem 401 Unauthorized (zdrojový kód 5.5)

5.3 Databáze a datový model

V rámci inkrementálního vývoje byl využit nástroj Doctrine Migrations, který nabízí tzv. databázové migrace. Zjednodušeně by se dal popsat proces migrací jako verzování databáze, kdy není potřeba uchovávat známé create nebo insert SQL scripty z tradičního přístupu k této problematice. Definice tabulek a jejich sloupců, stejně jako změny mezi verzemi jsou popsány v jazyce PHP s využitím objektově orientovaného programování. Spuštění migrací a provedení změn v databázi se provádí přes příkazovou řádku.

Tento přístup tak umožňoval rozšiřování a změnu schématu postupným přidáváním definic během vývoje.

Samotný návrh probíhal během implementace s využitím znalostí autora práce v této oblasti.

Výsledné schéma aktuální verze aplikace je zobrazeno na obrázku 5.1.

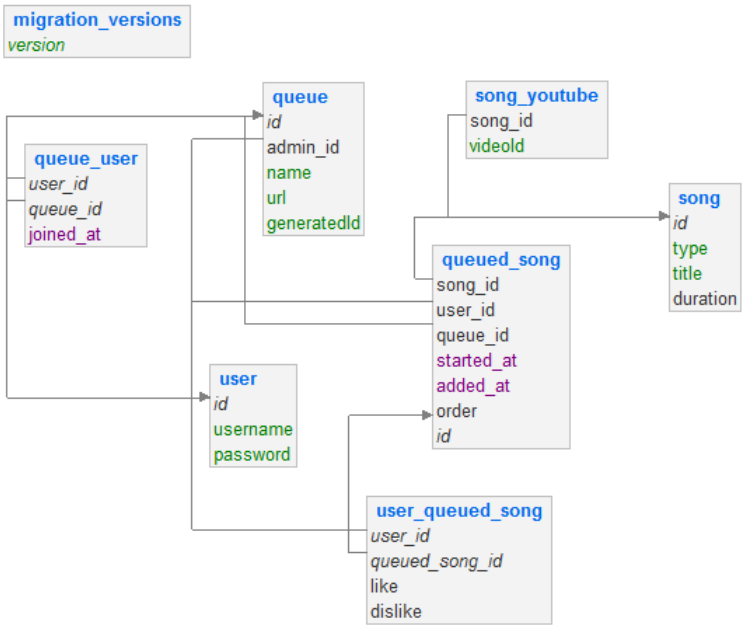
```php
class JWTAuthenticator
{

    /* ... */

    public function authenticate( Request $request )
    {

        $authentication = $request->headers->get( 'Authorization' );
        if ( ! $authentication) {

            $data                = [ ];
            $data['status']      = 401;
            $data['code']        = ErrorCodes::UNAUTHENTICATED;
            $data['message']     = 'Unauthenticated';
            $data['description'] = 'You need to login before this action';

            return new JsonResponse( $data, 401 );
        }
        $authentication = explode( ' ', $authentication );

        $token = @$authentication[1];
        try {
            JWT::decode( $token, $this->secretKey, [ 'HS256' ] );
        } catch ( \Exception $e ) {
            $data                = [ ];
            $data['status']      = 401;
            $data['code']        = ErrorCodes::BAD_TOKEN;
            $data['message']     = 'Bad Token';
            $data['description'] = 'Token you provide with request is bad';
            return new JsonResponse( $data, 401 );
        }
    }

    /* ... */
}
```

Zdrojový kód 5.5: Metoda pro autentizaci na straně serveru (PHP)

5.4 Získávání dat a jejich tok aplikací

Dle návrhu v aplikaci jsou veškeré procesy řízeny architekturou Flux, jejíž diagram znázorňuje obrázek 4.6 z části práce věnované použitým technologiím. Následující odstavce popisují jednotlivé části přístupu architektury Flux pro akci vyhledání písničky pro její následné přidání do fronty.

Stránka pro přidání písničky je stejně jako ostatní stránky je React komponenta `AddSongView` z návrhu známá jako controller-view. Obsahuje metodu render, která vykreslí informace uživatelům a vyhledávací pole s tlačítkem pro potvrzení akce. Obě tyto pole jsou obaleny formulářem, na jehož o odeslání je navázána metoda komponenty `handleSubmit`.



Obrázek 5.1: Schéma databázového modelu

Uživatel zadá část názvu požadované písničky do vyhledávacího pole a akci potvrdí odesláním formuláře. Akce je odchycena metodou již zmíněnou metodou `handleSubmit`, která nad událostí zavolá metodu `preventDefault`. Tím je zastaveno výchozí zpracování formuláře, které by odeslalo data na server a překreslilo stránku. Překreslení stránky, jak již bylo v této práci několikrát zmíněno, není u SPA žádané. Z vyhledávacího pole je získána hodnota pomocí připraveného `this.refs.searchInput.value` a spolu s tokenem uživatele, který je v daném controller-view dostupný v proměnné `this.props.user.token`, předána metodě `searchSongs` pomocné třídy `ServerActionCreator`.

`ServerActionCreator`, stejně jako ostatní action creators popsané v části 4.5.2.4, vytváří akce. `ServerActionCreator` je speciálním případem, který ještě před vytvořením samotných akcí Fluxu komunikuje se serverem pomocí API.

Požadavek na server je zaslán skrz pomocné třídy nazvané jednoduše `Api`, která za pomoci knihovny `superagent` vytvoří AJAXový `GET` požadavek na zdroj `/api/v1/search-result` s query stringem hledané hodnoty.

Tento požadavek zpracuje na serveru příslušný controller Silexu. Ten pomocí metody `listSearch` zavolané nad instancí třídy `Google_Service_YouTube_Search_Resource` a předáním parametru `q` přesměruje dotaz na API Google respektive YouTube. Odpověď je v controlleru zpracována a ve formátu JSON zaslána zpět.

V případě úspěšného volání je vykonán callback třídy `ServerActionCreator`,

který byl na straně klienta zaregistrován. Tento callback vytvoří akci typu `'SHOW_SEARCH_RESULT'` a spolu s daty ze serveru je předán Dispatcheru, který je další součástí architektury Flux. Zdrojový kód metody `searchSongs` třídy `ServerActionCreator` znázorňuje zdrojový kód .

```
searchSongs: function(query, token) {
    Api.getSearchResult(query, token, function(result) {
        Dispatcher.dispatch({
            type: 'SHOW_SEARCH_RESULT',
            result: result
        });
    }, function() {
        Dispatcher.dispatch({
            type: 'API_ERROR'
        });
    });
}
```

Zdrojový kód 5.6: Metoda třídy `ServerActionCreator` komunikující se serverem a vytvářející akce Fluxu

Na akci typu `'SHOW_SEARCH_RESULT'` reaguje store `DataStore`, který uloží data vyhledaných písní do interní proměnné a zavolá funkci `emitDataSearchResultChange` daného uložiště, která vyšle událost `'data-search-result-change'`.

Komponenta `AddSongView`, která během svého vykreslení zaregistrovala privátní funkci `this._onSearchResultReceived` ji při přijetí události spustí. Tato metoda nastaví stav komponenty s využitím volání `DataStore.getSearchResult()`. Takto je celý tok dat a událostí v architektuře Flux dokončen.

Při změně stavu je komponenta díky knihovně React automaticky překreslena. Zobrazeny jsou výsledky z YouTube s možností přidání jedné z nich do seznamu přehrávání. Kliknutím na tlačítko „Zařadit do fronty“, které je dostupně u každého výsledku hledání, je spuštěn voláním `ServerActionCreator.queueYoutubeSong(/*...*/)` spuštěn nový tok dat.

Výše popsaným způsobem funguje prakticky celá klientská část aplikace. Základem je jednosměrný datový tok, který dělá chování uživatelského rozhraní a zároveň celé aplikace více předvídatelným z pohledu vývojáře.

Velkou výhodou je oddělení serverové část skrz API, které umožňuje snadnou změnu uživatelského rozhraní bez nutnosti měnit logiku aplikace na serveru. Druhou výhodou je okamžitá připravenost pro implementaci mobilní aplikace.

5.5 Priorita přehrávání

Během testování první verze se zvýhodněním uživatelů pomocí bodů, tak jak je uvedeno v zadání, tedy situace, kdy **uživatel, který má více bodu, muže**

zařadit svou písničku před ostatní, ukázala jako značně demotivující po uživatele s méně body. Tito uživatelé aplikaci přestali používat.

5.5.1 Přincip hlasování

Pro napravení situace byla navržena metoda přidělování bodů za samotným písním. Priorita přehrávání je pak určena na základně oblíbenosti. Jedná se o formu hlasování kdy každý uživatel ve skupině může dát kladný (pozitivní) nebo záporný (negativní) hlas zařazené písničce kliknutím na symbol zápěstí s palcem směřujícím vzhůru respektive dolů.

Jak využít tyto parametry ve formě hlasů k řazení od nejlepších po nejhorší řeší článek „How Not To Sort By Average Rating“ [36]. Uvádí špatné způsoby výpočtu „skóre“, kterým je dobré se vyvarovat a v závěru navrhuje správné řešení problému.

Jedním ze **špatných** řešení je rozdíl definovaný rovnicí:

$$skore = (pocet\ pozitivnich\ hlasu) - (pocet\ negativnich\ hlasu)$$

Proč je tento přístup špatný? V situaci kdy první prvek má 600 pozitivních a 400 negativních hodnocení má ve výsledku 60 % pozitivních hlasů. V druhém případě, kdy bude mít 5 500 pozitivních a 4 500 negativních hlasů, je výsledek 55 % pozitivních hlasů. Dosazeno do předchozí rovnice, algoritmus udělí v druhém případě $skóre = 1000$ (s 55 %), které je větší než $skóre = 200$ (s 60 %). To je bohužel špatně.

Druhým příkladem **špatného** řešení je výpočet skóre podle rovnice:

$$skore = prumer\ hodnoceni = \frac{(pocet\ pozitivnich\ hlasu)}{(celkovy\ pocet\ hlasu)}$$

Proč je tento přístup špatný? Průměrné hodnocení funguje správně pouze pokud existuje velké množství hlasů. Předpokládejme první prvek s 2 pozitivními hlasy a 0 negativních. Druhý prvek má 100 pozitivních hlasů a 2 negativní. Tento algoritmus zařadí druhý prvek (mnoho pozitivních hlasů) pod první prvek (opravdu málo pozitivních hlasů). To je znovu špatně.

Správné řešení je výpočtem spodní meze Wilsonova skóre konfidenčního intervalu pro Bernoulliho parametr (angl. Lower bound of Wilson score confidence interval for a Bernoulli parameter).

Jinak řečeno. Je nutné srovnat poměr pozitivních hlasů s nejistotou malých čísel. Toto naštěstí popsal matematicky v roce 1927 Edwin B. Wilson. Otázka zní: „Kolik je nejméně s pravděpodobností 95 % „reálný“ podíl pozitivních hlasů s daným hodnocením?“. Odpovědí je Wilsonovo skóre. Předpokladem jsou pouze pozitivní a negativní hlasy, potom spodní mez podílu pozitivních

hlasů je dána vzorcem

$$\frac{\left(\hat{p} + \frac{z^2_{\alpha/2}}{2n} - z_{\alpha/2}\sqrt{\frac{\hat{p}}{n}\left(1-\hat{p}\right) + \frac{z^2_{\alpha/2}}{4n^2}}\right)}{1 + \frac{z^2_{\alpha/2}}{n}}$$

kde p je sledovaný podíl pozitivních hlasů $z_{\alpha/2}$ je kvantil $(1 - \alpha/2)$ normálního rozdělení a n je celkový počet hlasů.

Pro zadaný interval spolehlivosti (konfidenční interval) je možné místo proměnné z použít konstantu. Pro pravděpodobnost 95 % je hodnota z rovna $1,96$.

Celý tento postup je využit pro seřazení písní ve frontě přehrávání od neoblíbenějších. SQL dotaz je znázorněn v kódu 5.7.

```
SELECT /* ... */,
  (SELECT COALESCE(SUM(uqs.like), 0)
     FROM user_queued_song uqs
     WHERE uqs.queued_song_id = qs.id
  ) AS likes,
  (SELECT COALESCE(SUM(uqs.dislike), 0)
     FROM user_queued_song uqs
     WHERE uqs.queued_song_id = qs.id
  ) AS dislikes,
  (SELECT likes - dislikes) AS simple_score,
  (SELECT likes + dislikes) AS total_votes,
  (SELECT simple_score > 0) AS positive,
  (SELECT COALESCE( ( (likes + 1.9208) / total_votes -
     1.96 * SQRT( (likes * dislikes) / total_votes + 0.9604) / total_votes)
      ↪  / (1 + 3.8416 / total_votes ), 0)
  ) AS score
FROM queue q
JOIN queued_song qs ON q.id = qs.queue_id
/* ... */
ORDER BY positive DESC, score DESC;
```

Zdrojový kód 5.7: Část SQL dotazu pro výběr písní seřazených od neoblíbenější

Tento přístup se setkal s pozitivním ohlasem mezi uživateli. Zvýšila se použitelnost a přirozené chápání významu hlasů.

Kapitola 6

Uživatelské testování a možnosti rozšíření

Závěrečná kapitola popisuje ve stručnosti výsledky uživatelského testování a možnosti rozšíření aplikace, které byly navrženy uživateli během testování nebo jsou v plánu v rámci budoucího rozšiřování aplikace MusicQ.

Nutno podotknout, že během testování **nebyl zjištěn problém** ve fungování aplikace. Proto je možné aplikaci v její současné podobě prohlásit za **funkční a otestovanou**.

Testování ukázalo, že aplikace splňuje všechny požadavky sepsané v kapitole 2 Analýza požadavků.

6.1 Uživatelské testování

Uživatelské testování probíhalo v cílové skupině uživatelů vybrané dle zadání práce. Jednalo se o skupinu 5-10 vývojářů v kancelářích otevřeného prostoru po dobu 1 týdne.

Vývojářům byl poskytnut přístup do aplikace rozšířením přihlašovacího formuláře o podporu protokolu LDAP, jež autentizoval uživatele s využitím firemního serveru. Přidáni byly automaticky do testovací fronty přehrávání.

Během testování byly zaznamenány následující žádosti a návrhy samotných uživatelů.

- **Automatické přehrávání** Jedním z nejžádanějších vlastností, které uživatelé postrádali je automatické přehrávání v případě prázdné fronty. Předpokládané fungování prázdné fronty přehrávání (po ukončení poslední skladby) je takové, že aplikace začne přehrávat hudbu od nejlepší hodnocené písničky. V případě, že je zařazena do seznamu nová písnička, bude jí dána priorita a spuštěna bude jako následující skladba.

- **Online uživatelé** Aplikace by mohla zobrazovat seznam aktuálně připojených uživatelů. Tento požadavek vnikl v souvislosti s hodnocením písniček, aby byl uživatel informován o celkovém počtu lidí, kteří mohou aktuálně hodnotit.

- **Organizace front** Dalším logickým požadavkem byla podpora pro chybějící funkčnost organizace front přehrávání. Možnost založit frontu přehrávání a přizvání úzkého okruhu uživatelů.

- **Seznam přehraných písniček** Aplikace v současné podobě nabízí pouze seznam písniček zařazených do fronty k následnému přehrání. Po ukončení přehrávání písnička „zmizí“. Uživatelé postrádali seznam písniček, které nedávno skončily.

6.2 Budoucí rozvoj

Aplikace svým návrhem přímo počítá s dalším vývojem a rozšiřováním funkčnosti. Všechny zaznamenané uživatelské poznatky a požadavky jsou brány v úvahu a mohou sloužit pro další postup.

Již při samotném návrhu a během implementace aktuální podoby aplikace byl zjištěn prostor pro budoucí vylepšení nebo rozšíření. Některá tato vylepšení jsou technologického charakteru, jiná přidávají novou funkcionalitu a vlastnosti budoucí verzi MusicQ. Jejich výčet s popisem je poslední částí práce.

- **Technologie Websockets** Využití technologie Websockets, která umožňuje komunikaci se serverem v reálném čase. Tento přístup umožní zobrazovanou dostupnost uživatelů v reálném čase. Další vylepšení, které počítá s touto technologií je oznámení o aktualizaci seznamu skladeb a odstranění současně používaného pravidelného dotazování na server. Tento přístup bude mít za následek zvýšení výkonu celé aplikace.

- **Chat** Další vlastnost, která se v budoucí aplikaci může objevit je chat. Chat by umožňoval komunikaci mezi uživateli, kteří jsou připojeni do stejné fronty přehrávání. Předchozí požadavek na použití Websockets vytvoření chatu přímo nahrává.

- **Opakované přehrávání písničky** V souvislosti s požadavkem uživatelů na seznam přehraných písniček se otevřel prostor pro možnost přidání již přehrané písničky znovu do seznamu jedním kliknutím.

- **Propojení s aplikacemi** Zajímavým rozšířením by bylo umožnění propojit MusicQ s externí aplikací, která by mohla například řídit prioritu seznamů. Tento požadavek by však vyžadoval větší analýzu a pokročilejší návrh.

Aplikace díky svému návrhu poskytuje snadnou možnost rozšíření o nové vlastnosti a funkce. Velkým krokem by bylo uvolnění zdrojového kódu pro skupinu vývojářů, kteří aplikaci v kanceláři využívají, tak aby sami mohli přidat požadovanou funkčnost.

Závěr

Předmětem práce bylo vytvořit webovou aplikaci, která umožní poslech hudby pro uzavřenou skupinu lidí. Vytváření sdílených playlistů mělo probíhat formou řazení písniček do seznamu s využitím priority uživatelů. Jako zdroj měla sloužit existující streamovací služba. Cíl práce se podařilo splnit a výsledkem je aplikace, která splňuje všechny body zadání.

Kvůli požadavku na nepřetržitý poslech hudby během práce s webovou aplikací byla využita architektura tzv. single-page aplikace. Aplikace nabírá dojmu klasické desktopové aplikace a plní požadovaný účel. Zároveň je díky této architektuře použit netradiční přístup, kdy se ke komunikaci se serverem a získávání dat používá pouze webové API dodržující principy REST, a uživatelské rozhraní je tak mimo získaných dat nezávislé na serverové části. Rozhraní pro přístup k datům je proto možné v budoucnu bez modifikace využít například pro implementaci aplikace pro mobilní telefony.

Pro vývoj bylo využito moderních technologií a postupů. Velmi zajímavou částí je Javascriptová knihovna React, která slouží pro implementaci uživatelského rozhraní. Zároveň aplikace přináší nové poznatky a principy řazení písniček podle oblíbenosti, které zavádí požadovanou prioritu v přehrávání.

Důraz byl kladen na správný návrh aplikace sdíleného prioritní webového playlistu a dodržení tohoto návrhu během fáze implementace. To umožňuje snadné navázání na tuto práci a možnost rozšíření aplikace o novou funkcionalitu ve všech možných směrech.

Aplikace je v současné době používána v prostředí kanceláře okruhu spolupracovníků a nahradila předchozí způsoby poslechu hudby.

Literatura

[1] Kadlec, L.: *Historie záznamu zvuku*. 2014. Dostupné z: http://test-nastroju.webnode.cz/nahravani/historie-zaznamu-zvuku/

[2] Frow, G.: *The Edison disc phonographs and the Diamond discs: a history with illustrations*. G.L. Frow, 1982, ISBN 9780950546254.

[3] Kroulík, L.: *Gramofon: vynález a historie přístroje, který změnil svět*. Březen 2014. Dostupné z: http://avmania.e15.cz/gramofon-vynalez-a-historie-pristroje-ktery-zmenil-svet

[4] Taintor, C.: *Chronology: Technology And The Music Industry*. Květen 2004. Dostupné z: http://www.pbs.org/wgbh/pages/frontline/shows/music/inside/cron.html

[5] Markoff, J.: *Apple Sells 70 Million Songs in First Year of ITunes Service*. Duben 2004. Dostupné z: http://www.nytimes.com/2004/04/29/technology/29apple.html

[6] Plug.dj: *Our Mission*. Dostupné z: https://plug.dj/about

[7] AngelList: *plug.dj*. Dostupné z: https://angel.co/plug-dj

[8] Spotify Press: *Information*. Dostupné z: https://press.spotify.com/us/information/

[9] Allsopp, A.: *Deezer vs Spotify vs YouTube Music Key vs Tidal comparison: What is the best music streaming service?* Duben 2015. Dostupné z: http://www.pcadvisor.co.uk/reviews/audio/3523953/deezer-vs-spotify-youtube-music-key-tidal-comparison-review/

[10] Deezer for developers: *Introduction: Content access rules*. Dostupné z: https://developers.deezer.com/guidelines#content-access

[11] Pan, J.: *Grooveshark Circumvents Mobile Bans by Launching an HTML5 Player*. Září 2012. Dostupné z: http://mashable.com/2012/09/05/grooveshark-html5-player/

[12] Vyleťal, M.: *Google nabízí investici do služby VEVO, chce si naklonit hudební vydavatele*. Únor 2013. Dostupné z: http://www.lupa.cz/clanky/google-nabizi-investici-do-projektu-vevo-chce-si-naklonit-hudebni-vydavatele/

[13] Nielsen, J.: 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group.*, Leden 1995.

[14] Wasson, M.: *Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET*. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[15] Zijderveld, W.-J.: *Using the Accept Header to version your API*. Říjen 2014. Dostupné z: http://labs.qandidate.com/blog/2014/10/16/using-the-accept-header-to-version-your-api/

[16] Saint-Andre, P.; Crocker, D.; Nottingham, M.: *Deprecating the "X-" Prefix and Similar Constructs in Application Protocols*. RFC 6648, Internet Engineering Task Force, Červen 2012. Dostupné z: http://tools.ietf.org/html/rfc6648

[17] Freed, N.; Klensin, J. C.: *Media Type Specifications and Registration Procedures - section 3.2. Vendor Tree*. RFC 4288, The Internet Society, Prosinec 2005. Dostupné z: https://tools.ietf.org/html/rfc4288#section-3.2

[18] Stripe, I.: *Api Reference - Versioning*. Březen 2015. Dostupné z: https://stripe.com/docs/api#versioning

[19] Stripe, I.: *API upgrades - API changelog*. Březen 2015. Dostupné z: https://stripe.com/docs/upgrades#api-changelog

[20] Wood, T.: *How are REST APIs versioned?* Březen 2012. Dostupné z: http://www.lexicalscope.com/blog/2012/03/12/how-are-rest-apis-versioned/

[21] Divilly, C.: *Why trailing slashes on URIs are important*. Březen 2013. Dostupné z: https://cdivilly.wordpress.com/2014/03/11/why-trailing-slashes-on-uris-are-important/

[22] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Network Working Group, Červen 1999. Dostupné z: https://tools.ietf.org/html/rfc2616#section-10.3.2

[23] Fog Geek Software: *Table Names: Singular or Plural?* Duben 2002. Dostupné z: http://discuss.fogcreek.com/joelonsoftware/default.asp?cmd=show&ixPost=5904

[24] Sirota, J.: *All those sses: what pluralization method do you use for your REST APIs?* Únor 2011. Dostupné z: http://jasonsirota.com/all-those-sses-what-pluralization-method-do-y

[25] Zazueta, R.: *API Data Exchange: XML vs. JSON*. Leden 2014. Dostupné z: http://www.mashery.com/blog/api-data-exchange-xml-vs-json

[26] Postel, J.: *Media Type Registration Procedure*. RFC 1590, Network Working Group, Březen 1994. Dostupné z: http://www.ietf.org/rfc/rfc1590.txt

[27] Franks, J.; Hallam-Baker, P. M.; Hostetler, J. L.; aj.: *An Extension to HTTP : Digest Access Authentication*. RFC 2069, Network Working Group, Leden 1997. Dostupné z: http://tools.ietf.org/html/rfc2069

[28] Hammer-Lahav, E.: *The OAuth 1.0 Protocol*. RFC 5849, Internet Engineering Task Force, Duben 2010. Dostupné z: http://tools.ietf.org/html/rfc5849

[29] Brail, G.: *Top Differences between OAuth 1.0 and OAuth 2.0 for API Calls*. Červenec 2010. Dostupné z: https://blog.apigee.com/detail/oauth_differences/

[30] Kalla, R.: *Designing a Secure REST (Web) API without OAuth*. Duben 2011. Dostupné z: http://www.thebuzzmedia.com/designing-a-secure-rest-api-without-oauth-authentication/

[31] Jones, M. B.; Bradley, J.; Sakimura, N.: *JSON Web Token (JWT)*. Technická zpráva, OAuth Working Group, Prosinec 2014. Dostupné z: https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32

[32] Pose, A.: *Cookies vs Tokens. Getting auth right with Angular.JS*. Leden 2014. Dostupné z: https://auth0.com/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/

[33] Facebook Inc.: *Flux: Overview*. 2014. Dostupné z: http://facebook.github.io/flux/docs/overview.html

[34] Google Developers: *Google Developers: YouTube Player Demo*. Leden 2015. Dostupné z: https://developers.google.com/youtube/youtube_player_demo

[35] Google Developers: *Google Developers: YouTube Player API Reference for iframe Embeds*. Dostupné z: https://developers.google.com/youtube/iframe_api_reference#Overview

[36] Miller, E.: *How Not To Sort By Average Rating*. Únor 2009. Dostupné z: http://www.evanmiller.org/how-not-to-sort-by-average-rating.html

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CD Compact disc

CORS Cross-origin resource sharing

CRUD create, read, update and delete

CSRF Cross-site Request Forgery

CSS Cascading Style Sheets

DBAL Database abstraction layer

DMCA Digital Millennium Copyright Act

DJ Disc jockey

DOM Document Object Model

DRY Don't Repeat Yourself

DVD Digital versatile disc

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

JWT JSON Web Token

LDAP Lightweight Directory Access Protocol

MP3 Moving Picture Experts Group-1, Layer-3

MVC Model–view–controller

PDO PHP Data Objects

PHP PHP: Hypertext Preprocessor

REST Representational state transfer

RFC Request for Comments

RIAA Record Industry American Association

SEO Search engine optimization

SPA Single-page application

SQL Structured Query Language

SSL Secure Sockets Layer

URL Uniform Resource Locator

UTF Unicode Transformation Format

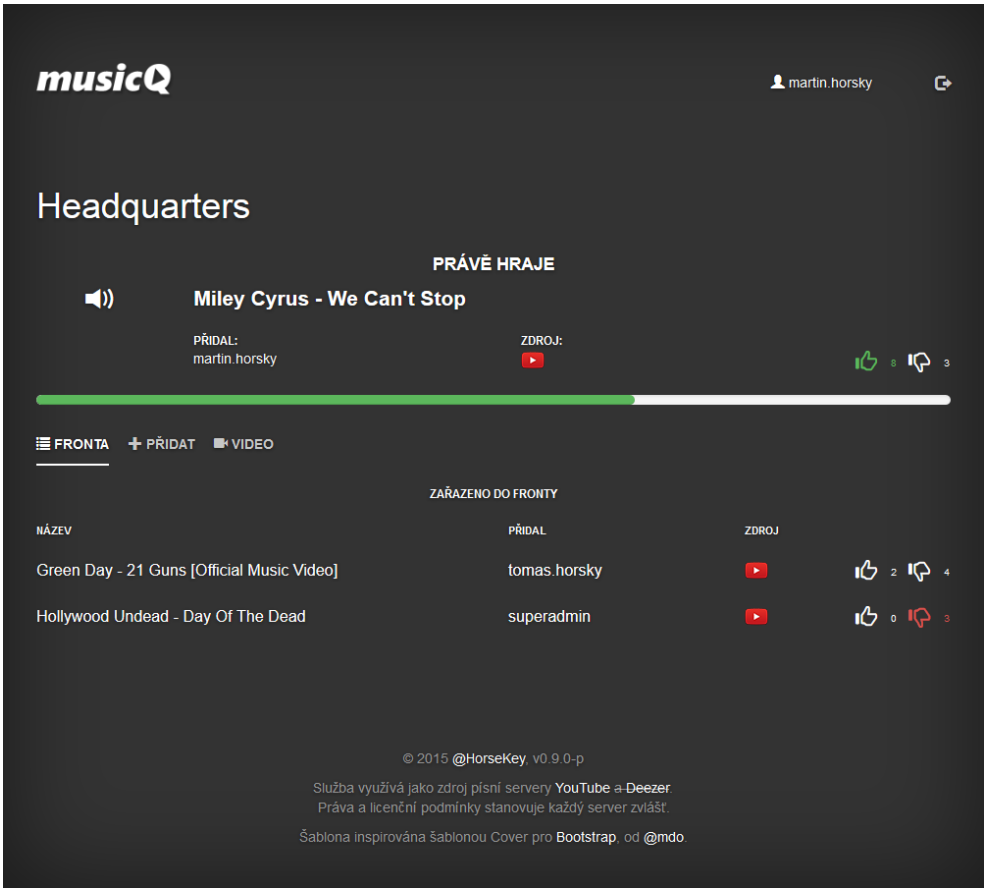
SDK Software development kit

XML Extensible Markup Language

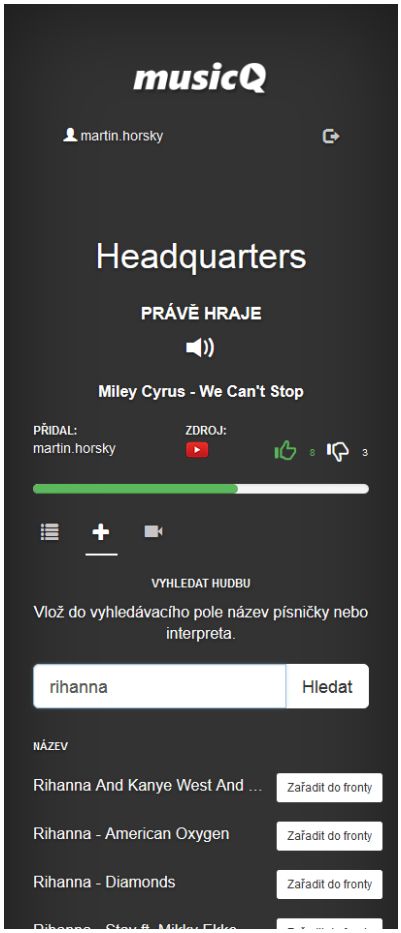
XSD XML Schema Definition

Příloha B

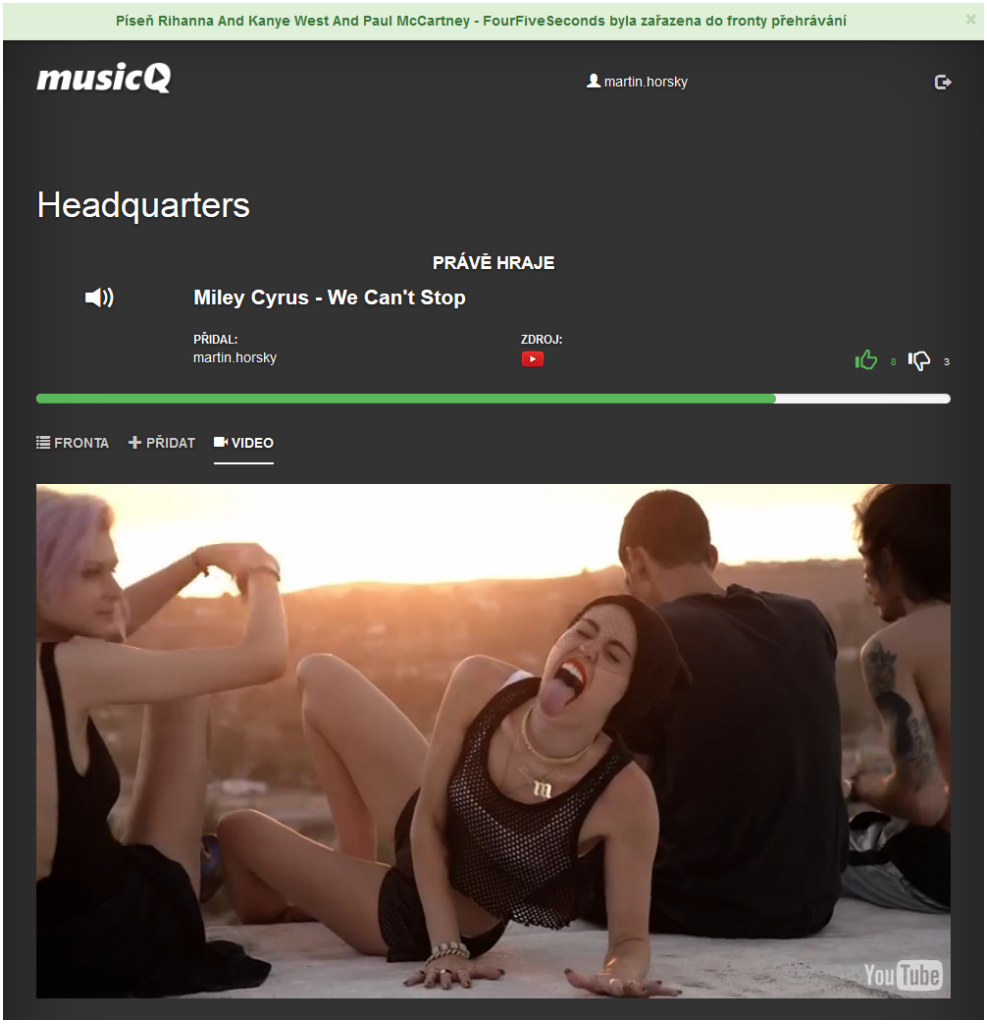
Ukázky aplikace



Obrázek B.1: Fronta přehrávání



Obrázek B.2: Mobilní zobrazení s výsledky hledání



Obrázek B.3: Video aktuálně přehrávané písničky s hláškou o přidání

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `screenshots` ..... ukázkové obrázky výsledné implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Horsky_Martin_2015.pdf` ..... text práce ve formátu PDF