FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Interaktivní Jukebox
Student:	Tomáš Rokos
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Cílem je navrhnout a implementovat interaktivní internetový Jukebox. Jukebox bude možné ovládat pomocí webové aplikace. Bude využívat hudební knihovnu digitální služby Spotify. V systému se bude jedno zařízení chovat jako Jukebox a ostatní zařízení budou moci přidávat písničky do jeho fronty. Aplikace bude napsána v návrhovém vzoru SSD (Split Stack Development). Všechny části projektu budou sdílet jazyk - Typescript. Její klientská část bude napsána pomocí knihovny React (server NextJS, správa stavů v Redux). Její serverová část bude napsána ve frameworku NestJS (přístup k API pomocí GraphQL, ORM - TypeORM).

1. Proveďte analýzu existujících řešení.
2. Proveďte softwarový návrh projektu.
3. Implementujte prototyp aplikace.
4. Aplikaci podrobte vhodným testům.
5. Připravte aplikaci pro nasazení minimálně pro testovací účely (Continuous Deployment).

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 27. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Jukex - Interaktivní Jukebox

Tomáš Rokos

Katedra softwarového inženýrství
Vedoucí práce: Ing. Filip Glazar

27. května 2020

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. května 2020 . . . . . .

Odkaz na tuto práci

Rokos, Tomáš. *Jukex - Interaktivní Jukebox*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW: ⟨`https://jukex.eu`⟩.

Abstrakt

Cílem bakalářské práce je navrhnout a realizovat jukebox 21. století, který lze spustit na jakémkoliv počítači s internetovým připojením a internetovým prohlížečem a který využívá hudební databázi streamovací služby Spotify.

V bakalářské práci je dále čtenář seznámen s historií jukeboxu. Je provedeno porovnání hudebních streamovacích platforem a dostupných aplikací. Dále jsou vybrány technologie pro implementaci prototypu jukeboxu. V realizační části je pak popsáno, jak tyto technologie byly použity.

Cíle bakalářské práce bylo dosaženo, realizovaný prototyp jukeboxu je dostupný na adrese `https://jukex.eu`. Jukebox může vyzkoušet každý s premium předplatným služby Spotify.

Klíčová slova Webová aplikace, Spotify, Interaktivní Jukebox, Jukex, Moderní technologie pro web, GraphQL, React, TypeScript, Single Page Application, hudební streamovací platformy

Abstract

The aim of this bachelor thesis is to design and implement a jukebox of 21st century that can run on any computer with internet connection and internet browser. It uses a music database of the streaming service Spotify.

This bachelor thesis brings jukebox history and compares web music platforms and available applications. Then the technologies for implementation of the jukebox prototype are introduced. Finally, the usage of these technologies is described in the realization part.

The aim has been achieved and the prototype of the jukebox is available on `https://jukex.eu`. Anyone with Spotify premium subscription can try it out.

Keywords Web application, Spotify, Interactive Jukebox, Jukex, Modern web technologies, GraphQL, React, TypeScript, Single Page Applicattion, music stream platforms

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Každý zažil situaci, kdy mu v jeho oblíbeném podniku, nebo na domácím večírku hraje hudba, která se mu opravdu nelíbí. Pro výběr hudby podle nálady a vkusu publika vznikla úplně nová pracovní pozice – DJ. Ne každý ale chce platit takového zaměstnance, proto byl postupně automatizován námi známým jukeboxem.

Jukebox funguje v podnicích od svého vynalezení úplně stejně již přes jedno století. S příchodem dotykových displejů získal sice nový kabát, ale princip zůstává stejný. Člověk přijde k „bedně“, zapátrá v databázi po jeho písničce a pokud tam je, tak ji přidá do hudební fronty jukeboxu. Zvuk nejčastěji hraje přímo z jukeboxu, místo z lepšího ozvučení, které většina podniků již má. Už od svého vzniku mají jukeboxy problém s databází písniček, která je omezená kapacitou zařízení. Podniky jsou závislé na firmě, která jukebox provozuje kvůli jeho opravám a aktualizacím hudební databáze.

Soukromých majitelů jukeboxů pro domácí použití je naprosté minimum, jen málokdo si pořídí nebo zapůjčí jukebox pro domácí večírek. Přitom většina hostů by možnost podělit se o své oblíbené skladby s přáteli určitě uvítala.

S vývojem technologií již ale není třeba problém s hudební databází řešit. Vznikly hudební služby, které mají pravidelně aktualizované databáze a nestojí víc, než deset dolarů měsíčně. Většina podniků i jednotlivců již má vlastní hudební aparaturu. Ozvučení celé místnosti už tedy nemusí být pouze z jednoho rohu, kde jukebox stojí.

Tato bakalářská práce popisuje návrh a vývoj aplikace, která pokryje všechny požadavky kladené na novodobý jukebox.

Jaké požadavky budou na jukebox kladeny naleznete v kapitole Cíl práce (1). Pro získání historického i obecného kontextu následuje historie jukeboxů a porovnání konkurenčních produktů v této oblasti. Představení streamovacích platforem a výběr nejlepší následuje v kapitole 4. Pro implementaci jukeboxu je třeba vybrat správné technologie a poté ukázat jejich využití v praktické části. V závěru poté najdete shrnutí poznatků z vývoje prototypu, zhodno-

cení rozhodnutí výběru technologií a možné další funkcionality prototypu do budoucna.

Kapitola 1

Cíl práce

Cílem práce je navrhnout a implementovat jukebox pro 21. století. Takový jukebox se dá spustit na jakémkoliv počítači s internetovým prohlížečem. Využívá databázi hudební streamovací služby Spotify. Databáze Spotify obsahuje přes padesát milionů písniček a každým dnem přibývají nové. [4]

Každý uživatel s kódem místnosti, který získá od provozovatele jukeboxu, bude moci přidat písničku do fronty. Přidávat písničky do fronty k přehrání bude možné z jakéhokoliv zařízení s internetovým připojením - v praxi nejčastěji z mobilních telefonů a tabletů a to bez nutnosti mít vlastní předplacený Spotify účet.

Na rozdíl od normálního jukeboxu není přidávání písniček jediná možná interakce. Uživatelé připojení k jukeboxu můžou nejen skladby do fronty přidávat, ale i hodnotit jednotlivé písně a měnit jejich pořadí ve frontě. Písně s vyšším počtem hlasů se ve frontě posunou nahoru a budou přehrány nejdříve. Písně bez preferenčních hlasů se budou přehrávat podle pořadí zařazení do fronty.

Tento jukebox bude využívat nejnovější technologie a je navrhován s možností škálování pro tisíce uživatelů. Nikdo nechce aktualizovat stránku při každém kliku, všechny změny se budou dít okamžitě a budou synchronizovány v reálném čase. Aby všechny tyto požadavky bylo možné splnit, bude aplikace využívat moderní softwarové návrhové vzory a jeho kód bude konzistentně formátován a strukturován.

Kapitola 2

Jukebox a jeho historie

Klasický jukebox je automatické zařízení, které přehrává hudbu na přání. Nejčastěji je nutné za přehrání skladby zaplatit. Poté si zákazník může vybrat z písní, které má v sobě jukebox uložené. Písně se většinou vybíraly pomocí kombinace dvou tlačítek, která unikátně identifikovala píseň. Jukebox lze označit jako jeden z hlavních faktorů pro šíření populární hudby a formování hudebního vkusu několika generací.

Termín jukebox vznikl kolem roku 1940 ve Spojených státech amerických ze slova „Juke“, nebo „Joog“, které znamená neukázněný, hlučný nebo hříšný.

První jukeboxy byly hrací skříňky, nebo dokonce automatická piána. Písničky zakódované pomocí kovových či dřevěných disků, válců nebo pruhů papíru přehrávaly reálné hudební nástroje.

Zlom přišel až 19. února 1878, kdy Thomas Alva Edison vynalezl fonograf. Toto zařízení dokázalo nahrávat a reprodukovat hudbu. S příchodem fonografu již nebylo potřeba pro přehrání písničky hudebních nástrojů.

Upravit fonograf na jukebox napadlo nejdříve Louise Glasse a Williama S. Arnolda. Vytvořili zařízení, které spouštělo fonograf po vhození pěticentové mince (niklák). První takový stroj byl instalován v baru Palais Royal Saloon v San Franciscu 23. listopadu 1889. Tento bar byl totiž jen 2 bloky od jejich kanceláří.

„*Úspěch hrací skříňky byl okamžitý. Do května následujícího roku se v různých sanfranciských barech objevilo dalších 14 jukeboxů a Glass s Arnoldem si svůj vynález nechali patentovat. Vhozenými mincemi se jim pořizovací cena údajně vrátila u každého automatu za pouhé dva týdny. A zkrátka nepřišel ani Edison. Ten si na své konto připsal deset tisíc dolarů za propůjčení licence na fonografy a drobný podíl ze zisku provozovaných automatů.*“ [6]

Tyto jukeboxy byly ale omezené pouze na jednu píseň. Až v roce 1950 Seeburg Corporation představilo gramofonový jukebox. Gramofonové jukeboxy postupně zcela ovládly trh v první polovině 20. století. Byly menší, lehčí a především nabízely možnost výběru skladeb.

S příchodem nových technologií CD a DVD začaly nahrazovat gramodesky datové disky, nutnost mechanického podávání disku do přehrávací mechaniky však stále zůstala. Další zvrat ve vývoji jukeboxů přinesly technologie umožňující ukládání digitalizované hudby na pevné disky ve formátu MP3, WAV nebo FLAC. Od nich byl už jen krůček k vybudování online prodeje hudby a nástupu platforem pro poslech na bázi předplatného. Právě s těmito moderními produkty budeme v následující kapitole porovnávat prototyp jukeboxu této bakalářské práce. [6], [7], [8]

Kapitola 3

Analýza konkurence

Konkurenční řešení pro prototyp této bakalářské práce lze rozdělit dvou skupin – fyzické a internetové jukeboxy. Produkt je implementován jako softwarové řešení, protože většina podniků má již instalováno vlastní ozvučení. Připojení počítače je záležitost jednoho kabelu, který připojí výstup zvukové karty k zesilovači a náhrada za drahý fyzický jukebox je na světě.

V tomto textu bude využíván pojem hostitel a uživatel. Hostitel vlastní zařízení, které funguje jako přehrávač písniček z fronty (počítač a zvukovou aparaturu). Uživatel vlastní zařízení, které může přidávat písničky do fronty (typicky chytré mobilní zařízení).

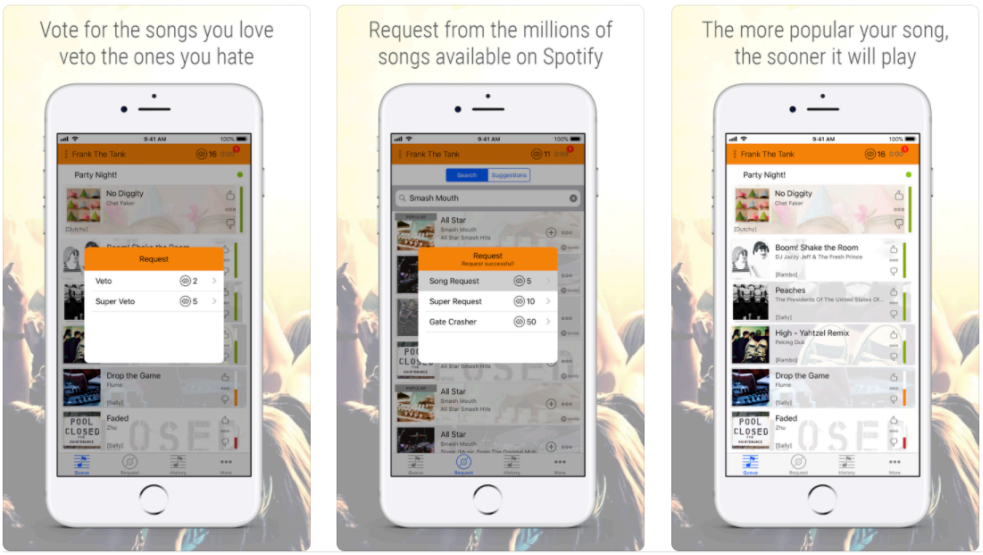
3.1 Jukestar

Jukestar je internetový jukebox pro domácí večírky. Produkt je ve formě mobilní aplikace. Nabízí dvě mobilní aplikace pro platformy Android a iOS.

První aplikace je pro hostitele a druhá pro uživatele. Hostitel tedy aplikaci nainstaluje a pouze přehrává písničky ze svého mobilního zařízení.

Mezi výhody aplikace patří, že je zdarma a využívá Spotify (4.3). Nabízí možnost hodnotit písničky a tím měnit jejich pozici ve frontě.

Hlavní nevýhodou je, že pro spouštění písniček musí mít hostitel i uživatelé nainstalovanou mobilní aplikaci. Hostitel tedy musí přesvědčit ostatní, aby si jí nainstalovali. Design této aplikace není příliš povedený. Celé řešení je v praxi navíc pomalé, po přidání písničky trvá i 30 sekund než se objeví ve frontě, na což uživatel často reaguje opakovaným přidáním stejné písničky. Nenabízí žádnou podporu pro veřejné zařízení (gastronomické podniky). Nemá možnost omezit délky písniček, žánru ani počtu písniček na uživatele. [9]



Obrázek 3.1: Aplikace Jukestar [1]

3.2 Mubo

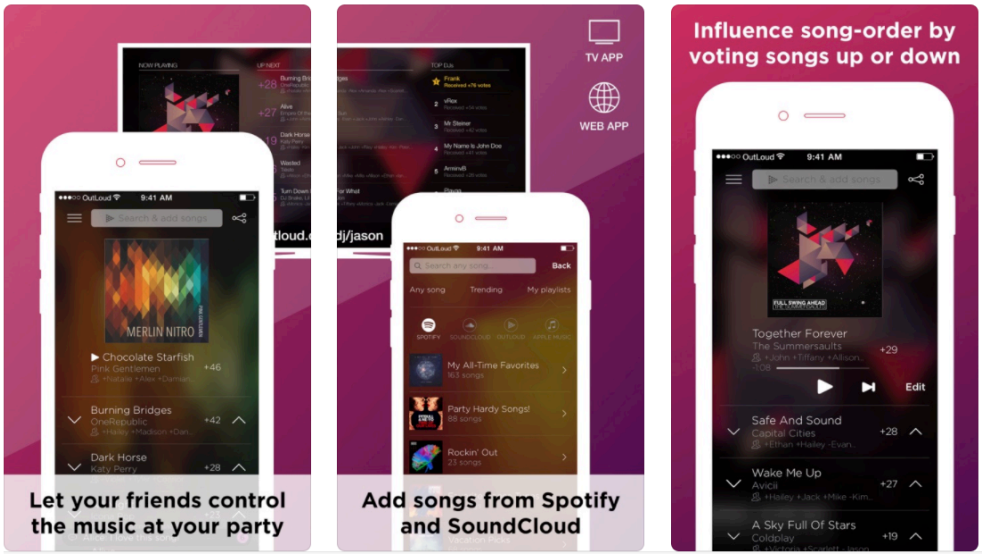
Mubo je internetový jukebox ve formě mobilní aplikace primárně určený pro veřejné prostory. Je třeba aby hostitel i uživatelé měli nainstalovanou Mubo aplikaci. Mobilní aplikace existuje pro Android i iOS. Podporuje ochranu místnosti pomocí GPS, takže ani při znalosti kódu místnosti nemůže náhodný uživatel přidávat písničky.

Celá aplikace má, na rozdíl od těch konkurenčních, líbivý design a intuitivní UI. Umožňuje i spustit na zařízení uživatele krátkou ukázku písničky, aby se mohl ujistit o správnosti výběru. Nabízí možnost připojení pouze pro uživatele, kteří se nacházejí ve stanovené maximální vzdálenosti od jukeboxu. Dovoluje regulovat počet přehraných písniček na uživatele. Podporuje více hudebních služeb.

Pro spouštění písniček ale musí hostitel i uživatelé nainstalovat mobilní aplikaci. Mubo nemá možnost restrikce žánru, nebo délky písniček. Nefunguje přihlášení přes sociální sítě a registrace do služby je matoucí. [10]

3.3 OutLoud

OutLoud je internetový jukebox, který nabízí mobilní i webovou aplikaci. Hostitel musí mít mobilní aplikaci, ale uživatelé se do místnosti mohou připojit i pomocí odkazu na webu. Podporuje Spotify, SoundCloud i Apple Music (4). Mohl by být využíván i ve veřejných prostorech, díky podpoře promítání právě přehrávané písničky na televizi. OutLoud nenabízí své služby v Evropě (aplikace je ke stažení pouze v americkém App Store). Aplikace tedy nebyla



Obrázek 3.2: Aplikace OutLoud [2]

vyzkoušena a její kladné a záporné stránky jsou získány z recenzí na App Store.

OutLoud má podporu pro více hudebních služeb. Má i aplikaci pro televizi a uživatelé mohou přidávat písničky do fronty pomocí webové aplikace. Podporuje hodnocení písniček ve frontě. Nabízí pojmenování místnosti a vytvoření kódu, přes který se uživatelé připojují.

OutLoud nenabízí své služby v Evropě. Design webové stránky na mobilu není dobře vyřešen. V recenzích na App Store má 3,7 hvězdičky [2]. Nenabízí žádnou restrikci žánru, nebo délky písniček. [11]

3.4 Jubox

Fyzický nástěnný jukebox, který nabízí dotykovou obrazovku. Určený pouze pro majitele veřejných prostor, nikoliv pro domácí večírky.

Jukebox do veřejného prostoru nainstalují zdarma jeho technici. Nabízí i karaoke a lze u něj platit kartou.

Databáze písniček je ale omezená výběrem společnosti. Nutnost fyzické přítomnosti techniků při instalaci a správě jukeboxu. Pro přidání písničky je nutné k jukeboxu dojít, protože jej nelze ovládat přes mobilní zařízení. Nelze měnit pořadí písniček. [3]

3.5 Thosli jukeboxy

Fyzický jukebox určený pouze pro veřejné prostory.

Obrázek 3.3: Dotykový jukebox od společnosti Jubox [3]

Dobrými vlastnostmi tohoto jukeboxu je jednoduché ovládání a funkčnost i bez internetu. Instalace jukeboxu do veřejných prostorů nabízí firma zdarma.

Jukebox má ale omezenou knihovnu písniček (110 tisíc), které se obměňují jen při příjezdu technika. Pro spuštění písničky je nutné platit hotově. Nelze měnit pořadí písniček. Nutnost fyzické přítomnosti techniků při instalaci a správě jukeboxu. [12]

3.6 Závěr

Přes velkou konkurenci v této oblasti, žádný jukebox nevyužívá přehrávání písniček přímo v prohlížeči.

Zařízení, které se chová jako hostitel, tedy musí mít mobilní operační systém a nelze využít počítač. Nutí tedy hostitele a hlavně uživatele instalovat do mobilního zařízení různé aplikace. Často je těžké aplikace nastavit a mají v některých případech velkou prodlevu mezi přidáním písničky a zobrazením této změny (operace se nedějí v reálném čase).

I přes velkou nabídku funkčností u konkurenčních jukeboxů, jsou často nedotažené ty úplně základní koncepty a design. Prototyp této bakalářské práce by měl kombinovat různé funkce jednotlivých aplikací zmíněných v této kapitole. Zároveň by měl být ve formě webové aplikace.

Kapitola 4

Hudební streamovací platformy

Pro tuto bakalářskou práci je třeba vybrat hudební streamovací platformu, aby prototyp mohl uživatelům nabídnout dostatečně velkou databázi písniček. Při výběru budou zohledněna hlavně tři kritéria. Popularita dané streamovací platformy, velikost databáze a její API (5.3). Musíme vybrat takovou platformu, která dovoluje službám třetích stran spouštět jejich digitálně chráněnou hudbu v prohlížeči.

Proto jsou vynechány streamovací platformy, která nemají žádné API (Google Play Music, Tidal a Youtube Red).

V této sekci je probereme jednotlivé hudební streamovací služby a vybereme jednu primární platformu, která bude využita v prototypu této bakalářské práce.

4.1 SoundCloud

SoundCloud je hudební platforma pro sdílení hudby. Každý registrovaný uživatel na ni může nahrát písničku a přeposlat jí k poslechu ostatním. Je tedy velmi atraktivní pro začínající umělce, kteří tu mají velkou komunitu. Platforma je pro všechny uživatele zdarma.

SoundCloud nabízí API a umožňuje přehrávat hudbu přímo v prohlížeči na jakémkoliv zařízení.

Bohužel ale nemá mnoho veřejně známé hudby. Každý na SoundCloud může přidat jakýkoliv hudební soubor. Přidávané skladby nejsou samotnou službou hodnoceny ani posuzovány. [13]

4.2 Apple Music

Apple Music je streamovací platforma od firmy Apple. Má velkou databázi písniček a hojnou uživatelskou základnu. Firma dává k zakoupenému Apple zařízení Apple Music až na tři měsíce zdarma. Má dokonce i exkluzivní smlouvy

s některými umělci, kteří hudbu publikují pouze na tuto platformu. Platforma je placená předplatným, které stojí 149 Kč měsíčně.

Apple Music nabízí API a umožňuje přehrávat hudbu přímo v prohlížeči na jakémkoliv zařízení.

I přes název Apple Music není svázaná pouze s zařízeními Apple. Platforma nabízí aplikaci i pro Android a Windows. Nemá webovou aplikaci. [14]

4.3 Spotify

Spotify je švédská hudební streamovací platforma. Na trhu s takovými službami byla jako jedna z prvních. Nabízí aplikace pro všechny platformy i pro web. Nabízí velkou databázi komerčních písniček (4.1).

Nabízí i verzi zdarma, kde uživatel může poslouchat jen náhodně poskládané písničky z výběru a má omezený počet jejich přeskočení. Prémiový účet dovolí přehrát jakoukoliv písničku bez omezení. Měsíc prémiového účtu nabízí zdarma a poté stojí 6 euro za měsíc, což je kolem 160 Kč.

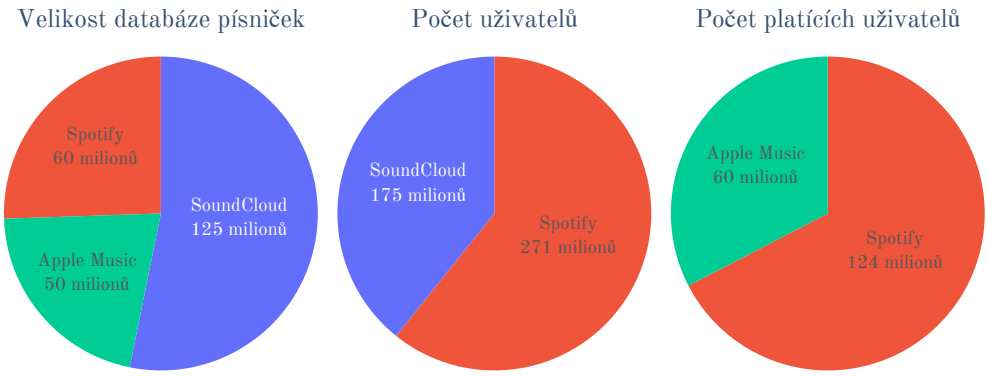
Spotify je i sociální sítí. Umožňuje uživatelům vidět, co poslouchají jeho přátelé a jejich oblíbené písničky.

Nabízí API a umožňuje přehrávat hudbu přímo v prohlížeči. Prohlížeč a dané zařízení ale musí být podporováno. [15]

4.4 Závěr

Platforma SoundCloud má velkou databázi písniček. Většina z této databáze jsou písničky nahrané komunitou zdarma k poslechu. Uživatel by tedy neměl dostatek komerčních písniček k přidávání do fronty.

Spotify i Apple Music mají dostatečně velké databáze komerčních písniček a podobné ceny za předplatné. Obě nabízí přehrávání hudby v prohlížeči, přičemž Apple Music má větší podporu pro zařízení i prohlížeče, na kterých je hudba spouštěna.



Obrázek 4.1: Porovnání streamovacích služeb [4], [5]

Uživatel určitě nebude chtít kupovat předplatné streamovací platformy jen kvůli využití v prototypu této bakalářské práce. Proto o výběru rozhodla popularita platforem.

Podle zdroje musically má Spotify 124 milionů uživatelů s prémiovým předplatným, zatímco Apple Music má 60 miliónů. Spotify má tedy dvakrát větší uživatelskou bázi. [5]

Pro tuto bakalářskou práci bude tedy využita streamovací platforma Spotify i přes omezení na zařízení a prohlížeč. Apple Music bude skvělá volba pro možné budoucí rozšíření aplikace.

Kapitola 5

Analýza a výběr technologií

Aby mohl být jukebox dobře použitelný a reagoval na akce uživatele, je třeba synchronizovat data v reálném čase. Pro synchronizaci byly tedy vybrány takové technologie, které podporují websockety (5.2). Na frontendu musí být změny reflektovány okamžitě a správa stavu aplikace bude se synchronizací v reálném čase komplexní.

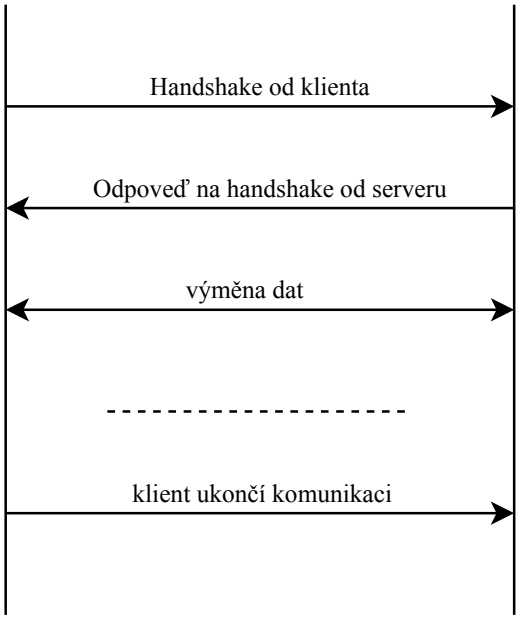
Jukebox nesmí přestat přehrávat písničky, i když se hostitel přesune na jinou stránku prototypu (například nastavení). Není tedy možno využít pouze tradiční technologii s vykreslením HTML na straně serveru, protože by přehrávač přestal hrát při každém přesměrování. Pro tento případ se vyplatí využití některého z velkého množství JavaScript frameworků.

5.1 HTTP

Hypertext Transfer Protocol (HTTP) je protokol na aplikační vrstvě pro přenos hypermedia dokumentů, jako například HTML. Byl navržen pro komunikaci mezi webovými prohlížeči a webovými servery, ale může být také využit na jiné účely. (překlad autora [16])

HTTP protokol definuje, jaké požadavky od prohlížeče by měl server přijímat a jakým způsobem na ně má odpovědět. Přidává i další nástroje jako hlavičky, cachování a další. Tento protokol je bezstavový, takže v požadavku musí být uvedena všechna data (například autorizace uživatele). Nejčastěji využívá pro komunikaci TCP/IP, protože potřebuje spolehlivý neztrátový protokol. [16]

HTTP je zde zmíněn kvůli následujícímu tématu WebSocket (5.2), který ho využívá.



Obrázek 5.1: Komunikace pomocí WebSocketu

5.2 WebSocket

Pro realizaci prototypu nestačí jednostranná komunikace pomocí požadavků na server. Server totiž musí notifikovat všechny uživatele připojené k jukeboxu o změně fronty hrající písničky nebo jiných dat. V minulosti se tento problém řešil pomocí konstantního dotazování serveru, zda nedošlo ke změně dat. Tento způsob byl ale velmi neefektivní, protože se musela vždy posílat celá HTTP hlavička a klient dotazoval server, i když nebylo potřeba.

Přesně pro tento záměr vznikl protokol WebSocket. Byl navržen, aby umožnil obousměrnou komunikaci, která využívá HTTP jako transportační vrstvu. Využívá tedy hlaviček tohoto protokolu, takže se dá implementovat stejná autorizace pro požadavek, jako pro připojení k WebSocketu.

Websocket je protokol, který dovoluje perzistentní TCP spojení mezi serverem a klientem, čímž si můžou vyměňovat data kdykoliv v reálném čase. (překlad autora [17]).

Protokol má dvě části – vytvoření spojení a přenos dat. Pro vytvoření spojení klient-server se využívá inicializační komunikace (handshake). Klient odešle požadavek se specifikovanou hlavičkou na server. Ten může odpovědět pomocí jakéhokoliv kódu HTTP protokolu. Když je s požadavkem všechno v pořádku, odpoví pomocí kódu „101 Switching Protocols“.

Tím je komunikace potvrzená a dvojice si může vyměňovat data. Pro ukončení komunikace je definována speciální sekvence, která se pošle jako data. Ukončit komunikaci můžou obě strany. [18]

5.3 API

API je sada definic a protokolů pro vytváření a integrování aplikačního softwaru. Zkratka API znamená Application Programming Interface. (překlad autora [19])

Takové rozhraní umožňuje komunikaci aplikací nebo služeb. Při komunikaci aplikace zná vstupní a výstupní formát dat, ale nemusí znát jeho implementaci. Termínem API se často nazývá HTTP API. Obecně tomu tak nemusí být, lze mluvit i o přístupu k funkcím, které daná knihovna nabízí.

5.4 Spotify Web Playback SDK

Pro funkčnost jukeboxu je třeba najít cestu pro přehrávání autorsky chráněné hudby přímo v prohlížeči. Je tedy nutné, aby hudební streamovací služba měla knihovnu implementující tuto funkcionalitu.

Web Playback SDK je klientská JavaScriptová knihovna, která umožňuje vytvořit nový přehrávač v Spotify Connect a přehrát jakoukoliv písničku z Spotify v prohlížeči díky Encrypted Media Extensions. (překlad autora [20])

Pomocí Web Playback SDK lze využívat většinu služeb Spotify přímo v prohlížeči. Toto SDK je stále ve fázi vývoje beta. Není tedy tolik služeb využívající toto SDK.

Spotify využívá specifikaci Encrypted Media Extensions, které umožňuje přehrávání autorsky chráněného obsahu v prohlížeči. Tato specifikace musí být prohlížečem podporována. V budoucnu chce Spotify rozšiřovat své služby i na mobilní zařízení, čímž by hostitel již nemusel být počítač.

Tabulka 5.1: Zařízení podporující Spotify Web Playback SDK

Operating System	**Browsers**	**Status**
Mac/Windows/Linux	Chrome, Firefox, IE*	Supported
	Microsoft Edge	Supported
	Safari	Not Supported
Android	Chrome, Firefox	Not Supported
iOS	Safari, Chrome	Not Supported

5.5 OAuth 2.0

OAuth 2.0 je protokol, který umožňuje uživateli předat limitovaný přístup k prostředkům jedné stránky jiné stránce, bez nutnosti předání přístupových údajů. (překlad autora [21])

Tento způsob autorizace bude využit k získání přístupu k Spotify účtu uživatele. Uživatel musí povolit přístup k jeho přehrávači.

Pro účely průchodu budeme nazývat aplikaci, která chce využít OAuth 2.0 „aplikace“. Službu u které chceme implementaci tohoto protokolu využít, budeme nazývat „webová služba“. Obecný průchod je takový, že aplikace přesměruje uživatele na OAuth 2.0 bránu dané webové služby. Tam potvrdí přístup k prostředkům jeho účtu a vrátí se zpět do aplikace.

Aplikace získá od webové služby autorizační kód, kterým si může vyžádat dvojici tokenů. Token v kontextu protokolu je náhodný alfanumerický řetězec o alespoň 512 bytech.

První token se nazývá access token a má omezenou dobu platnosti. Tento token se posílá v hlavičce požadavku, kterým chceme přistupovat k prostředkům. Druhý token se nazývá refresh token. Refresh token je nejčastěji permanentní, dokud uživatel nezakáže přístup aplikaci k webové službě. Využívá se k získání nového access tokenu po jeho vypršení. [21]

5.6 Škálování aplikací

Pod škálováním aplikace je v kontextu této bakalářské práce myšleno zvýšení výkonu aplikace podle potřeby. Jsou dva přístupy k škálování aplikace – horizontální a vertikální.

5.6.1 Vertikální škálování

Vertikální škálování může být například upgrade serverových součástek, nebo zaplacení lepšího hardwaru. Toto škálování je jednoduché, protože pro vyměnění procesoru za lepší není potřeba řešit architekturu aplikace.

Je ale konečné, protože i superpočítač může být pouze konečně výkonný. Pro vyšší výkon roste cena od jistého bodu v horizontálním škálování exponenciálně.

5.6.2 Horizontální škálování

U tohoto přístupu je zvýšen výkon pomocí vytěžování zátěže. Taková aplikace poběží na více serverech. Požadavek bude řešit takový server, který je nejméně zatížený a má optimální odezvu. Pro tento přístup musí být správně napsán i program, protože servery budou pravděpodobně sdílet nějaké prostředky.

5.7 JavaScript

JavaScript (JS) je lightweight, interpretovaný, nebo just-in-time kompilovaný programovací jazyk s first-class funkcemi. I když je nejvíce známý pro vývoj webových stránek, mnoho prostředí mimo prohlížeč ho také využívá, jako například Node.js, Apache CouchDB a Adobe Acrobat. JavaScript je prototypovaný, multiparadigmatický, jednojádrový, dynamický jazyk podporující

objektově orientované, imperativní a deklarativní programování. (překlad autora [22]).

Javascriptový standard ECMAScript prošel za poslední roky obrovskou proměnou a vývojem. Díky tomu, že JavaScript je nejpopulárnějším jazykem prohlížeče, jsou vynaloženy prostředky velkých firem pro jeho optimalizaci a zrychlení jeho interpreteru.

5.8 NodeJS

Node.js je technologie umožňující využívat optimalizované jádro pro interpretování JavaScriptu. Může tedy místo prohlížeče běžet přímo na jakémkoliv podporovaném zařízení. Lze ho tedy využívat jako jakýkoliv jiný interpretovaný jazyk. Node.js přidává standardní knihovny pro práci se soubory, http server a další funkcionality.

Jako asynchronní událostmi řízený JavaScriptový runtime, je Node.js navržen pro vytváření škálovatelných síťových aplikací. (překlad autora [23]).

S Node.js přišel i systém pro správu modulů Node Package Manager. NPM má jednu z nejaktivnějších vývojářských komunit ze všech správců balíčků. Počet balíčků napsaných pro Node.js velmi rychle roste – v roce 2019 přibývalo 896 Node.js modulů denně. [24]

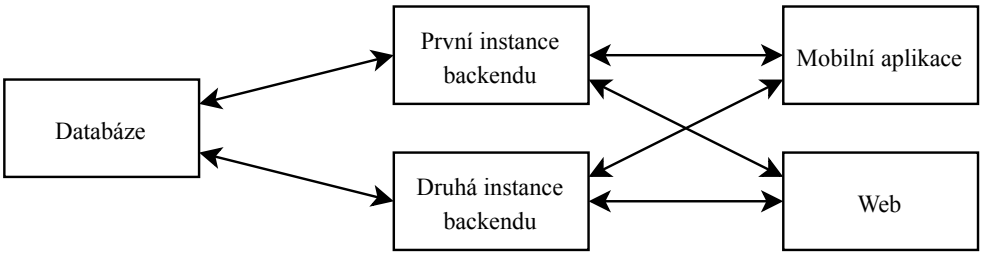
5.9 Split Stack Development

SSD je návrhový vzor softwaru, který odděluje backend a frontend aplikace do dvou nezávislých částí. Tyto části spolu komunikují pomocí API (5.3). Správné oddělení částí aplikace je v počátcích vývoje těžší na implementaci. V dlouhodobém hledisku ale tento vzor přináší mnoho výhod.

Díky rozdělení aplikace lze jednoduše přidávat, nebo měnit jednotlivé komponenty aplikace. To znamená, že pokud bychom se rozhodli přejít u frontendu na jiný javascriptový framework, nemusíme se nijak dotýkat serverové části. Stejně tak v případě, že bychom chtěli zvolit na backendu jiný programovací jazyk, nebo jen výkonnější framework, není nutné upravovat frontend aplikace. Využití tohoto návrhového vzoru zaručí jeho snadnou rozšiřitelnost na další platformy, protože všechna potřebná logika je již implementovaná na backendu.

Další výhoda je vidět při nezávislé práci více lidí na aplikaci. V budoucnu lze najmout vývojáře, který umí programovat pouze jednu část aplikace. Jednotlivé části na sebe nutně nemusejí čekat, protože si frontendová část nejdříve navrhne design s testovacími daty a poté jen přidá volání pro reálné data. Je taky jednoduché produkt outsourcovat jiné firmě, která navrhne a implementuje pouze jednu platformu.

Za předpokladu, že jsou jednotlivé části opravdu nezávislé, je možné aplikaci horizontálně škálovat (5.6). Při větším zátěži serveru dodávající frontend



Obrázek 5.2: Návrh aplikace pomocí SSD

aplikace lze zvýšit výkon pouze této části, nebo přesunout zátěž na jiný identický server. Stejně lze škálovat všechny části aplikace. [25]

5.10 Docker

Prototyp této bakalářské práce je třeba nasadit na server s operačním systémem Linux. Musí být také připravený na budoucí horizontální škálování aplikace.

Docker je open source projekt pro automatizaci nasazení aplikací jako přenosných, soběstačných kontejnerů, které lze spustit v cloudu nebo v lokálním prostředí. (překlad autora [26])

Docker vyřeší problém „funguje to na mém zařízení“ a ještě přidá velký ekosystém nástrojů pro nasazování aplikace. Díky Dockeru se vývoj aplikace nemusí vázat na operační systém. Pro prototyp šlo sice zvolit normální linuxovou virtualizaci, ale ta nemá tak dobré nástroje pro škálování (projekty jako Kubernetes a mnohé další).

V praxi se Docker využívá skoro stejně jako lokální, nebo virtualizovaná linuxová instalace. Je třeba vytvořit nějakou formu konfiguračního souboru, která nainstaluje potřebné závislosti a služby. V tomto prostředí s nainstalovanými závislostmi (Docker container) lze aplikaci spustit, a dokonce i přímo vyvíjet.

Docker dobře pracuje v kombinaci s návrhovým vzorem SSD (5.9). Pokud na aplikaci pracuje více lidí, je pouze třeba, aby všichni měli nainstalovaný Docker. Není nutné rozbíhat lokální prostředí, nebo instalovat určitý operační systém pro vývoj.

5.11 Model View Controller

MVC je softwarová architektura, podle které by měl programátor strukturovat kód do tří nezávislých komponent.

Komponenta Model (M) by měla řešit logiku a získávání dat. Komponenta View (V) by měla zobrazovat data uživateli. Komponenty View a Model o sobě nevědí. Pro jejich komunikaci slouží třetí komponenta, která je spojuje. Ta se

nazývá Controller (C). Controller komponenta zavolá Model a získá data, která pak předá View a ta je zobrazí.

5.12 TypeScript

TypeScript je typovaná nadstavba JavaScriptu, která se kompiluje do prostého JavaScriptu. (překlad autora [27])

Pro složitější aplikace je statické typování obrovskou výhodou. Díky TypeScriptu je možné využívat výhod statického typování i v JavaScriptu. TypeScript nabízí kompilátor, který typově kontroluje kód, přehledně ukazuje chyby a vygeneruje prostý JavaScript kód. [27]

TypeScript do typování ale nenutí, takže lze využít kladné vlastnosti typovaného jazyka pouze tam, kde je to potřeba.

Komunita kolem TypeScriptu aktivně přidává typy i pro knihovny napsané v prostém JavaScriptu. Díky těmto typům, editor našeptává i při volání knihovních funkcí, které nebyly napsané v TypeScriptu.

5.13 Technologie pro komunikaci

Pro komunikaci mezi frontendem a backendem bude využit protokol HTTP. Prohlížeče tento protokol nativně využívají. Pro formát dat v těle požadavků byl zvolen JSON. JSON se stal „de facto“ standardem pro výměnu dat mezi backendem a frontendem. Byl zvolen jako alternativa k XML, kvůli jeho podpoře mnoha jazyky a i nativně v JavaScriptu.

Je třeba rozhodnout o protokolu pro vyžádání dat od serveru. Populární přístupy k vývoji API jsou REST a GraphQL. Obě dvě řeší problém přístupu k prostředkům pomocí HTTP, ale používají se jiným způsobem.

5.13.1 REST

REST je architektura pro návrh API. Tato architektura vyniká svou jednoduchostí a snadnou implementací. Využívá 4 HTTP verbs pro manipulaci s prostředky – GET, POST, PUT, DELETE. Každý prostředek má svou unikátní URI, na kterou se požadavek zasílá. Například pro přístup k právě přehrávané písničce použijeme kombinaci HTTP verbu (GET) a unikátního URI (https://jukex.eu/api/currently-playing).

Problém s touto architekturou je, že dává programátorovi velkou svobodu pro návrh. Právě v návrhu REST API se dělá mnoho chyb. Problémy jsou hlavně s názvy URI (neunikátnost, množný/jednotný tvar), se správným využití HTTP verbs (nedodržení vlastnosti metod, idempotentnosti) i s konzistentním vracením chybových hlášek.

Kvůli velké svobodě návrhu REST API je nutné k API vždy dodat dokumentaci. Ta obsahuje, jaké mají prostředky URI, jaké tělo vrací v odpovědi

Kód 5.1: Ukázka GraphQL dotazu na právě hrající písničku

```
query currentTrack($roomIdentifier: String!) {
  currentTrack(roomIdentifier: $roomIdentifier) {
    name
    albumArt
    position
    duration
    createdAt
    artists {
      name
    }
  }
}
```

a jakých typů jsou klíče v JSON objektu. Vývojář může dokonce vrátit jinou strukturu JSON objektu na základě požadavku.

5.13.2 GraphQL

GraphQL je dotazovací jazyk pro API a knihovna pro předání existujících dat. (překlad autora [28])

GraphQL přichází se striktnějšími pravidly v přístupu k jednotlivým prostředkům. Pro manipulaci nebo získání dat je třeba použít dotazovací jazyk. Pro manipulaci s daty použijeme mutaci (mutation), zatímco pro získání dat dotaz (query). Na serveru je dotaz zpracován a jsou zkontrolovány typy vstupů i struktura JSONu. Server je nucen vždy specifikovat strukturu JSONu a vrátit tělo odpovědi podle této struktury.

Díky využití dotazovacího jazyka je možné zvolit jaké klíče s daty má server vrátit. Je také možné poslat několik dotazů s jedním HTTP požadavkem. Server může specifikovat i relace, ke kterým je možné v dotazu přistoupit. Klíče této relace jsou definovány v typu dotazu. Je tedy možné například u písničky získat název a stáří jejího autora.

Vždy je specifikována struktura vstupu i výstupu a o jejich předání se stará knihovna pro implementaci GraphQL v daném jazyce (za předpokladu, že si nechce vývojář protokol implementovat sám). Dokumentaci je tedy možné generovat z těchto typů. Klient využívající toto API tedy vždy ví, kde najít chybovou hlášku, nebo jednotlivé klíče, ke kterým může přistoupit. GraphQL nabízí prvotřídní nástroje pro testování požadavků a zobrazení dokumentace. Lze také v kódu specifikovat zastaralé klíče, které jsou zde jen kvůli zpětné kompatibilitě.

Při komunikaci s GraphQL API využíváme jen jednu URI a HTTP verb

POST. Do těla požadavku vložíme dotaz, na který chceme, aby server odpověděl.

GraphQL nabízí i podporu pro realtime spojení pomocí WebSocketu. Takové spojení se v GraphQL nazývá subscription. I u takové komunikace server specifikuje strukturu, kterou může klient odebírat v reálném čase. [28]

5.13.3 Závěr

Výběr mezi GraphQL a REST je velmi závislý na potřebách aplikace. Nedá se říci, že by byla jedna technologie lepší než druhá. GraphQL má striktní typování, automatické generování dokumentace a lepší práci s objekty, co mají relace. REST je zase jednoduchý, více variabilní a má lepší možnosti cachování.

Pro prototyp byla zvolena technologie GraphQL. Jedním z hlavních důvodů je podpora WebSocketu (ve formě subscriptions), striktní typová syntaxe a využití možného načtení relací k prostředkům (data v aplikaci jsou často provázaná).

5.14 Technologie pro frontend

V této aplikaci bude frontend velmi bohatý na aktualizace jeho prvků v reálném čase (např. fronta musí být okamžitě aktualizována při přidání písničky na všech připojených zařízeních). Změn dat bude probíhat mnoho, je tedy třeba vybrat i architekturu pro aktualizaci a uložení stavu aplikace.

Pro lepší rozšiřitelnost aplikace i za hranice musí být aplikace vícejazyčná. Je tedy třeba vybrat knihovnu pro implementaci překladů aplikace. V základní fázi bude aplikace podporovat češtinu a angličtinu.

Je třeba, aby přehrávač běžící na hostiteli nepřestal přehrávat při přesměrování na jinou stránku jukeboxu, nebo při jiné interakci. Frontend bude běžet v prohlížeči a je nutno zamezit obnovení (refresh) stránek.

Přehrávač jukeboxu se musí synchronizovat s ostatními uživateli připojenými na jukebox.

5.14.1 Design aplikace

Aby uživatelé aplikaci rádi využívali, musí být přívětivá a mít nadstandardní design. Protože předmětem této bakalářské práce není designový návrh, byla pro tyto účely zakoupena šablona. V šabloně jsou navržené některé komponenty a skrz celý návrh je konzistentní design (vzdálenosti prvků, barvy a další). Využitá šablona se jmenuje Dashkit. [29]

Implementace těchto komponent do JavaScriptového frameworku ale již součástí bakalářské práce je. Šablona byla závislá na jQuery a jiných knihovnách, které v prototypu nejsou potřeba. Bylo tedy třeba nahradit funkcionalitu komponent vlastní implementací.

5.14.2 Document Object Model

Při změně stavu aplikace je třeba překreslovat stránku v prohlížeči (např. při přidání písničky musíme přidat písničku do uživatelského rozhraní). Takové překreslení lze udělat jen díky DOMu.

Document Object Model (DOM) spojuje webové stránky se skripty, nebo programovacími jazyky skrz reprezentaci struktury dokumentu (např. HTML reprezentující webovou stránku) v paměti.

Webové stránky jsou napsány jako dokument ve značkovacím jazyce. DOM poté reprezentuje tento dokument pomocí logického stromu. Značky poté reprezentuje jako uzly v logickém stromu, které se můžou větvit. V tomto stromu nám umožňuje dělat úpravy pomocí jeho API. Dokáže i přidat listenery, které umí odchytit různé události na jeho uzlech (kliknutí, táhnutí myší a další).

Díky DOMu lze překreslit pouze část stránky, bez nutnosti aktualizace v prohlížeči. [30]

5.14.3 Single Page Application

Při vyvíjení webové aplikace je možné využít dvě možnosti přístupu k vykreslování jejího rozhraní.

Lze vytvořit HTML na serveru, které se následně odešle prohlížeči k vykreslení. Prohlížeč vyšle požadavek pro stránku a server mu pošle celé HTML. Prohlížeč získá potřebné závislosti (např. styly, obrázky a JavaScript) a poté stránku vykreslí. Pokud uživatel přejde na jinou stránku, tak se děje úplně to stejné a stránka se celá překreslí. Znovu se vykresluje i například navigace, která může být na obou stránkách stejná. Stejné chování se děje i při odeslání formuláře. Uživatel ho odešle a čeká na překreslení stránky. Když se ale nepodívá v prohlížeči, že se mu nová stránka načítá, tak omylem může odeslat formulář kliknutím znovu.

V této době JavaScriptových frameworků existuje ještě metoda SPA. Ta HTML vykreslí na straně klienta a poté překresluje jen ty části, které se musí změnit. Uživatel tak nevidí „problikávání“ nové stránky. Tento způsob se velmi se přibližuje funkcionalitě desktopových aplikací. Při odeslání formuláře komunikuje SPA jako mobilní, nebo desktopová aplikace. Odesílá na nějaké API požadavek a čeká na odpověď, takže může jednoduše klientovi zobrazit načítání.

Pro prototyp této bakalářské práce je vhodné, aby byla aplikace interaktivní a nepotřebovala aktualizace stránky, takže prototyp bude vyvíjen jako Single Page Application.

5.14.4 React

React je JavaScriptový framework pro tvorbu uživatelského rozhraní.

Tento framework implementuje pouze View (V) vrstvu z modelu MVC (5.11). Unixový přístup Reactu se pro tento projekt velmi hodí, protože je možné zvolit vyhovující knihovny, které implementují ostatní vrstvy MVC.

V Reactu lze využít jakoukoliv architekturu pro manipulaci se stavem aplikace, místo ohýbání „best practices“ nějakého frameworku. Je také využívaný v produkci hned několika velkými společnostmi, má velkou komunitu vývojářů a vynikající dokumentaci.

React využívá koncept virtuálního DOMu. Vytváří virtuální reprezentaci DOMu v paměti a tuto reprezentaci synchronizuje s reálným DOMem.

Tento framework nám umožňuje přesunout logiku z globálního HTML do komponent. Komponenta je v Reactu funkce, která nakonec vrátí HTML. Do této komponenty lze předávat data, takže je možné je znovu využívat na více místech. Příklad komponenty může být tlačítko, které přijímá, zda má být velké nebo malé. Tím je nejen udržená konzistence v tlačítkách na celém projektu, ale hlavně změna tohoto tlačítka nám upraví všechna tlačítka v celé aplikaci.

Má dobré nástroje pro tvorbu Single Page Application. Dokáže rychle překreslovat jednotlivé části webové stránky.

5.14.5 Next.js

Při vytváření SPA (5.14.3) je v čistém Reactu implementačně náročným problémem přesměrování na jinou stránku.

Jedním z řešení je poslat klientovi při načtení první stránky úplně všechny stránky. První načtení jakékoliv stránky je tedy velmi dlouhé, ale poté již není potřeba donačítat prostředky při přesměrování. Tento přístup není výhodný kvůli prvotnímu dlouhému načítání.

Framework Next.js řeší donačítání potřebných komponent a stylů pro překreslení stránky (Automatic Code Splitting). Načtení stránky je tedy velmi rychlé, protože vždy stahuje jen to, co je třeba pro vykreslení.

Tento framework byl vybrán, protože je dobře konfigurovatelný a lze využívat se všemi webpack (5.14.8) pluginy. Podporuje také Server Side Rendering (SSR). SSR je funkcionalita frameworku, která předkreslí část HTML již na serveru, aby klient nebyl tolik zatížený a pro lepší indexovatelnost vyhledávači.

5.14.6 Redux

Redux je předvídatelný kontejner pro manipulaci se stavem aplikace v jazyce JavaScript.

Tato architektura pro stav aplikace byla vybrána, protože je velmi jednoduchá a má velmi blízko k funkcionálnímu programování. Architektura má implementaci pro React, která dovoluje přístup ke stavu z komponent.

Při využití virtuálního DOMu musí React dělat porovnání stavů, aby zjistil jestli má aplikaci překreslit. Pro tento proces je nejlepší, když je stav immutable (neměnitelný), což je přesně jeden z principů Reduxu.

Redux definuje základní tři principy pro správu stavu.

1. Aplikace má pouze jediný zdroj stavu, který je reprezentovaný jako strom.

2. Stav je pouze ke čtení. Nejde tedy stavem manipulovat jinak, než vytvořit nový, kterým se starý stav nahradí.

3. Změny stavu se dějí pomocí „čistých“ (pure) funkcí. Čisté funkce jsou takové funkce, které mají pro stejné argumenty vždy stejnou návratovou hodnotu. Její zavolání nemá žádné vedlejší efekty, jako změnu globálních proměnných, nebo argumentů funkce.

Změna stavu aplikace v Reduxu začíná u **action** (akce). Redux nabízí **dispatcher**, který vezme akci a odešle ji. Akce jde přes **middleware** do **reduceru**. Reducer je implementován jako čistá funkce, která vrátí nový stav aplikace. Middleware je možný mezičlánek, který dovoluje zareagovat na jakoukoliv vyvolanou akci. Nemusí ale vůbec existovat. [31]

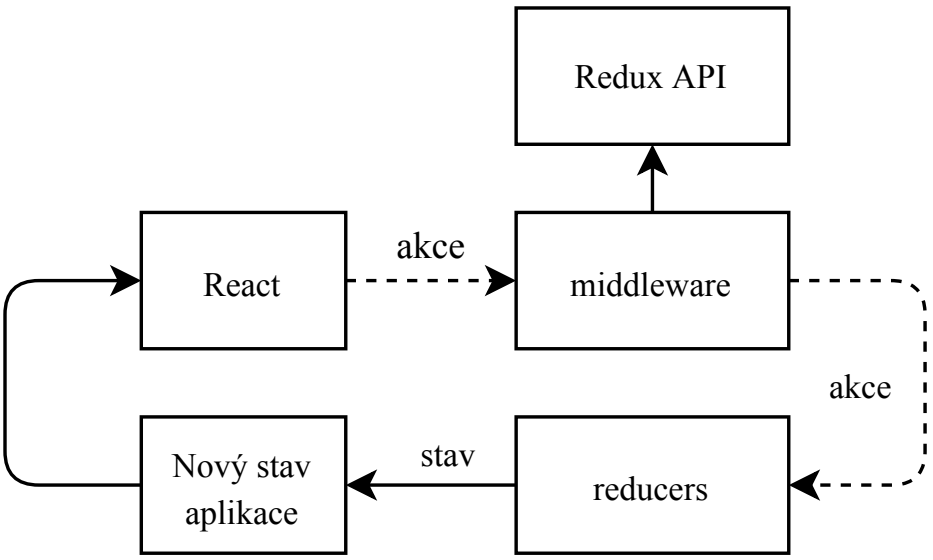
Příklad využití Reduxu bude ukázán na jednoduchém úkolníčku, který bude jen zapisovat a odstraňovat úkoly. Bude vytvořena akce ADD_TODO (přidat úkol do úkolníčku), která bude mít ve svém těle vypsaný úkol. Tato akce jde do reduceru, který vytvoří nový stav se seznamem úkolů. Pro odstranění úkolu z úkolníčku bude vytvořena akce REMOVE_TODO, v jejímž těle bude jaký úkol odstranit. Na tuto akci zareaguje znovu reducer tím, že vytvoří nový stav bez vybraného úkolu.

5.14.7 Redux-saga

Redux-saga je middleware pro Redux. Chová se jako nezávislý proces, který podle akcí reaguje programátorem definovaným způsobem. Má nástroje pro práci s asynchronním procesem, díky kterým syntaxe vypadá podobně jako v synchronních jazycích.

Redux-saga je knihovna, která má za cíl u aplikačních side effects (asynchronní události jako získání dat) zjednodušit správu, zvýšit účinnost jejich vykonávání, zjednodušit testování a usnadnit zacházení s chybami.

Příkladem využití Redux-saga může být volání na API. Dispatcher zavolá akci GET_DATA, která jde do reduceru a ten změní stav aplikace na LOADING (aplikace čeká na data). Middleware využívající Redux-saga zachytí akci a vytvoří požadavek pro získání dat. Po získání dat middleware odešle přes Redux API akci DATA_LOADED, která obsahuje data z odpovědi na požadavek. Akci zpracuje reducer, který změní stav aplikace. [32]



Obrázek 5.3: Průchod změny stavu aplikace v Reduxu a Reactu

5.14.8 Webpack

Webový vývoj je proslulý svým porušováním velkého množství „best practices“ v programování. Webpack je jeden z důvodů, proč tomu tak v moderním JavaScript kódu nemusí být.

S příchodem Node.js (5.8) přibyl do JavaScriptu koncept modulů. Moduly (v některých jazycích balíčky nebo knihovny) reprezentují nějaké funkční celky. Například modul Matematika, který obsahuje konstanty a funkce pro základní matematické operace.

Prohlížeč ale JavaScriptový kód, který má v sobě import modulů, neumí interpretovat. Tento problém řeší Webpack. Získá všechny moduly a přidá je do kontextu jednoho nebo více souborů, který prohlížeč umí interpretovat. Konec globálních proměnných napojených na „window“ objekt prohlížeče.

5.14.9 LinguiJS

Prototyp této bakalářské práce by měl být lokalizován do více jazyků.

LinguiJS je knihovna pro správu jazyků a překladů. Dokáže generovat překladové soubory hned v několika formátech, na který jsou překladatelé zvyklí a jednoduše lokalizovat frontend prohlížeče. Místo daného překladu se umístí do kódu unikátní identifikátor, který je poté nahrazen překladem.á

5.15 Technologie pro backend

Backend aplikace musí implementovat GraphQL API. Implementaci není potřeba vyvíjet vlastní, protože pro Node.js již existují open source knihovny pro

tento účel. Při komunikaci budou data vždy typované, takže by bylo vhodné najít knihovnu implementující GraphQL API s typovými TypeScript definicemi.

Díky Node.js může běžet JavaScript i na serveru a jeho asynchronnost se velmi hodí pro implementaci webového serveru.

V prototypu bakalářské práce je třeba ukládat data o místnostech, uživatelích a dalších údajích. Bude tedy třeba najít databázi s potřebnými vlastnostmi a knihovnu pro komunikaci s takovou databází.

5.15.1 Výběr databáze

V aplikaci je většina dat provázaných, mají tedy mezi sebou relace. U písničky je například relace s uživatelem, který ji přidal do fronty a s místností do které byla přidána. Využití nerelační databáze je v tomto případě nevhodné, protože je třeba často měnit pouze některá z těchto dat.

Nejvíce využívané relační databáze používají dotazovací jazyk SQL. Je třeba najít SQL databázi, která bude nabízet nějakou formu interakce v reálném čase (notifikace o změně dat). Je také potřeba, aby byla databáze open source a měla aktivní vývojářskou komunitu, protože pro účely tohoto projektu nejsou vyhrazeny žádné finanční prostředky.

Pro prototyp byla zvolena databáze PostgreSQL. Tato databáze je open source, má aktivní komunitu vývojářů a je vyvíjená již desítky let. Knihovna pro komunikaci s PostgreSQL v Node.js existuje.

5.15.2 Object Relational Mapping

Objektově relační mapování je technika pro dotazování a manipulování s daty v databázi. Programátora odstiňuje od syrového přístupu k databázi, který využívá pouze SQL dotazy. Může využívat svůj oblíbený programovací jazyk a znát pouze základy nějakého dotazovacího jazyka a ORM se postará o překlad dotazu a získání dat.

Pro ORM programátor vytvoří objekty (tzv. entity), do kterých ORM předá data z databáze. Programátor pracuje s vlastními objekty, což velmi usnadní práci s databází. Protože objekty vytváří programátor, může na nich definovat i vlastní metody, nebo jinou funkcionalitu. To se velmi hodí pro tuto aplikaci, protože lze entity využít i jako GraphQL typy.

5.15.3 TypeORM

TypeORM je implementace ORM pro Node.js. Tato implementace vyniká díky možnosti statického typování entit databáze. Inspirovala se projekty jako je Hibernate (Java ORM), nebo Doctrine (PHP ORM).

Typovaný ORM byl vybrán kvůli předání dat z databáze přímo GraphQL API. Entity jsou otypovány pomocí databázových i GraphQL anotací. Poté

pokud je třeba získat data z databáze, stačí využít ORM a ten vrátí data namapovaná do vytvořených entit, které byly nadefinovány.

Při vrácení této entity z funkce jí GraphQL implementace rovnou přetransformuje v potřebný JSON. Díky tomuto způsobu předávání dat je čitelnější a přehlednější kód, který neobsahuje žádné transformace z jedné struktury do jiné.

5.15.4 SOLID

Pro psaní čistého OOP kódu je třeba držet standardy pro návrh tříd. Psát čistý kód je velmi důležité pro jeho následné ladění a řešení závislostí mezi jednotlivými komponentami. Tento projekt se tedy drží pěti principů pro návrh tříd SOLID.

SRP (Single Responsibility Principle) – každá třída by měla mít pouze jednu odpovědnost.

OCP (The Open Closed Principle) – třídy by měly být rozšiřitelné bez nutnosti změny jejich implementace.

LSP (The Liskov Substitution Principle) – všechny podtřídy musí mít zaměnitelné s jejich rodičovskou třídou.

ISP (The Interface Segregation Principle) – při vytváření rozhraní je nutné, aby bylo specifické pro daný problém.

DIP (The Dependency Inversion Principle) – závislosti musí být vždy nad abstraktním, ne na konkrétním. [33]

5.15.5 Inversion of Control, Dependency Injection

Při psaní objektově orientovaného kódu se často stane, že je potřeba využít v třídě instanci jiné třídy (např. třída pro získávání písniček chce využít třídu pro přístup k databázi). Pokud se v této třídě vytvoří instance další třídy, tak by na sobě byly závislé a instance potřebné třídy nenahraditelná jinou. Proto využijeme návrhový vzor IoC (Inversion of Control).

V třídě nadefinujeme způsob (metodu, interface), pomocí kterého programátor předá instanci potřebné třídy. Díky tomuto přístupu může mít i tato třída další závislosti, o kterých třída která ji využívá nemusí vědět.

Dependency Injection (DI) je poté užší implementace IoC. V případě této bakalářské práce se bude návrhový vzor využívat tak, že instance tříd se budou předávat přes konstruktor. Třída se závislostmi by tak v konstruktoru specifikace tříd, ke kterým chce mít přístup.

To je výhodné z hlediska testování, protože lze takové třídě předat jako instanci falešnou testovací třídu, která má stejné rozhraní, ale jinou implementaci. Třída tedy pro volání bude mít predikovatelný výsledek, čímž je vždy jasné, jaký by měl být její výstup.

DI kontejner je datová struktura instancí tříd udržovaný nad aplikací. V jeho konfiguračním souboru lze nadefinovat jaké třídy jsou v kontejneru po-

třeba. Další služba podle typů závislostí v konstruktoru předá dané instance automaticky, bez nutnosti je vždy vytvářet programátorem. Kontejner tedy řeší nejen jejich závislosti, ale i nepotřebné znovuvytváření instancí tříd (více tříd pro přístup k jedné databázi).

5.15.6 Nest.js

Progresivní Node.js framework pro vytváření efektivních, spolehlivých a škálovatelných serverových aplikací. (překlad autora [34])

Tento framework má prvotřídní podporu TypeScriptu a psaní otypovaného kódu je s ním velmi intuitivní. Hlavním přínosem je jednotná architektura, kterou díky frameworku může programátor využívat.

Nest.js je velmi modulární, takže neuzavře programátora pouze do ekosystému frameworku. Drží se pěti principů pro pochopitelný a udržitelný kód SOLID (5.15.4). Pro udržitelnou architekturu nabízí také kontejner pro Dependency Injection (5.15.5).

Framework nabízí modul pro implementaci GraphQL API. Tento modul využívá Apollo GraphQL platformu. Tato platforma nabízí své open source moduly pro využití zdarma a zároveň k nim dává enterprise podporu. [35] Modul, který framework nabízí, je pouze wrapper nad touto platformou. Konfigurace je ale prováděna přes framework a je udržena konzistentní architektura. [34]

5.16 Výhody kombinace technologií pro frontend a backend

Využít TypeScript, nebo JavaScript pro frontend je velmi časté. Využívat ho i pro backend není již tak časté. Je na místě uvést několik výhod pro toto rozhodnutí.

Jedna z výhod využití jednoho programovacího jazyka je, že není potřeba přepínat kontext, pokud je třeba implementovat backendovou, nebo frontendovou část projektu. Různé nuance mezi syntaxí jednotlivých jazyků jsou častým důvodem pro nepříjemné problémy. Tady se stačí naučit operátorovou logiku jen jednou a není třeba ji měnit v závislosti, na jaké části projektu programátor pracuje.

Další z výhod je při přijímání externích vývojářů do projektu a jejich následné komunikaci. Jistě, že se frontend vývojář specializuje na úplně jinou část TypeScriptového světa, než ten backendový. Jejich komunikace je usnadněná skutečností, že využívají stejné datové struktury a syntaxi.

Hlavní výhodou je určitě sdílená logika. Ostatní výhody sice zefektivní vývojový proces, ale nijak nezmenší počet řádků, které musí programátor napsat. Lze implementovat logiku (např. validování vstupu, získávání odkazů) a poté tuto logiku využít jak na frontendu, tak na backendu aplikace. Pro

tuto logiku je napsán pouze jeden test a při její změně je uskutečněna pouze na jednom místě. Tuto logiku je možné vytvořit jako modul, který se využije v obou částech aplikace.

5.16.1 Monorepo

Ve verzovacích systémech je označována technika monorepo, jako verzování více projektů v jednom repozitáři.

Takový přístup se pro tento projekt velmi hodí. I když jsou vyvíjeny dvě nezávislé aplikace, komunikují spolu skrz nějaké API. Pokud na backendu změníme přístup nebo návratový typ nějakého endpointu (dotazu, mutace), musíme tuto změnu reflektovat i na frontendu.

Při verzování je často potřeba najít starší verzi repozitáře, a kdyby byly obě části verzovány zvlášť, muselo by se přejít na stejné verze obou částí. Při monorepo přístupu se vždy přejde s frontendem i backendem na stejnou verzi. Jednotlivé změny jsou také lépe vidět, protože jsou ve verzi zaznamenány rozdíly obou částí aplikace najednou.

Kapitola 6

Realizace

V minulé kapitole byly představeny technologie, které se budou v prototypu využívat. V této kapitole bude popsáno jak byly tyto technologie použity v praxi.

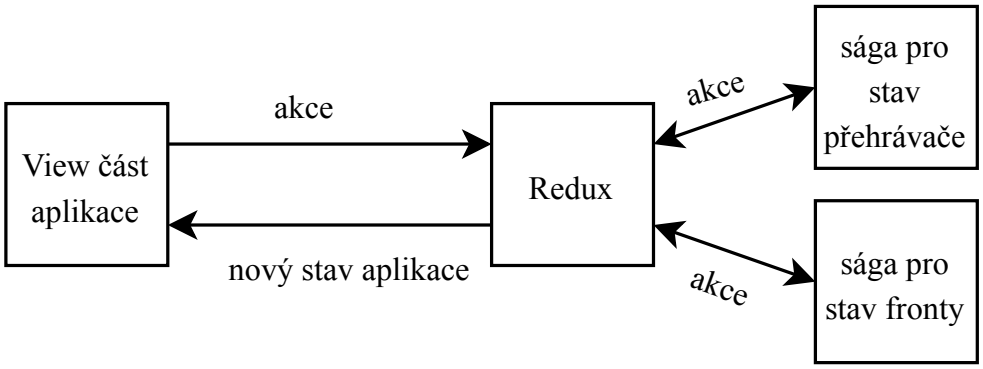
Krom krátkých ukázek z kódu je popsán i diagram pro výměnu OAuth tokenu a snímky obrazovky z prototypu aplikace.

6.1 Implementace frontendu

JavaScript je známý pro jeho asynchronnost. Pomocí knihovny redux-saga můžeme využívat asynchronní frontu JavaScriptu jako v synchronních paralelních jazycích. Díky tomu je kód čitelnější.

Architektura je rozdělena do dvou částí. První část má na starost vykreslení aplikace (View část aplikace) a druhá se stará se o správu stavu (Redux část aplikace).

V Reduxové části aplikace jsou vytvořeny dvě ságy. Jedna se stará o události týkající se Spotify (např. pauza písničky, přepnutí písničky a jiné). Druhá



Obrázek 6.1: Správa stavu aplikace

je pro stav místnosti, která se stará o synchronizaci fronty a právě hrajících písniček (6.1).

6.1.1 View část aplikace

Pokud uživatel otevře prohlížeč a vyžádá si hlavní stránku prototypu, je vyslán požadavek na Next.js server. Ten předvykreslí některé části na serveru a pošle uživateli všechen potřebný JavaScript pro dovykreslení stránky.

Všechny prvky na stránce jsou React komponenty. Z těchto komponent je pak volána logika pro získání potřebných dat z GraphQL API.

6.1.2 Redux a ságy

V pozadí aplikace se dějí změny jejího stavu, které se pak promítají ve View části. Zde je ukázka, jak funguje změna stavu pro ságu sloužící k aktualizaci fronty.

Nejdříve je třeba nadefinovat dotaz na GraphQL server. Má být navázáno WebSocket spojení, takže je využita GraphQL subscription. V této subscription specifikujeme, jaké klíče ze struktury má server vrátit. Jako parametr serveru posíláme identifikátor místnosti (roomIdentifier).

Kód 6.1: GraphQL subscription dotaz pro poslouchání změny fronty

```
subscription queue($roomIdentifier: String!) {
  queue(roomIdentifier: $roomIdentifier) {
    id
    spotifyId
    name
    popularity
    albumArt
    artists {
      id
      name
    }
  }
}
```

V kódu je dotaz z ukázky uložený v konstantě SUBSCRIBE_QUEUE. Kód je třeba číst zdola nahoru. Pokud dojde k akci CHANGE_ROOM (změna místnosti), je zavolána funkce checkRoomChange. Ta získá unikátní identifikátor místnosti z těla akce a vytvoří nový „proces“ na kontrolu změny fronty.

V tomto procesu je vytvořen listener, který poslouchá, zda nepřišla ze serveru nové data o frontě. Když přijdou, zavolá funkci handleQueueChange, která vezme předaná data a vytvoří novou akci pro změnu stavu aplikace.

Kód 6.2: Redux sága pro poslouchání změny fronty

```
function* handleQueueChange(queue: QueueSubscriptionResult) {
  if (queue?.data?.queue) {
    yield put(changeQueue(queue.data.queue));
  }
}

function* queueSubscription(
  roomIdentifier: string,
  client: ApolloClient<NormalizedCacheObject>,
) {
  const queueChanged = yield call(createSubscription, () =>
    client.subscribe<QueueSubscription>({
      query: SUBSCRIBE_QUEUE,
      variables: {
        roomIdentifier,
      },
    }),
  );
  yield takeEvery(queueChanged, handleQueueChange);
}

function* checkRoomChange(action: RoomAction) {
  const roomIdentifier = action.room.identifier;
  const client = getApolloClient();
  yield fork(queueSubscription, roomIdentifier, client);
}

export function* watchRoomChange() {
  yield takeLatest(CHANGE_ROOM, checkRoomChange);
}
```

6.2 Implementace backendu

Backend implementuje GraphQL API využívané frontendem. Pro tyto potřeby byly vytvořeny entity, které jsou využívány nejen ORM k namapování dat z databáze, ale často i předávány přímo v návratové hodnotě do implementace GraphQL. GraphQL modul poté tyto entity podle anotací zparsuje, vytvoří dokumentaci a vrací je v dané struktuře.

Zde je ukázka, jak backend reaguje na subscription od klienta. Je definován typ, který řekne klientovi, jaké klíče může získat dotazem.

Kód 6.3: GraphQL objekt se strukturou

```
@ObjectType()
export class RatedTrack {
  @Field()
  id: number;

  @Field()
  name: string;

  @Field(returns => [Artist])
  artists: Artist[];

  @Field()
  albumArt: string;

  @Field()
  spotifyId: string;

  @Field()
  popularity: number;
}
```

Poté je definována třída, která se chová jako Resolver. Resolver je třída, která odpovídá na GraphQL dotazy. Pomocí Dependency Injection je v konstruktoru předána služba, která se v aplikaci stará o subscriptions.

Anotací @Subscription nad funkcí queue je specifikováno, že se jedná o GraphQL subscription a že vrací typ pole, v němž se nachází objekty typu RatedTrack. V argumentech musí klient předat identifikátor místnosti. Validace se děje automaticky jen z typů v anotacích.

Poté se zavolá subscription service, aby vrátila WebSocket spojení.

Kód 6.4: GraphQL Resolver, který poskytuje subscription

```
@Resolver()
export class QueueResolver {
  constructor(
    private readonly subscriptionService: SubscriptionService,
  ) {}

  @Subscription(returns => [RatedTrack])
  queue(@Args('roomIdentifier') roomIdentifier: string) {
    return this.subscriptionService
        .subscribeToQueue(roomIdentifier);
  }
}
```

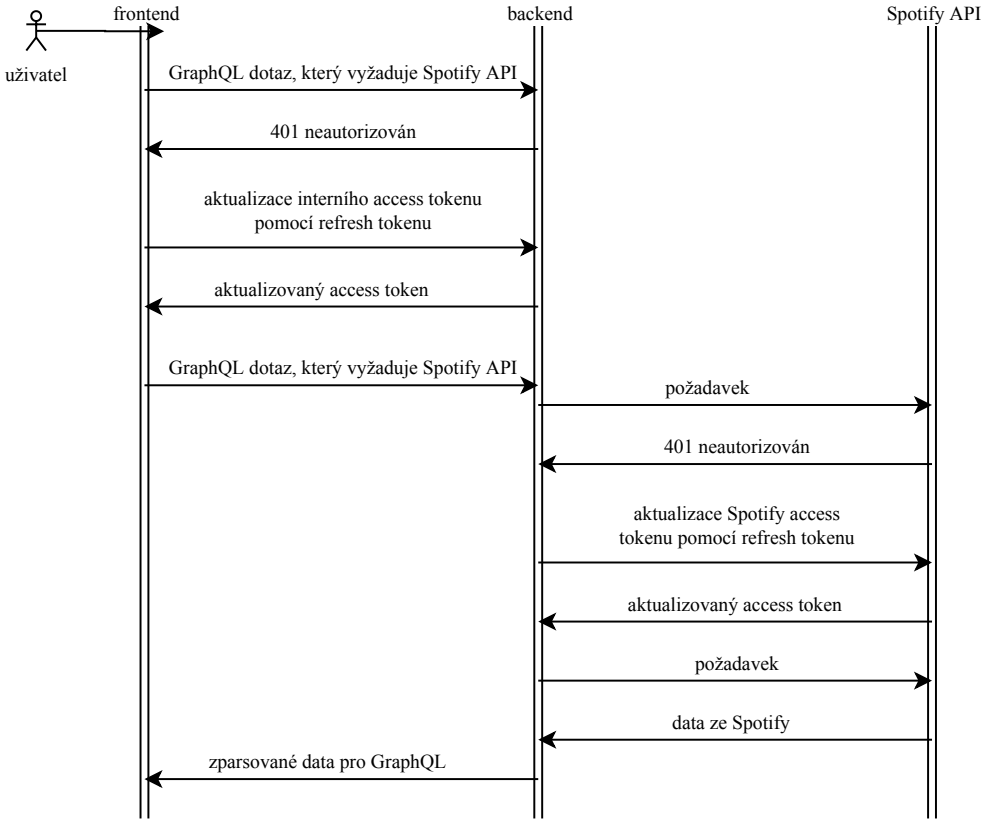
6.2.1 Správa tokenů

Vytvoření nové místnosti se děje přes autorizaci skrz Spotify. Klient si musí od backendu vyžádat autentifikační token (access token) podle OAuth 2.0 protokolu. Backend při autorizaci Spotify účtem vytvoří i interní účet. V tomto účtu je interní access token a Spotify access token.

Spotify access token je třeba ukládat do databáze. Když si totiž neautorizovaný uživatel chce najít písničku pro přidání do fronty, je třeba použít token pro požadavek na Spotify API. Hostitel ale zároveň musí mít také lokálně Spotify access token, protože pomocí něho přehrává písničky v prohlížeči.

Podle standardu OAuth 2.0 je autorizační token omezený dobou platnosti. Doba platnosti Spotify tokenu je jedna hodina. I když interní token má taky dobu platnosti jednu hodinu, je třeba sledovat expiraci obou tokenů.

Byla tedy vytvořena single source of truth (data jsou manipulována pouze na jednom místě) architektura, která udržuje vždy platný Spotify access token na backendu. Když hostiteli vyprší Spotify access token, zavolá backend, aby mu získal nový, a autorizuje se pomocí interního access tokenu.



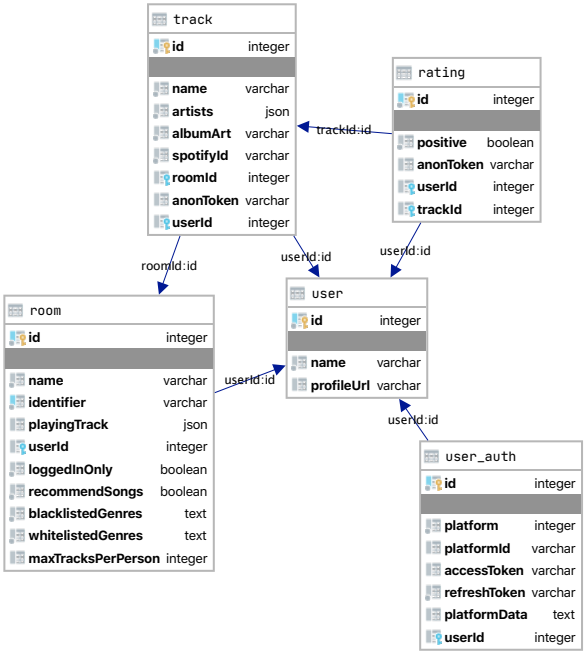
Obrázek 6.2: Ukázka komunikace při vypršení obou tokenů

6.2.2 Návrh databáze

Databáze aplikace je kompaktní. Pro evidenci místností je tabulka **room**. Tato tabulka obsahuje všechno nastavení místnosti, identifikátor místnosti a uživatele, který místnost vlastní.

Uživatel je reprezentován v tabulce **user** a může být zaregistrován a přihlášen přes více OAuth 2.0 služeb, které jsou evidované tabulkou **user_auth**. V této tabulce se nachází access token a unikátní identifikátor uživatele u dané platformy.

Poté jsou již písničky v tabulce **track**. U písničky je evidována místnost, v jejíž frontě se nachází a uživatel, který jí do fronty přidal. U dané písničky jsou ještě hodnocení v tabulce **rating**, které evidují kladnou nebo zápornou hodnotu písničky ve frontě.



Obrázek 6.3: Návrh databáze

6.3 Problémy při vývoji

Díky synchronizaci dat v reálném čase nastalo na frontendu mnoho problémů. Jeden z nejzávažnějších problémů se týkal mobilních zařízení. Když se mobil uspí, nebo uživatel otevře jinou aplikaci než prohlížeč, přeruší se spojení WebSocket. Když aplikaci prohlížeče ale uživatel znovu otevře, je stránka uložená v paměti a neprovede se refresh (aktualizace stránky). Tím jsou data na zařízení neaktuální.

Naštěstí většina prohlížečů již používá Page Visibility API, které dovoluje zjistit stav viditelnosti stránky na zařízení. V aplikaci je tedy zaregistrován listener pro opětovné otevření stránky. Pokud uživatel znovu otevře aplikaci, je opětovně připojen do WebSocketu a vyžádá si od serveru nová data.

Prototyp této bakalářské práce se připojuje k Spotify a přidává se jako zařízení pro přehrávání hudby Spotify Connect. Bylo tedy nutné synchronizovat ovládání z Spotify aplikací a prohlížeče. Když uživatel pozastaví písničku v Spotify aplikaci, musí se pozastavit i v tomto prototypu.

Aby mohli neautorizovaní uživatelé hodnotit a přidávat písničky do fronty, je třeba je nějak identifikovat. Tato autorizace anonymního uživatele se podařila vyřešit velmi elegantně díky Nest.js. Umí totiž zaregistrovat tzv. interceptor. Interceptor se dá zaregistrovat u některých GraphQL resolverů a umí spustit kód při přijmutí požadavku a před odesláním odpovědi. Lze tedy při přijmutí požadavku zkontrolovat, zda má uživatel přidělený anonymní token, a případně mu ho vytvořit a poté poslat s výsledkem dotazu.

6.4 Testování

Při vývoji se vždy vyskytnou nějaké chyby v implementaci. Pokud se funkčnosti projektu skládají z malých funkčních celků (služeb), lze napsat logiku pro testovaní těchto celků. V takovém kódu je specifikováno, jak by měl vypadat vstup a jaký by měl mít výsledek.

Těžším případem k otestování je služba, která potřebuje pro vykonání kódu nějaké závislosti (další služby). Pokud chceme testovat službu se závislostmi, musíme buď inicializovat všechny její závislosti, nebo je nahradit testovacími (falešnými).

Na projektu se proto používají dva typy testů.

Prvním typem jsou unit testy. Takové testy mají nahrazené závislosti za testovací a nikdy nepřistupují k reálným prostředkům (např. k databázi). Napsáním takových testů je v budoucnu jednodušší odchytit všechny chyby v nově napsaném kódu, protože je vidět, že kvůli změně kód neprošel testem. Při psaní kódu programátor dělá co nejmenší celky, aby se pro ně jednodušeji psaly unit testy. V takových testech je také vidět, co je přesně vstupem a výstupem kódu, takže se vyplatí před čtením implementace nejdříve otevřít test.

Kód 6.5: Jednoduchý test pro kontrolu žánrů

```
import { isGenreListed } from '@api/models/genres';

describe('genresHelper', () => {
  it('should check blacklisted genres', () => {
    const blacklisted = ['rap', 'metal'];
    expect(isGenreListed(blacklisted, ['folk', 'pop']))
        .toBeFalsy();
    expect(isGenreListed(blacklisted, ['deep rap', 'pop']))
        .toBeTruthy();
    expect(isGenreListed(blacklisted, [])).toBeFalsy();
  });
});
```

Druhý typ testů jsou funkcionální testy. Hodně služeb často interaguje s databází a i tento kód je třeba mít otestovaný. V funkcionálních testech vytvoříme kontejner a vyměníme modul pro interakci s databází. Místo hlavní databáze využívá jinou s předpřipravenými daty k testování. Tyto testy jsou velmi užitečné, protože odhalí i chyby při změně struktury databáze.

6.5 Nasazení

Pro účely nasazení aplikace byl vytvořen nasazovací skript v TypeScriptu. Nasazovací skript předá serveru novou verzi aplikace a poté ji spustí. Protože je využívaná architektura SSD, jsou v projektu dvě nezávislé části aplikace. Části aplikace jsou na dvou portech. Na jednom portu je frontend aplikace a na druhém backend. Lze tedy jednoduše nasadit pouze jednu část.

Server musí být správně nastaven a mít nainstalované všechny závislosti. Musí být nainstalovaný webový server Nginx, PostgreSQL databáze a Docker.

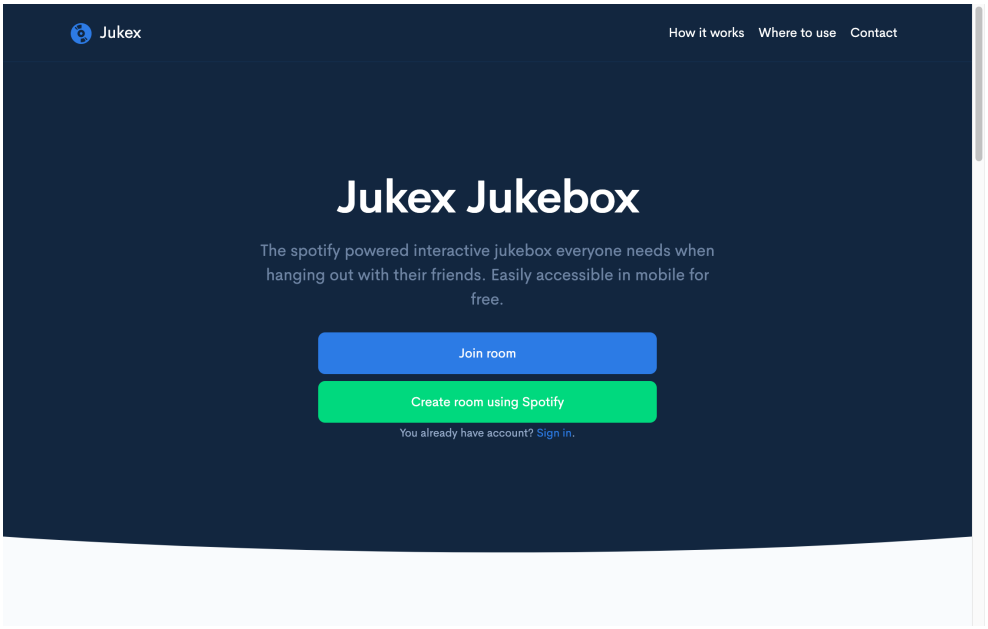
Každý požadavek na server nejdříve přijme webový server Nginx. Nginx se stará o https spojení a funguje jako proxy pro obě části aplikace. Podle url adresy přepošle požadavek na správný port, kde daná část běží.

Obě dvě části aplikace mají v kořenu projektu Dockerfile. V Dockerfile je, jaké má daná část závislosti (Node.js) a jaké příkazy se mají provést při build fázi. V obou se načte konfigurace, nainstalují Node.js knihovny pomocí npm a vytvoří se produkční image částí aplikace.

Produkční image aplikace je pomocí bzip2 zkomprimována a odeslána na server, kde je načtena Dockerem. Pokud se celý tento proces povede, je na serveru zavolán příkaz pro vypnutí staré image a zapnutí nové. Nasazovat je tedy bezpečné, protože při chybě se image nevymění.

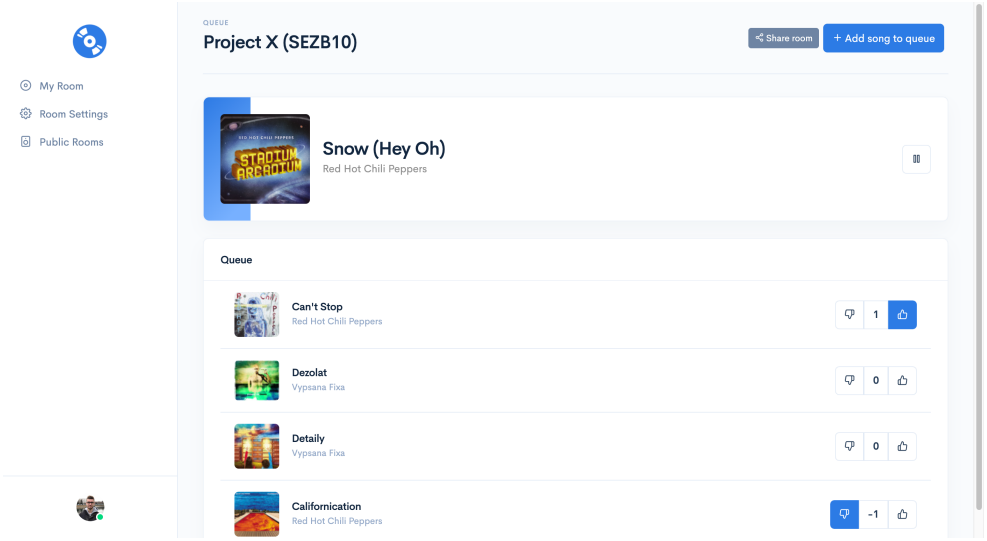
6.6 Prototyp aplikace

Při načtení stránky je uživateli popsáno, jak aplikace funguje a jaké kroky má udělat k vytvoření místnosti.



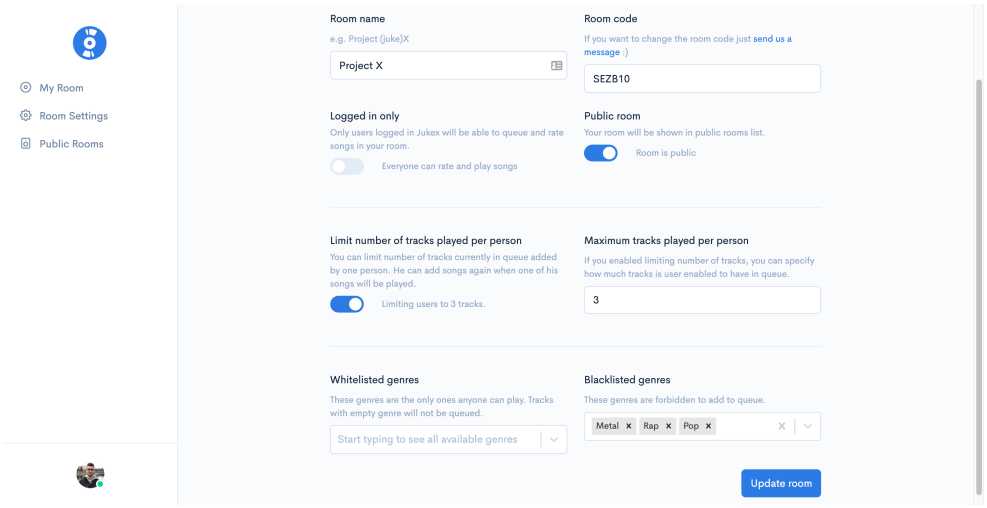
Obrázek 6.4: Ukázka hlavní stránky aplikace

Na stránce samotné místnosti je vidět právě přehrávaná písnička a fronta následujících písniček. Tato fronta se mění v závislosti na hodnocení jednotlivých písniček. V hlavičce stránky je vidět tlačítko pro přidání písničky do fronty.



Obrázek 6.5: Ukázka místnosti

V nastavení místnosti je poté možné omezit počet písniček na uživatele, nebo žánry písniček.



Obrázek 6.6: Ukázka nastavení místnosti

Závěr

Cílem této bakalářského práce bylo navrhnout a vytvořit prototyp interaktivního jukeboxu. Tento jukebox měl být spustitelný v internetovém prohlížeči a všechny jeho akce měly být synchronizovány v reálném čase.

Takový jukebox se v této bakalářské práci podařilo navrhnout. Tento návrh byl implementován a prototyp je nasazen pro testování na webové adrese `https://jukex.eu`.

Prototyp jukeboxu má implementovaný celý proces přidávání písniček do fronty, jejich přehrávání pomocí Spotify Web SDK a synchronizaci v reálném čase mezi všemi zařízeními připojenými k místnosti. Má také komplexní nastavení, ve kterém může hostitel specifikovat žánr písniček. Lze specifikovat, jaké žánry písniček můžou být místnosti přehrávány, ale i jaké žánry mají být zakázané. Může také omezit počet písniček přidaných na uživatele. Prototyp je přeložený do dvou jazyků – čeština a angličtina.

Výběr technologií a správný návrh architektury backendu i frontendu byl jeden z nejtěžších a nejvíce časové náročných částí této bakalářské práce. Investovaný čas do této fáze se určitě vyplatil, protože se aplikace lépe ladí, má predikovatelnou změnu stavů a striktně typovaný kód. S vybranými technologiemi se pracovalo dobře a věřím, že se produkt bude rozšiřovat svou funkčností i do budoucna.

I přes implementování všech nutně potřebných funkcionalit zůstalo místo pro zlepšení. Do budoucna by aplikace měla podporovat i největšího konkurenta Spotify – Apple Music. Také by měla implementovat lepší funkcionalitu pro veřejné místnosti, například připojení k místnosti pouze na základě GPS pozice zařízení. Mohla by být přeložena do více než dvou jazyků.

Přijít by mohla i správa uživatelů připojených k místnosti. Zakázání přístupu k místnosti, nebo zakázání přidávat písničky na nějaký časový interval. Hostitel by poté mohl i přidávat zákazníkům nějaký počet písniček podle svého uvážení, nebo jejich útraty.

Tato bakalářská práce mi mnohé dala. Zjistil jsem, jak efektivně spravovat čas v době práce z domova. Pro její potřeby jsem objevil mnoho softwarových

vzorů pro práci s kódem. Naučil jsem se správnou implementaci GraphQL API a práci s WebSockety. Zlepšil jsem svoji znalost v manipulaci s immutable datovými strukturami a návrhem stavu aplikace. Vybrané téma bakalářské práce mě velmi bavilo a aplikaci budu rozšiřovat i do budoucna.

Literatura

[1] Litjens, J.: *Jukestar Guest: Social Jukebox on the App Store*. [online], navštíveno 3. 4. 2020. Dostupné z: https://apps.apple.com/us/app/jukestar-guest/id1232450793

[2] Pas, F.: *OutLoud Social Jukebox on the App Store*. [online], navštíveno 3. 4. 2020. Dostupné z: https://apps.apple.com/us/app/outloud-social-playlists-app/id928078942

[3] *Jubox.cz – Výroba a pronájem dotykových jukeboxů a karaoke*. [online], navštíveno 3. 4. 2020. Dostupné z: https://www.jubox.cz/

[4] Spotify: *Spotify Company Info*. [online], navštíveno 26. 3. 2020. Dostupné z: https://newsroom.spotify.com/company-info/

[5] Dredge, S.: How many users do Spotify, Apple Music and streaming services have? *Musically*, 2020, navštíveno 1. 4. 2020. Dostupné z: https://musically.com/2020/02/19/spotify-apple-how-many-users-big-music-streaming-services

[6] Česká televize: První jukebox se objevil v San Francisku. *CT 24*, listopad 2009, navštíveno 1. 4. 2020. Dostupné z: https://ct24.ceskatelevize.cz/archiv/1371780-prvni-jukebox-se-objevil-v-san-francisku

[7] Casebeer: Today in History – The First Jukebox Made It's Musical Debut. *American Blues Scene*, 2013, navštíveno 26. 3. 2020. Dostupné z: https://www.americanbluesscene.com/today-in-history-the-first-jukebox-made-its-debut/

[8] Weiss, B.: *Jukebox*. [online], navštíveno 26. 3. 2020. Dostupné z: http://www.antiqueweek.com/Article.asp?newsid=1796

[9] Jukestar: *Social Jukebox App for Spotify - Jukestar*. [online], navštíveno 3. 4. 2020. Dostupné z: https://jukestar.mobi/

[10] Mubo: *mubo - The social jukebox app for your next party or event.* [online], navštíveno 3. 4. 2020. Dostupné z: https://www.mubo-app.com/

[11] OutLoud: *OutLoud - Your Social Jukebox*. [online], navštíveno 3. 4. 2020. Dostupné z: https://outloud.dj/

[12] Thosli: *Bezplatný pronájem MP3 jukeboxů včetně karaoke a videoklipů.* [online], navštíveno 3. 4. 2020. Dostupné z: https://www.jukeboxy.info/

[13] *SoundCloud - Hear the world's sounds*. [online], navštíveno 1. 4. 2020. Dostupné z: https://soundcloud.com/

[14] *Apple Music*. [online], navštíveno 1. 4. 2020. Dostupné z: https://www.apple.com/cz/music/

[15] *Spotify*. [online], navštíveno 1. 4. 2020. Dostupné z: https://www.spotify.com/

[16] Mozilla MDN: *HTTP*. [online], navštíveno 31. 3. 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP

[17] Mozilla MDN: *WebSocket*. [online], navštíveno 31. 3. 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/WebSockets

[18] I. Fette, A. M., Google Inc.: *RFC 6455 - The WebSocket Protocol*. [online], navštíveno 8. 4. 2020. Dostupné z: https://tools.ietf.org/html/rfc6455

[19] Red Hat: *API*. [online], navštíveno 31. 3. 2020. Dostupné z: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

[20] Spotify: *Web Playback SDK*. [online], navštíveno 3. 4. 2020. Dostupné z: https://developer.spotify.com/documentation/web-playback-sdk

[21] Auth0 contributors: OAuth 2.0. *OAuth 2.0 Docs*, 2020, navštíveno 1. 4. 2020. Dostupné z: https://auth0.com/docs/protocols/oauth2

[22] Mozilla MDN: *JavaScript*. [online], navštíveno 31. 3. 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[23] Node.js contributors: *About Node.JS*. [online], navštíveno 31. 3. 2020. Dostupné z: https://nodejs.org/en/about

[24] *Module Counts*. [online], navštíveno 31. 3. 2020. Dostupné z: http://www.modulecounts.com

[25] Dunkley, W.: Split Stack Development: A Model For Modern Applications. *Medium*, 2016, navštíveno 29. 3. 2020. Dostupné z: https://medium.com/@MentallyFriendly/split-stack-development-a-model-for-modern-applications-d7b9abb47bd5

[26] olprod: Co je Docker? *Microsoft Docs*, 2018, navštíveno 1. 4. 2020. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/architecture/microservices/container-docker-introduction/docker-defined

[27] Microsoft: *TypeScript*. [online], navštíveno 31. 3. 2020. Dostupné z: https://www.typescriptlang.org/

[28] GraphQL Contributors: *GraphQL*. [online], navštíveno 28. 4. 2020. Dostupné z: https://graphql.org/

[29] Good Themes: *Dashkit*. [online], navštíveno 28. 4. 2020. Dostupné z: https://themes.getbootstrap.com/product/dashkit-admin-dashboard-template/

[30] Mozilla MDN: *Document Object Model*. [online], navštíveno 31. 3. 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[31] Redux contributors: *Redux documentation*. [online], navštíveno 3. 4. 2020. Dostupné z: https://redux.js.org/introduction/getting-started

[32] Redux-saga Contributors: *Redux-saga*. [online], navštíveno 26. 4. 2020. Dostupné z: https://redux-saga.js.org/

[33] UncleBob: The Principles of OOD. *UncleBob Articles*, 2005, navštíveno 26. 3. 2020. Dostupné z: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

[34] NestJS Contributors: *NestJS*. [online], navštíveno 28. 4. 2020. Dostupné z: https://nestjs.com/

[35] Apollo Contributors: *The Apollo GraphQL platform*. [online], navštíveno 31. 3. 2020. Dostupné z: https://www.apollographql.com/

Příloha A

Seznam použitých zkratek

DI Dependency Injection.

DOM Document Object Model.

MVC Model View Controller.

NPM Node Package Manager.

OOP Object Oriented Programming.

ORM Object Relational Mapping.

SPA Single Page Application.

SSD Split Stack Development 5.9.

SSR Server Side Rendering.

UI User interface.

Příloha B

Slovník

hostitel Člověk vlastnící zařízení, které funguje jako přehrávač písniček z fronty.

počítač Zařízení, které používá desktopový operační systém jako Windows, Linux nebo MacOS.

prototyp Prototypem je myšlena pokusná implementace této bakalářské práce.

uživatel Člověk vlastnící zařízení, které může přidávat písničky do fronty a hodnotit je.

Příloha C

Seznam použitých zkratek

DI Dependency Injection.

DOM Document Object Model.

MVC Model View Controller.

NPM Node Package Manager.

OOP Object Oriented Programming.

ORM Object Relational Mapping.

SPA Single Page Application.

SSD Split Stack Development 5.9.

SSR Server Side Rendering.

UI User interface.

Příloha D

Obsah přiložené SD karty

/
- `api` ..... Implementace backendu aplikace
- `app` ..... Implementace frontendu aplikace
- `core` ..... Sdílená knihovna pro backend i frontend
- `postgres` ..... Inicializační skript pro PostgreSQL
- `deploy` ..... Skript pro nasazení aplikace na server
- `thesis` ..... Zdrojový LaTeX kód bakalářské práce
- `docker-compose.yml` ..... Konfigurace Dockeru pro spuštění aplikace
- `Thesis.pdf` ..... Bakalářská práce v PDF
- `README.md` ..... Dokumentace pro projekt