Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Integrace streamovacích platforem ve webové aplikaci

Merger

Bakalářská práce

Autor: Vítězslav Zotov
Studijní obor: ai3-p

Vedoucí práce: Mgr. Daniela Ponce, Ph. D.

Hradec Králové Srpen 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 15.8.2022 Jméno a Příjmení

Poděkování:

Děkuji vedoucímu Mgr. Daniela Ponce, Ph. D. za metodické vedení práce. Mé poděkování patří též Lukáši Tesařovi za trpělivost a pomoc při gramatické kontrole práce.

Anotace:

Tato bakalářská práce se zabývá problematikou integrace dvou streamovacích platforem YouTube a Spotify do jedné aplikace. V práci jsou popsané nástroje a prostředky, které byly vybrány a použity pro vytvoření implementace dané aplikace. Následně jsou popsány metody implementace a její popis.

Annotation:

This bachelor's thesis focuses on problematics of integrating two streaming platforms Spotify and Youtube into a single application. Thesis also describes the tools and resources which were chosen and used for implementation of the given application. Subsequently the work deals with methods of implementation and its description.

Obsah

1 Úvod

Globální studie provedena společností YouGov zkoumala čtyři státy v rámci nejvíce používaných streamovacích platforem. V rámci této studie bylo zjištěno, že mezi dospělými jsou nejvíce používány streamovací platformy Spotify a YouTube. Výjimkou je Indie, kde se Spotify nachází na posledním místě a jeho náhradou je Amazon Prime Musci, které je na místě druhém. [1]

Další studie provedena společností Pex poukazuje na to, které kategorie na YouTube mají největší vliv na růst platformy. [2] Výsledky ukazují, že na YouTube existuje přes 5,2 mld. videí a 1 mld. hodin obsahu. V roce 2018 bylo nahráno více než 1,3 mld. videí a z tohoto počtu videí 5% tvořila hudba. Tudíž tato kategorie se 65 milióny písniček se umísťuje na žebříčku jako poslední z hlediska procentuálního množství obsahu. Na druhou stranu generuje nejvíce zhlédnutí, tudíž je nejvíce výdělečnou kategorií na YouTube.

Podle společnosti Business of Apps Spotify obsahuje v databázi celkem 70 miliónu písniček [3], což je srovnatelné se 65 milióny písniček, které jsou na YouTube nahrávány za jeden rok. Zmíněné statistiky poukazují na to, že YouTube je ve vedení v rámci celkového počtu nahraných písniček a že všechny písničky nemusí být zahrnuty ve Spotify. Ve většině případů to je z důvodu způsobu licencování a nahrávání obsahu na Spotify, který je oproti YouTube obtížnější a placený.

2 Cíl práce

Cílem práce je integrovat dvě streamovací platformy do jedné aplikace a vyhodnotit dostupné technologie, které budou využity pro její vývoj. Aplikace umožní uživatelům přehrávat obsah jednotným způsobem nezávisle na platformě.

Následně je účelem práce demonstrovat použití prostředků a nástrojů, které aplikují potřebné funkce k provozu této aplikace a navrhnout, jakým způsobem tyto funkce implementovat (YouTube API, Spotify API a Web Playback SDK)

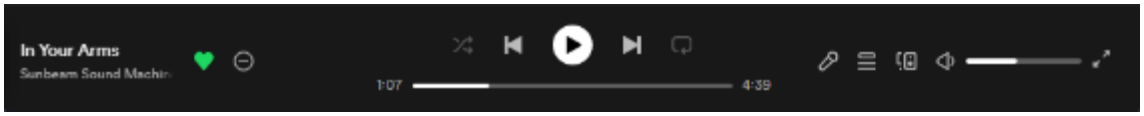
3 Metodika zpracování

V práci nejprve bude zahrnuta analýza komerčních přehrávačů, které existují na trhu. Bude se zkoumávat jejich funkcionalita a uživatelské rozhraní. Na základě této analýzy jsou stanoveny technologie nebo nástroje, které přehrávače zahrnují. Dalším krokem je stanovení funckionalit, které aplikace bude zahrnovat. Práce se poté zaměří na navržení vyvíjené aplikace a její architektury. Jakmile bude aplikace navržena, bude následovat její realizace. Tato sekce zahrnuje popis implementace, potřebných nástrojů a jejich použití v praxi.

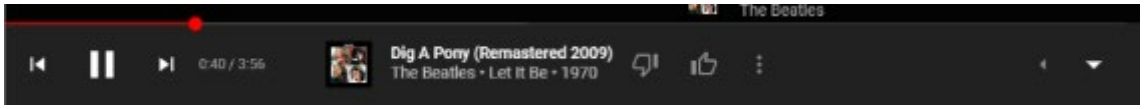
4 Analýza Komerčních přehrávačů

Předtím než bude aplikace vyvíjena, je potřeba zanalyzovat ostatní přehrávače, které existují na trhu. Mezi jejich zkoumané vlastnosti patří např. rozvržení uživatelského rozhraní a nabídka funkcionalit. Pokud si otevřeme Spotify přehrávač a YouTube vedle sebe, lze na první pohled spatřit několik podobností.

Každý přehrávač v sobě zahrnuje určitý druh ovládacího panelu (nacházející se většinou ve spodu okna), kterým lze ovládat přehrávání neboli playback skladby nebo videa. Ovládací panel obsahuje ve všech případech mnoha tlačítek, které uživateli vykoná specifickou akci jako např. pokračovat/pozastavit, přeskočit na jinou skladbu a další. Mezi dalšími ovládacími prvky patří ukazatelé průběhu a posuvníky hlasitosti. Ukazatel průběhu umožňuje uživateli pomocí myší přeskočit na určitou pozici skladby během přehrávání a posuvníkem hlasitosti lze měnit hlasitost přehrávače.

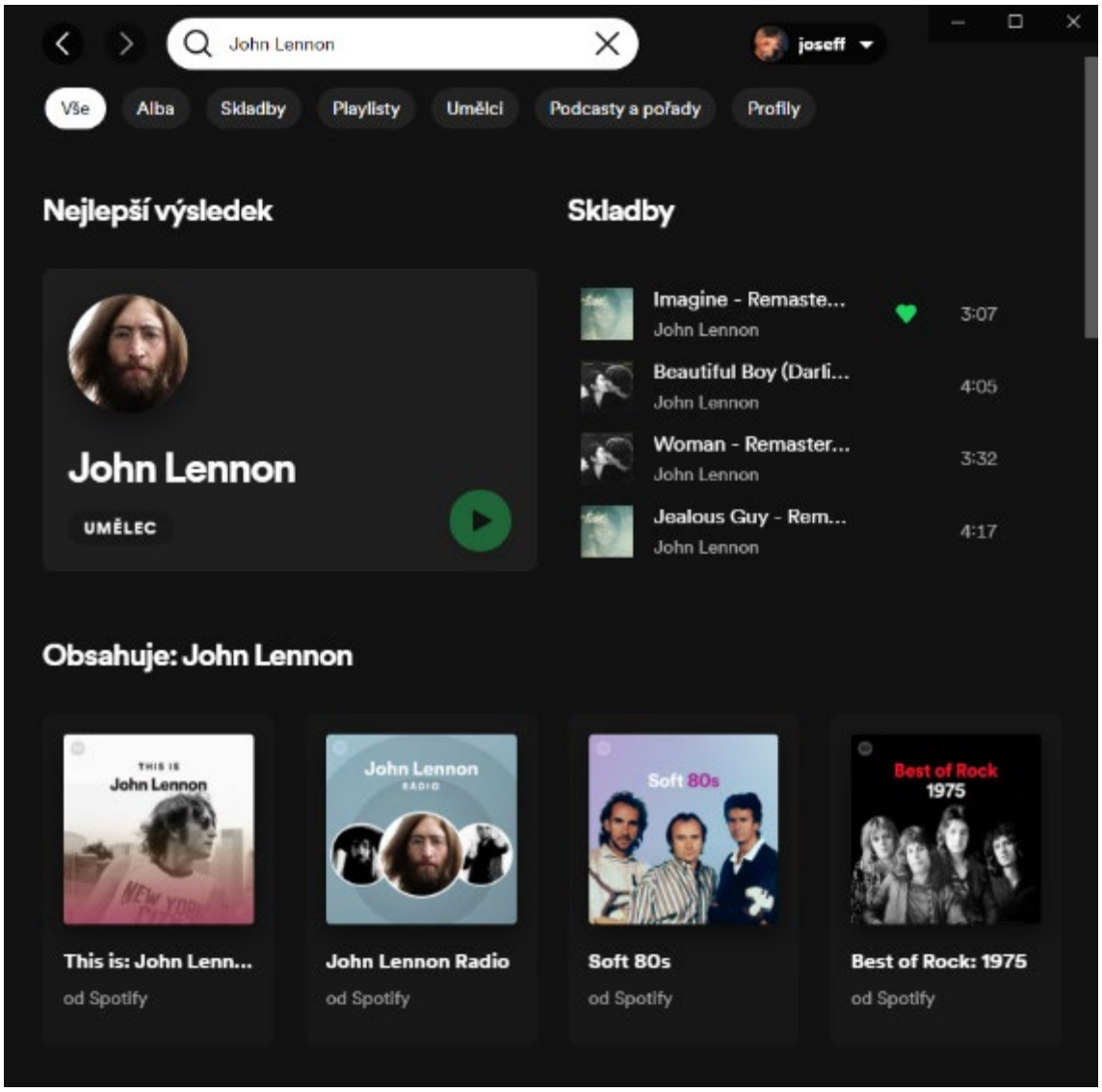


Obrázek 1 - Ovládací panel Spotify přehrávače. Zdroj: autor

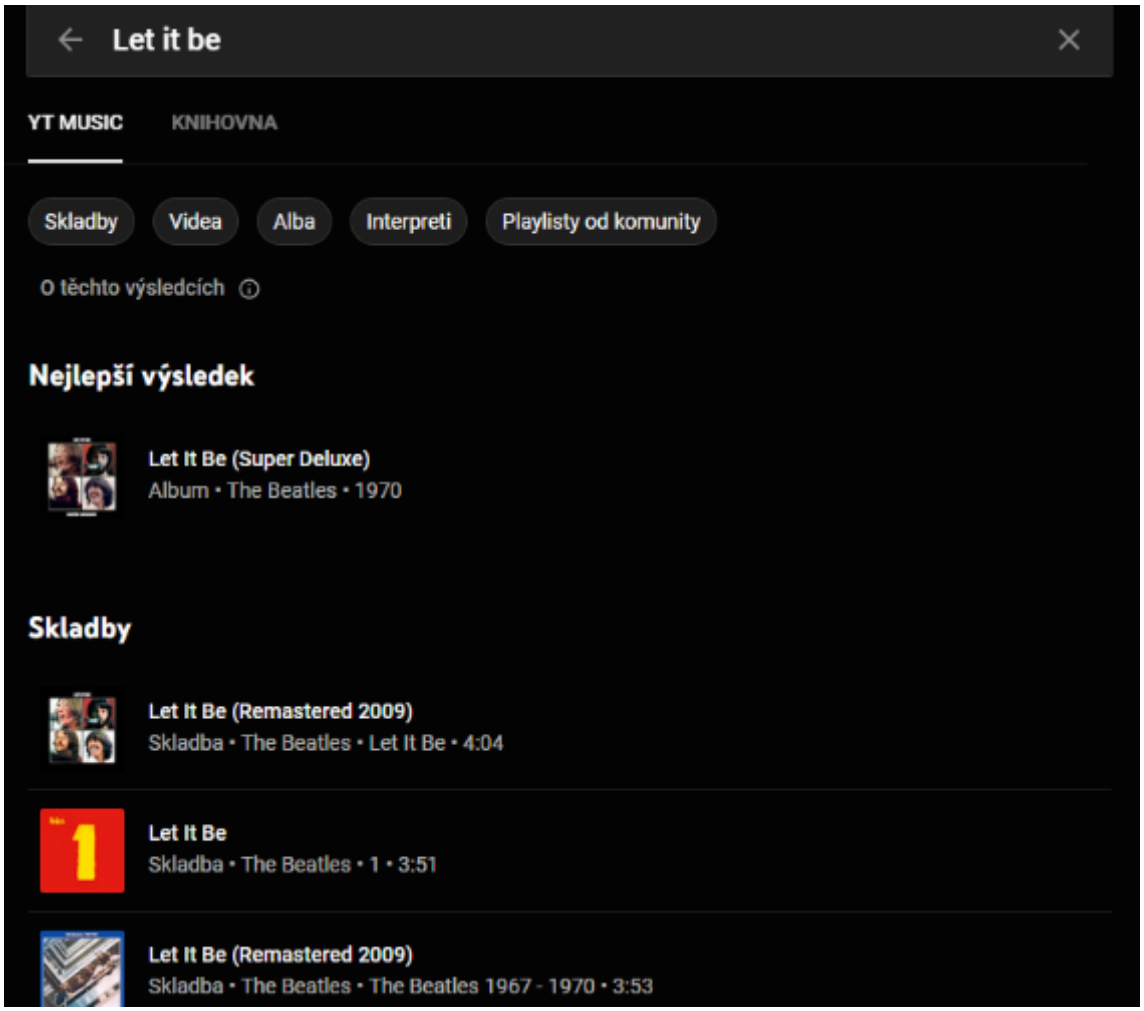


Obrázek 2 - Ovládací panel YouTube Music přehrávače. Zdroj: autor

Další vlastností přehrávačů bývá velmi často vyhledávání jejich obsahu. Tudíž mnoha přehrávačů nabízí textové pole, ve kterém může uživatel zadat svůj text a jsou mu vráceny zpátky výsledky vyhledávání jako např. alba, videa, skladby, seznam skladeb atd.

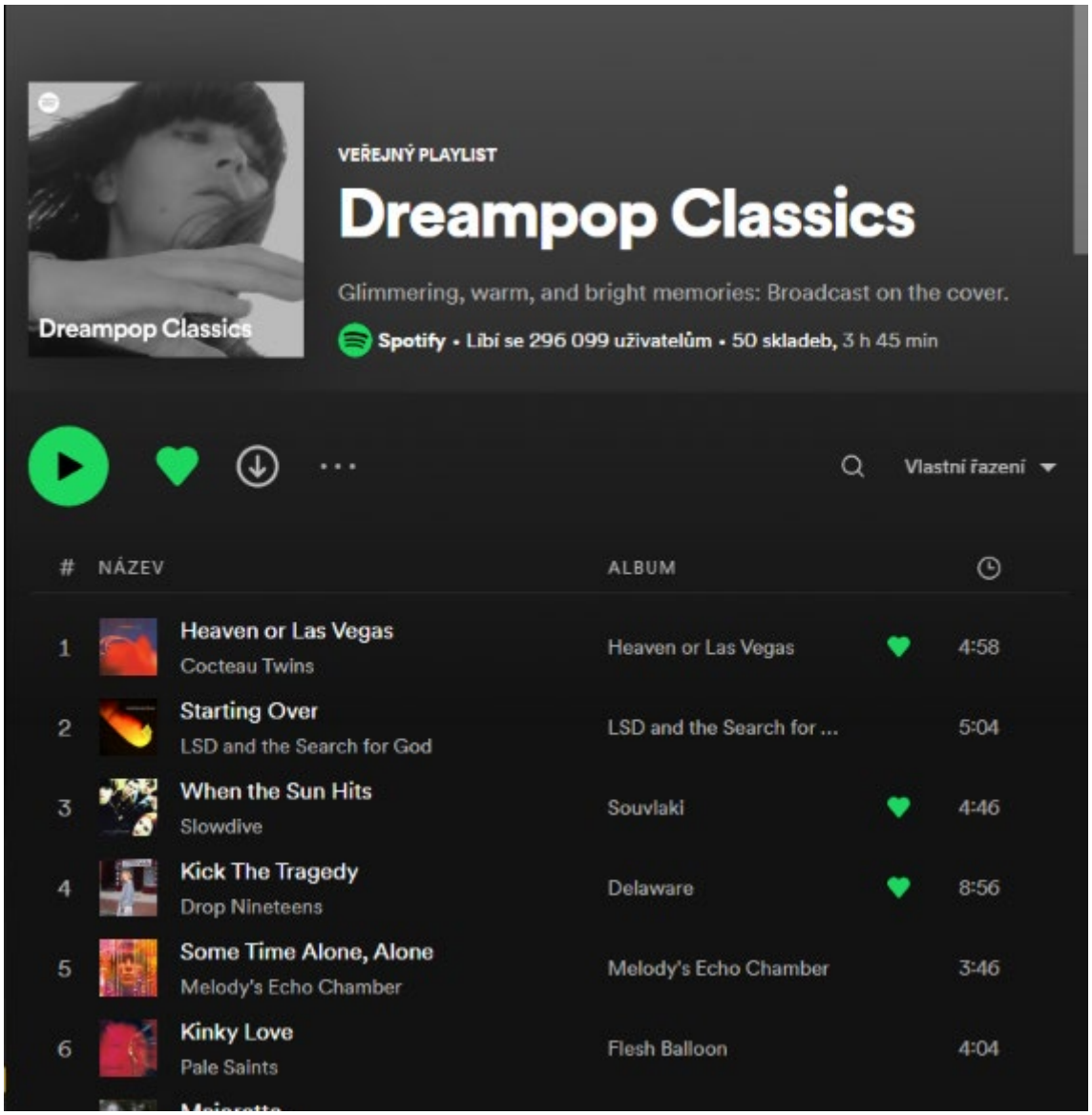


Obrázek 3 - Vyhledávání ve Spotify přehrávači. Zdroj autor



Obrázek 4 - Vyhledávání na YouTube Music. Zdroj autor

Uživateli je také umožněno přidávat skladby do fronty, kde jsou písně uložený v takovém pořadí, v jakém je uživatel vloží. Pokud momentálně puštěná píseň dohraje, je z fronty nasazena další skladba. Pokud uživatel chce, lze vytvářet seznamy skladeb dle libosti, které se uloží na účet uživatele.



Obrázek 5 - Reprezentace seznamu skladeb ve Spotify přehrávači. Zdroj autor

5 Použité technologie

Přehrávače Spotify a YouTube používají k provozu určité nástroje, které jsou veřejnosti volně dostupné. Vývojáři tedy mohou využít těchto nástrojů k vytvoření vlastního klonu přehrávače ve své aplikaci. V této kapitole jsou popsány tyto nástroje.

5.1 API

API se označuje jako programovací rozhraní, které obsahuje sbírku procedur, funkcí, tříd či protokolů splňující určité operace. Tímto lze využít dané API a implementovat ho do své aplikace. API také určuje, jakým způsobem jsou funkce knihovny volány ze zdrojového kódu programu. [1]

Nejpopulárnější forma API, které existuje u aplikací se nazývá Web API. Tento druh API je typicky definován pomocí HTTP (HyperText Transfer Protocol) a požaduje zprávy spolu s definicí struktury odpovědí obvykle v XML (Extensible Markup Language) nebo JSON (JavaScript Object Notation) formátu. Pokud uživatel nebo vývojář potřebuje provést určitou funkci aplikace nebo získat data, musí odeslat http požadavek, který je předem definovaný, a tím je po zavolání vrácena zpátky odpověď.[2]

Pro účely vyvíjené aplikace v rámci této bakalářské práce jsou potřeba následující API.

5.1.1 YouTube IFrame Player API

JavaScript knihovna YouTube Player API poskytuje samostatný vestavěný přehrávač a obsahuje všechny potřebné nástroje k ovládání přehrávače. Tento přehrávač je integrován do HTML stránky pomocí IFrame elementu. [3]

5.1.2 YouTube API

YouTube API poskytuje funkce a procedury pro vyhledávání videí, komentování, nahrávání videí a další funkce. Pro účely bakalářské práce postačí funkce pro vyhledávání videí a seznamů skladeb, které se vyskytují na YouTube. [4]

5.1.3 Spotify API

Spotify API poskytuje identické funkce jako YouTube API. Umožňuje vyhledávání, vypisování skladeb, seznamů skladeb, umělců, uživatelů a mnoho dalších typů dat.

5.2 Standard Development Kit

Standard Development Kit je sada nástrojů, která jsou poskytovaná výrobcem hardwaru, operačního systému, programovacího jazyka, nebo aplikace. Tyto nástroje lze využít pro vývoj aplikací pro specifickou platformu, systém nebo programovací jazyk. Typicky SDK zahrnuje kompilátor, debugger a API, ale mohou obsahovat i jiné nástroje např. dokumentace, knihovny, editory, ovladače a jiné. [5]

5.2.1 Web Playback SDK

JavaScriptová knihovna Web Playback SDK poskytuje veškeré nástroje potřebné k provozu a řízení playbacku hudby, které je možné nasadit na klientské aplikaci.

6 Požadavky aplikace

Pro úspěšnou integraci požadovaných streamovacích platforem je nejdříve potřeba zvážit jaké funkce tato aplikace bude nabízet.

6.1.1 Funkcionality aplikace

Podle předchozí analýzy rozhraní a funkcionalit přehrávačů bylo zjištěno, že se ve svém chování velice podobají. Funkce, které by vyvíjená aplikace měla obsahovat, jsou následující:

- Vyhledávání hudby
- Vytváření seznamů skladeb
- Ukládání oblíbených skladeb
- Řízení přehrávání hudby pomocí ovládacího panelu
- Zavedení fronty

Z hlediska smyslu a fungování této aplikace by měla splňovat to, aby přehrávala obsah jak z YouTube, tak ze Spotify. Zároveň by měla poskytovat, prostředí, které je responzivní a navigace mezi stránkami neovlivní přehrávání obsahu.

Jako doporučený požadavek lze zařadit krátkou dobu odezvy. Pokud aplikaci dlouho trvá zpracovat určité požadavky, uživatel může být lehce odrazen od používání aplikace.

6.1.2 Požadavky pro přehrávače

Aby YouTube přehrávač mohl bez problému fungovat, dle dokumentace YouTube IFrame API vyžaduje to, aby uživatelův prohlížeč podporovala HTML5 vlastnost postMessage. Tato vlastnost je podporovaná u velké většiny prohlížečů jak na stolních počítačích, tak na mobilních telefonech. Po nasazení vestavěného přehrávače do uživatelského rozhraní musí mít velikost minimálně 200 pixelů na šířku a výšku. Pokud vývojář chce využít jeho IFrame API, webová stránka musí implementovat JavaScript funkci onYoutubeIFrameAPIReady. Tato funkce zajistí vytvoření vestavěného přehrávače a načtení jeho příslušného videa. [3]

Web Playback SDK zajišťuje podporu u všech nejpopulárnějších prohlížečů. Momentálně tu existuje ale pár limitací.

- Pokud uživatel předá ovládání přehrávání z jiného zařízení na zařízení s iOS, přehrávač automaticky nezapne svoje přehrávání. Vzápětí uživatel musí interagovat s jeho nástroji a událostmi, aby ho zprovoznil zpátky.

- V systému iOS nemůže přehrávač a jeho nástroje fungovat, pokud prohlížeč běží v pozadí.

- Aby uživatel mohl úspěšně přehrávat hudbu u jiné aplikace než Spotify, musí mít odběr na Spotify Premium.

7 Návrh aplikace

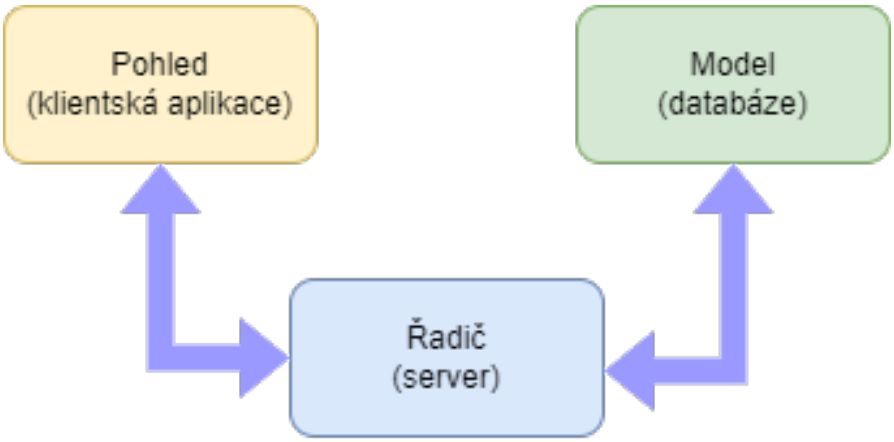
7.1 Architektura aplikace

Jelikož YouTube a Spotify přehrávače slouží jako JavaScript knihovny, vhodným prostředím pro jejich zprovoznění je vytvoření nové webové aplikace. Aplikace bude vyvíjena v jazyce TypeScript (JavaScript) a postavená na principu MVC (Model-View-Controller).

Model-View-Controller je softwarová architektura, která se stala velkou konvencí u webových aplikací. Účelem této architektury je rozdělit vyvíjenou aplikaci na tři komponenty. Pohled (View), Řadič (Controller) a Model.

- Model se chová jako komponent k reprezentaci dat, s nimiž aplikace pracuje.
- Řadič reaguje na události, které iniciuje uživatel, a zajišťuje změny v modelu. Na základě těchto změn se aktualizuje samotný pohled.
- Poslední komponent „pohled“ převádí data získána od řadiče do podoby vhodné k reprezentaci uživateli.

Tímto principem je zajištěna nezávislost mezi vrstvami a snadné rozdělení aplikace. Podle této architektury dojde k rozdělení na klientskou aplikaci, server a databázi.



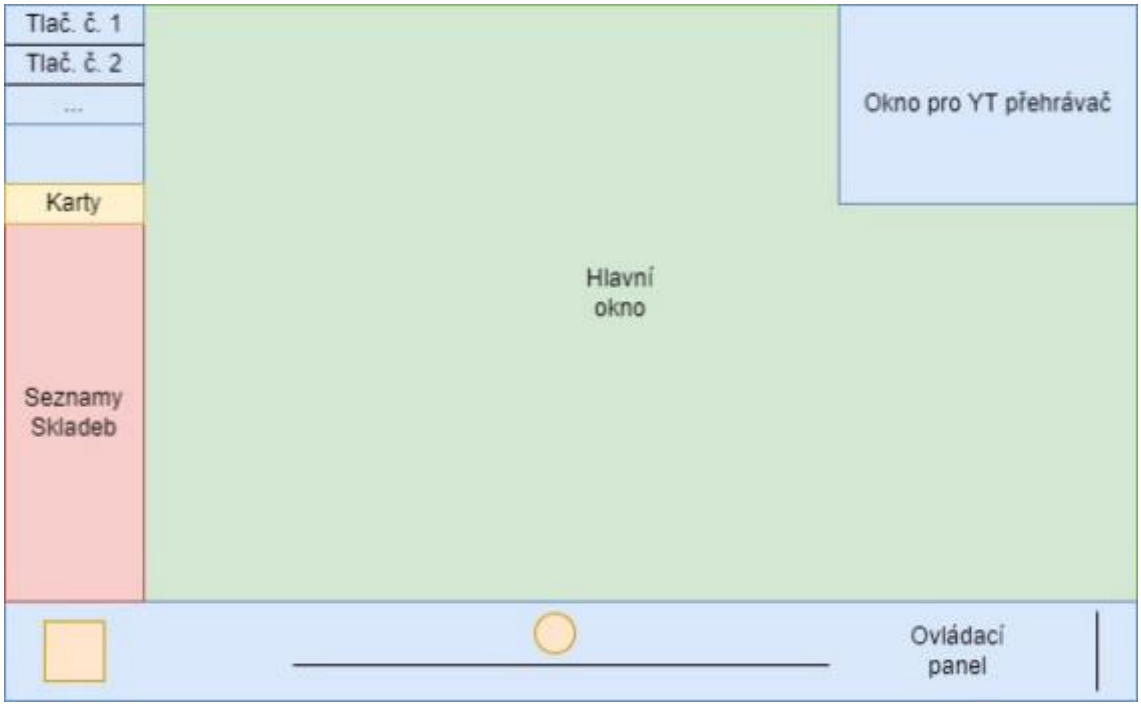
Obrázek 6 - Model MVC. zdroj: autor

7.2 Návrh klientské aplikace

7.2.1 Návrh uživatelského rozhraní

Prvním krokem pro vytvoření klientské aplikace je navržení a vytvoření uživatelského rozhraní, které umožní pohodlné navigování v aplikaci a přehrávání hudby. Při ovládání přehrávačů je potřebné uživateli předat zpětnou vazbu a zobrazit jejich změněný stav.

7.2.1.1 Základní stavební prvky



Obrázek 7 - Grafická reprezentace uživatelského rozhraní. Zdroj autor

Základní uživatelské rozhraní se skládá ze čtyř části:

- Boční panel – V této sekci se budou objevovat veškeré odkazy nebo tlačítka pro navštívení určitých stránek a seznamů skladeb.
- Ovládací panel – Tento panel bude sloužit k ovládání přehrávačů obou platforem. Obsahuje ukazatele průběhu, tlačítka k ovládání přehrávání hudby, informace o momentálně přehrávané hudbě a ukazatele hlasitosti.
- Hlavní okno – Okno sloužící pro zobrazení veškerého obsahu, pro vyhledávání a spouštění hudby.

- Okno pro YouTube přehrávač – Tento rámec je dedikovaný pouze pro YouTube přehrávač. Pokud ho chceme používat, je potřeba ho nasadit do určitého okna. Pokud uživatel chce, je mu umožněna maximalizace nebo minimalizace okna.

Tyto části rozhrání budou vždy zobrazeny při používání aplikace.

7.2.1.2 Vyhledávání a zobrazení jeho obsahu

Pro získávání obsahu z databází Spotify a YouTube je také potřeba rozvrhnout jakým způsobem reprezentovat tyto data. Způsob zobrazení dat se bude lišit na základě momentálně vybraného typu přehrávače.

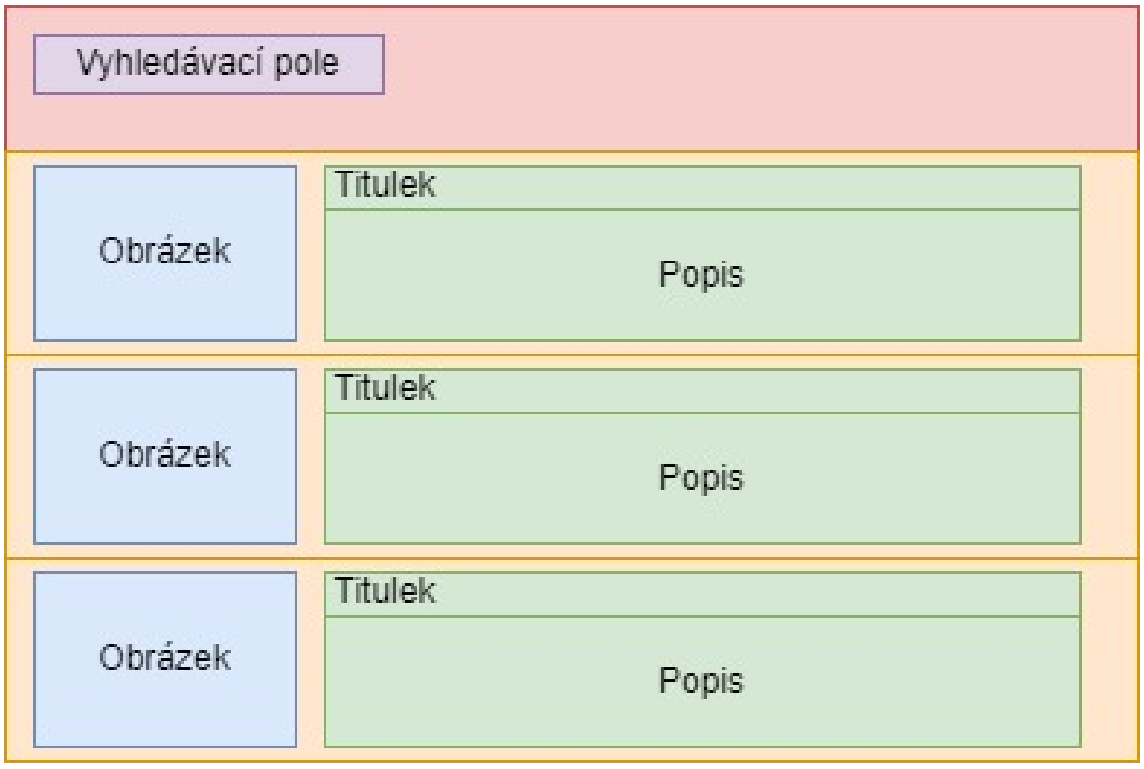
Obrázek 8 - Návrh pro zobrazení dat při vyhledávání ve Spotify. Zdroj autor

Pro vyhledávání ve Spotify budou data rozdělena do takových skupin, které budou specifikovány při získávání výsledků:

- Skladby
- Seznamy skladeb
- Umělci
- Alba

Uživateli bude umožněno se navigovat na samostatnou stránku umělců, alb a seznamů skladeb.

U YouTube se data budou zobrazovat odlišným způsobem.



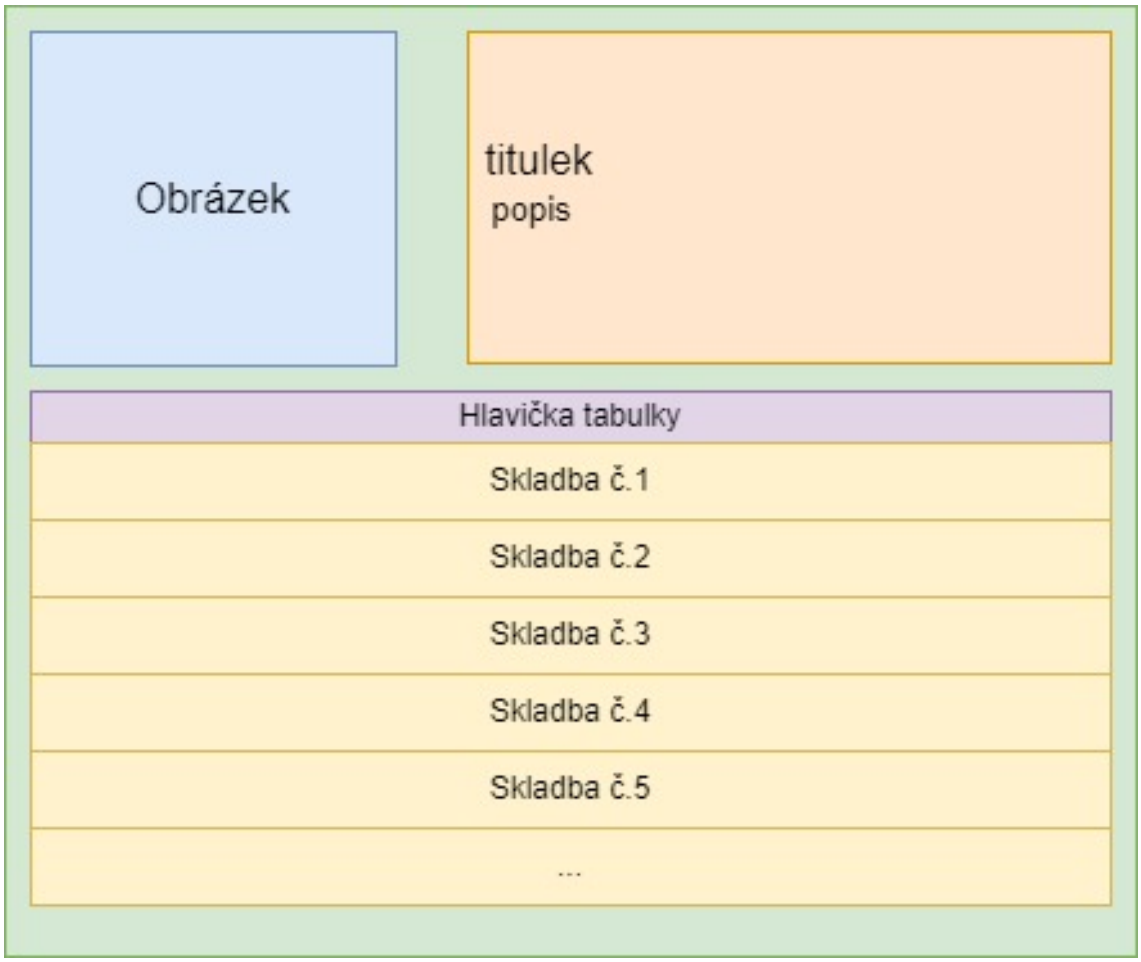
Obrázek 9 - Návrh rozložení výsledku vyhledávání u YouTube. Zdroj autor

Každý výsledek je reprezentován jedním řádkem v okně. Řádek s výsledkem bude obsahovat sekci pro obrázek, titulek a popis. Výsledek může reprezentovat buďto video nebo seznam skladeb.

7.2.1.3 Zobrazení alba a seznamů skladeb

Reprezentace alba nebo seznamu skladeb spočívá v zobrazení jejích informací a výpisu jednotlivých skladeb. Tato stránka se bude skládat ze tří částí:

- Obrázek – Obrázek alba nebo seznamu.
- Kontejner pro popis – V něm jsou obsažený různé informace jako např. titulek, počet skladeb, tvůrce, umělec a jiné.
- Tabulka skladeb

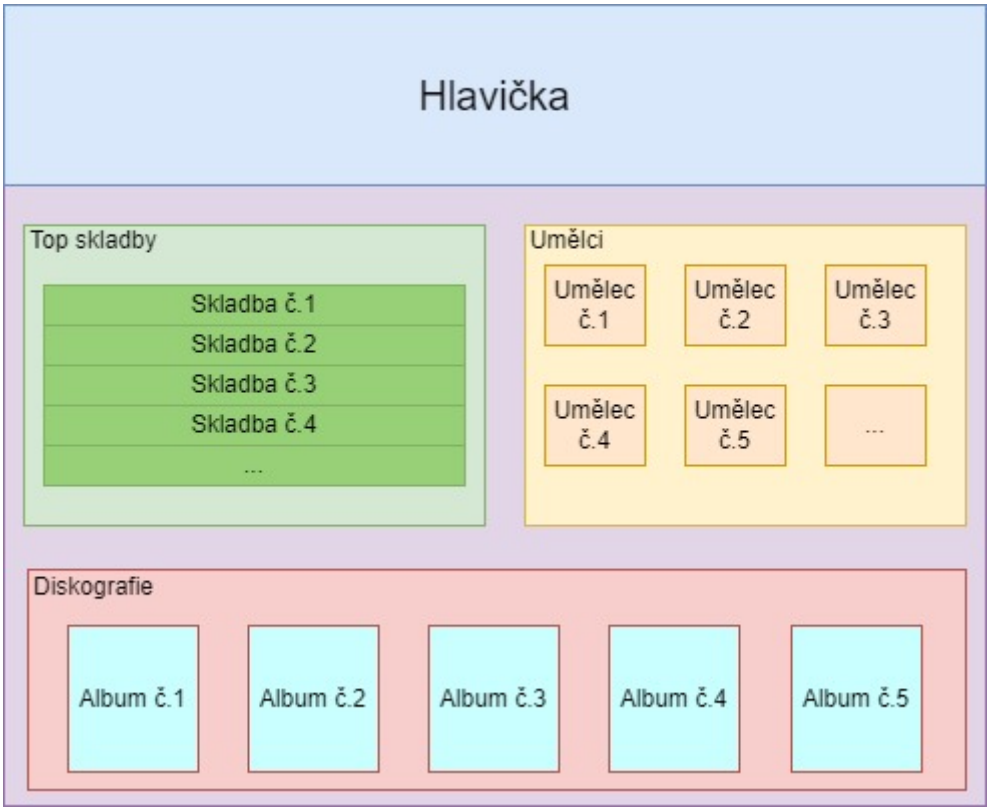


Obrázek 10 - Návrh rozložení stránky pro album a seznam skladeb. Zdroj autor

7.2.1.4 Zobrazení umělce Spotify

Stránka pro Spotify umělce bude rozložena na čtyři části:

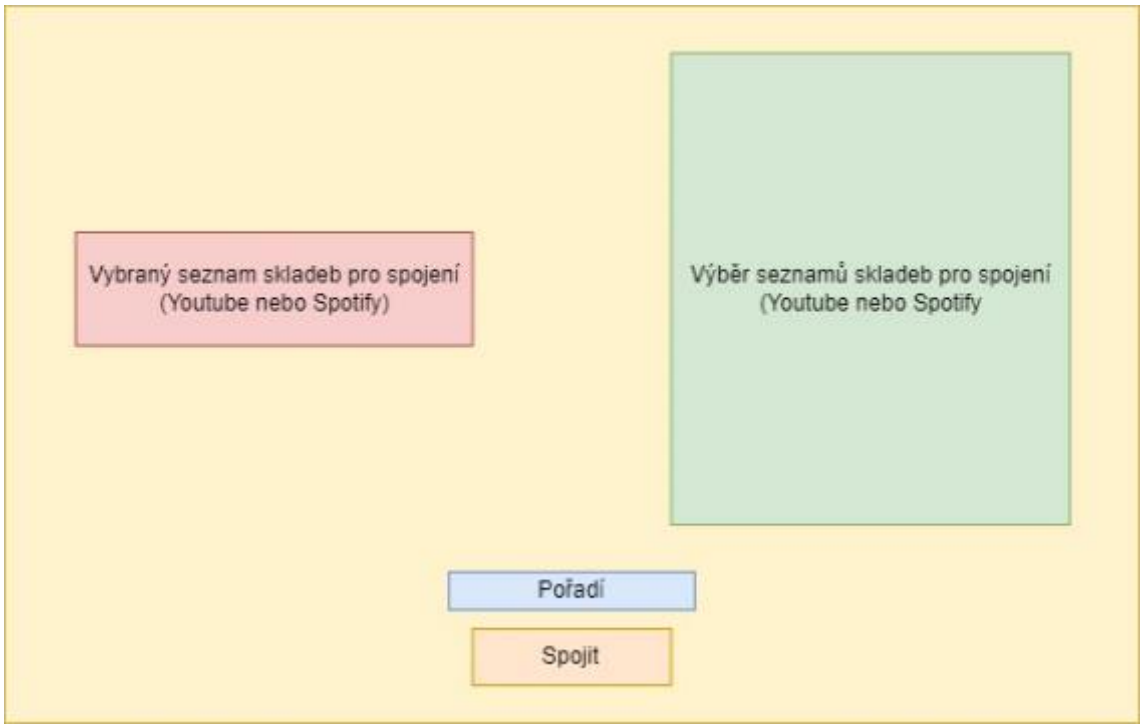
- Hlavička – Obsahuje název a banner umělce.
- Top skladby – Tabulka nejlepších skladeb umělce.
- Podobní umělci – Umělci, kteří se podobají momentálně zobrazenému umělci.
- Diskografie – Obsahuje všechny alba umělce, které vydal.



Obrázek 11 - Rozložení stránky pro Spotify umělce. Zdroj autor

7.2.1.5 Zobrazení stránky pro spojení dvou seznamů skladeb

Pokud uživatel chce, bude mu umožněno spojit seznam skladeb z YouTube a Spotify dohromady. Pro to je potřeba vytvořit stránku, která umožní využít tuto funkcionalitu. Stránka je znázorněna následujícím obrázkem.



Obrázek 12 - Návrh stránky pro spojení dvou seznamů skladeb. Zdroj autor

Na jedné straně bude zobrazen seznam skladeb vybraný uživatelem a na druhé straně bude moci uživatel vyhledávat dle určitého typu přehrávače seznam skladeb. Pokud uživatel vybere jeden seznam skladeb z vyhledávání a požadované pořadí, pomocí tlačítka „Spojit“ zahájí vytvoření nového seznamu skladeb, který obsahuje skladby z obou spojených seznamů.

7.2.1.6 Zobrazení stránky pro oblíbené sklady a frontu

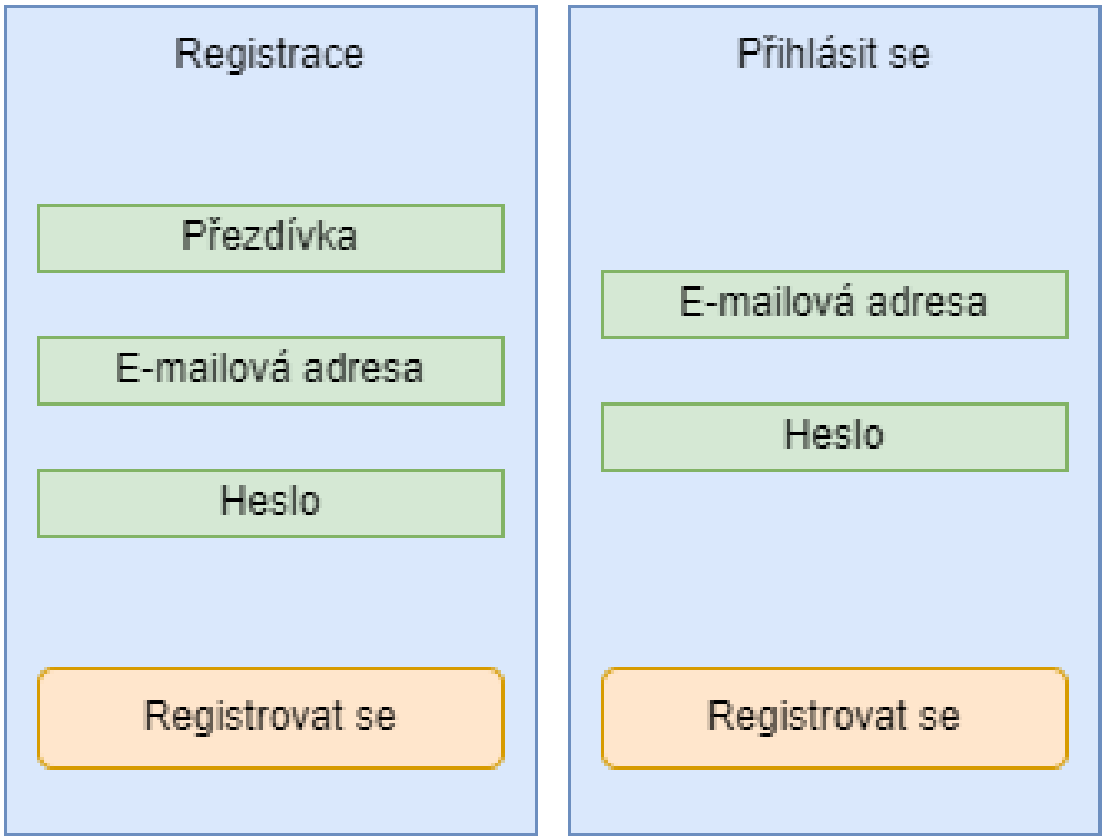
Jakým způsobem se budou tyto stránky zobrazovat je rozvrženo následujícím způsobem.

Obrázek 13 - Návrh stránky pro frontu a oblíbené skladby. Zdroj autor

V hlavičce se bude nacházet název navštívené stránky a v tabulce budou načteny jednotlivé skladby. Na stránce s frontou budou zobrazeny skladby, jenž uživatel uložil do fronty a na stránce s oblíbenými písněmi budou zobrazeny skladby, které si uživatel oblíbil.

7.2.1.7 Registrace a přihlášení

Pro registraci a přihlášení uživatele je potřeba vytvořit stránku, ve které budou zobrazeny formuláře pro doplnění údajů. Pro registraci bude uživateli zobrazen formulář s textovými poli pro e-mailovou adresu, přezdívku a heslo. Pří přihlášení bude po uživateli požadována jenom e-mailová adresa a heslo.



Obrázek 14 - Návrh formulářů pro registraci a přihlášení. Zdroj autor

7.2.2 Zajištění správného chodu aplikace

Funkcionalita této práce, která nesmí být pominuta, je přepínání mezi jednotlivými přehrávači. Pro dosažení takového plynulého chodu, je potřeba implementovat vlastní globální stav nad celou aplikací.

Velmi zásadní podmínkou vyvíjené klientské aplikace je zajištění plynulé navigace mezi stránkami, které uživatel navštěvuje. Pokud interaguje mezi jednotlivými stránkami, není žádané, aby se při návštěvě nové stránky aktualizovala celá aplikace. Toto chování způsobí neustále přerušování hudby. Velmi dobrou volbou pro splnění tohoto požadavku je vytvořit klientskou aplikaci na principu architektury Single-Page Application.

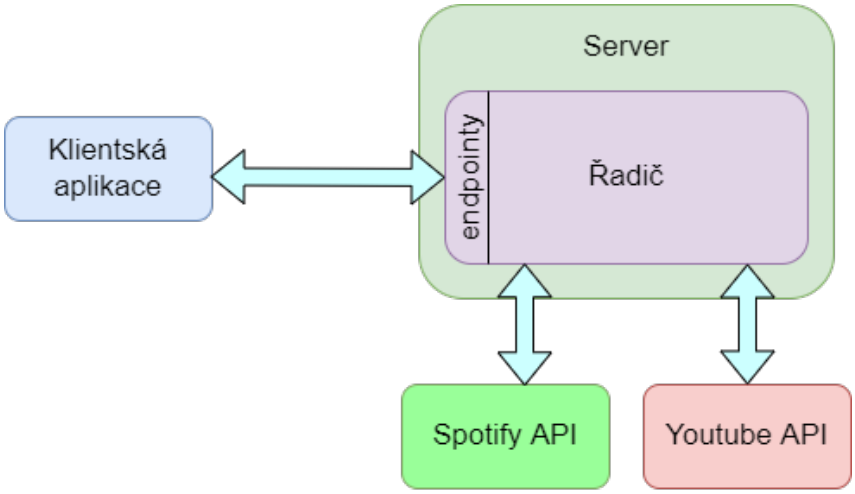
7.2.2.1 SPA (Single-Page Application)

Single-Page Application je tzv. jednostránková aplikace, jejíž veškerý obsah je tvořen jedinou stránkou. Tímto způsobem je eliminována potřeba prohlížeče znovu načítat webovou stránku pro aktualizaci uživatelského rozhraní. Lze dynamicky měnit určité části webové aplikace, které jsou potřeba, a přináší rychlou manipulaci a rychlejší načítání. Takové chování je dosaženo za pomoci JavaScriptových frameworků jako např. React, Angular, Vue, Knockout a jiné. [6]

Velmi častým principem těchto frameworků je využití konceptu tzv. komponent, které rozkládají stránku na části (komponenty) a tím je umožněna jejich univerzálnost a znovupoužití u jiných aplikací.

7.3 Návrh serveru

Navržení serveru spočívá v určení koncových bodů (endpointů) a vytvoření logiky pro zpracovávání požadavku od uživatele a pro práci s databází nebo jinými API. Velkou zásadou při vytváření serveru a jeho koncových dobu je zavedení mechanismu, který zajistí přijímání takových požadavků s doménou, která je na serveru povolená. Pokud tento mechanismus server neobsahuje, všechny ostatní aplikace mají přístup k tomuto serveru, a to přináší mnoho bezpečnostních rizik. K tomu, aby uživatel mohl úspěšně vyhledávat a zobrazovat obsah Spotify nebo YouTube, je potřeba na serveru zavést knihovny potřebné ke komunikaci s rozhraním Spotify a YouTube.



Obrázek 15 - Reprezentace návrhu serveru. Zdroj autor

Prostředí vhodné pro vývoj serveru je Node.js, jelikož využívá nejpopulárnější jazyk JavaScript a oba platformy pro něho poskytují knihovny.

7.4 Návrh databáze

K ukládání uživatelem určené skladby je potřeba zavést databázi, která bude obsahovat následující tabulky:

- Uživatelé (Users) - Zde budou uložené veškeré údaje o uživateli.
    - Vlastnosti uživatele – Heslo, e-mail, přezdívka a identifikační číslo.
- Skladby – Při ukládání skladeb tu zůstává jedna důležitá otázka, jakým způsobem se budou dané skladby ukládat. Dle uvážení existují tří způsoby:
    - Ukládat identifikační kód k písničce a typ přehrávače.

        - Tímto způsobem musí řadič vždy požádat o data od rozhrání třetí strany dle získaných identifikačních kódů z naší databáze. Výhodou tohoto přístupu je zajištění relevantnosti získaných dat, snadná implementace a nevyžaduje velké množství místa v databázi. Na druhou stranu vyžaduje větší nároky na počet požadavků, které aplikace potřebuje odeslat k získání dat.
    - Ukládat objekty skladeb.
        - Pokud si databáze ukládá samotné objekty skladeb, není potřeba požadovat o data ze třetí strany. Tímto řadič může rovnou odeslat data ke klientovi, čímž jsou ušetřeny zdroje aplikace. Nevýhodou tohoto přístupu je irelevantnost požadovaných dat a větší nároky na kapacitu uložiště v databázi.
    - Ukládat objekty skladeb a dle potřeb je aktualizovat.
        - Sloužící jako hybridní typ mezi dvěma předchozími způsoby, databáze si ukládá celé objekty skladeb. Pokud je ale nutné, řadič si v určitých intervalech požádá o aktualizaci dat. Tento způsob do určité míry zajišťuje výhody obou způsobu. Na druhou stranu je těžší pro implementaci a hrozí vysoké nároky pro kapacitu uložiště.
    - Dle rozhodnutí bude databáze ukládat jenom identifikační kód a typ přehrávače ke skladbě.
- Seznamy skladeb – Bude obsahovat údaje o vytvořeném seznamu skladeb v aplikaci.
    - Vlastnosti – Identifikační číslo, název a popis.
- Oblíbené skladby – Tato tabulka bude obsahovat záznamy o písničkách, které si uživatel přidá jako oblíbené.
    - Vlastnosti – Identifikační číslo spojení, identifikační číslo uživatele a identifikační číslo písničky.
- Skladby přiřazené k seznamům – Tato tabulka bude obsahovat záznamy o písničkách, které jsou přirazené k vytvořeným seznamům skladeb.

- Vlastnosti – identifikační číslo spojení, identifikační číslo písničky, pořadí, identifikační číslo seznamu.

8 Realizace aplikace

8.1 Implementace klientské aplikace

Celá klientská aplikace je nahrána na GitHub. [7]

8.1.1 Výběr webových technologií k vývoji

První krokem při implementaci klientské aplikace je potřeba vyřešit jaký framework je vhodný k použití programovat. Pro SPA architekturu existuje velké množství frameworků, které lze použít.

8.1.1.1 React

Pro integraci streamovacích platforem do webové aplikace bude využívána JavaScriptová knihovna React. Tato knihovna je převážně vyvíjená Facebookem a využívá se jako základ pro tvorbu SPA. [8]

React komponenty jsou často vytvářený pomocí jejich funkcí nebo tříd. Z těchto komponentů lze postupně vytvářet části stránky a skládat je takovým způsobem, aby reprezentovali vyžadované uživatelské rozhraní. Pokud je potřeba, lze tyto komponenty opakovaně používat u více částí webových stránek. Tím je eliminována redundance kódu a zajišťuje její přehlednost. [9]

Komponenty jsou obvykle psány pomocí tzv. JavaScript XML (JSX), který rozšiřuje syntaxi jazyka JavaScript. Toto rozšíření zajišťuje vytváření HTML elementů pomocí JavaScriptu.

8.1.2 TypeScript

TypeScript je programovací jazyk, který je vyvíjen a udržován Microsoftem. Slouží jako nadstavba JavaScriptu, která přináší mnoho dalších funkcionalit pro zajištění snazšího vývoje. Pro realizaci této aplikace, bude zajištěna separace objektů Spotify od YouTube.

8.1.2.1 Proč právě TypeScript?

Ačkoliv JavaScript je vhodný pro rychlejší programování a začátečníky, kvůli své povaze, může docházet k nepředvídatelnému chování a dokáže být náchylný k chybám.

Pokud jsou definovány v kódu proměnné, není potřeba určit explicitně o jaký typ se jedná. JavaScript dokáže automaticky dedukovat příslušný typ. Následkem takového mechanismu dochází k náročné udržovatelnosti aplikace a jeho nepředvídatelnosti.

TypeScript svým konceptem tento problém eliminuje pomocí statického typování, které je používané u statických programovacích jazyků jako např. C++, Java, C#. Je tedy možné v TypeScriptu definovat třídy, typy, rozhraní a obsahuje genericitu, definice výčtového typu a mnoho dalších vlastností. Tímto chováním zajišťuje lepší předvídatelnost, robustnost, stabilitu a udržovatelnost programu.

Jelikož je TypeScript nadstavbou JavaScriptu, kód napsaný v Javascriptu je zároveň validní v TypeScriptu. To je dosaženo pomocí kompilace souboru na JavaScriptu. TypeScript soubory obsahují příponu .ts nebo .tsx.

8.1.3 Vývoj základního uživatelského rozhraní

Dle předchozího návrhu uživatelského rozhraní je celá aplikace rozdělena na čtyři základní části. Tyto části jsou implementovány jako samostatné komponenty.

Obrázek 16 - Pohled na vytvořené uživatelské rozhraní. Zdroj autor

8.1.3.1 Ovládací panel

Ovládací panel je nejdůležitější části celé aplikace, která umožní uživateli přehrávání hudby za pomoci ovládacích prvků. Komponent je implementován v souboru Player.tsx a skládá se ze 4 částí.

- Informační panel – Zobrazuje uživateli momentálně přehrávanou skladbu, jeho interpreta a obrázek alba, ze kterého pochází.
- Ovládací tlačítka – Tato tlačítka zajišťuji ovládání přehrávané skladby. Mezi funkce patří pozastavení, pokračování a přechod na další nebo předchozí skladbu.
- Ovládání hlasitosti – Tento kontejner v sobě obsahuje posuvník, jimž lze měnit hlasitost přehrávané skladby. Součástí kontejneru je tlačítko, které uživatele naviguje na stránku s jeho momentálně načtenou frontou.

- Ukazatel průběhu – Uživatel může použít tento posuvník k přetáčení přehrávané hudby vpřed nebo vzad. Posuvník také promítá čas, ve které se momentálně přehrávaná písnička nachází.

Obrázek 17 - Rozdělení ovládacího panelu. Zdroj autor

Implementace ovládacího panelu je dosaženo pomocí dílčích komponentů, ze kterých se panel skládá.

```
return (
    <div id="player">
        <InfoPlayerContainer />
        <div>
            <div id="player-buttons-container">
                <PlayerButton disabled={!merger.state.previousSong}
id="prev-button"
                    src="/images/PrevButton.png" execFunc={merger.PrevSong} />
                <PlayerButton disabled={false} src={merger.state.paused ?
stateImg.play : stateImg.pause}
                    text="Toggle Play"
                    id="play-button" execFunc={merger.TogglePlayBack} />
                <PlayerButton disabled={!merger.state.nextSong} id="next-
button"
                    src="/images/NextButton.png" execFunc={merger.NextSong} />
            </div>
            <ProgressBar func={setProgress} />
        </div>
        <div>
            <Link to={"/queue"}>
                <img src="/images/queueIcon.png" alt="error" />
            </Link>
        </div>
        <VolumeSlider isDisabled={!store.getState().state.currentPlayer}
func={setVolume} />
    </div>
);
```

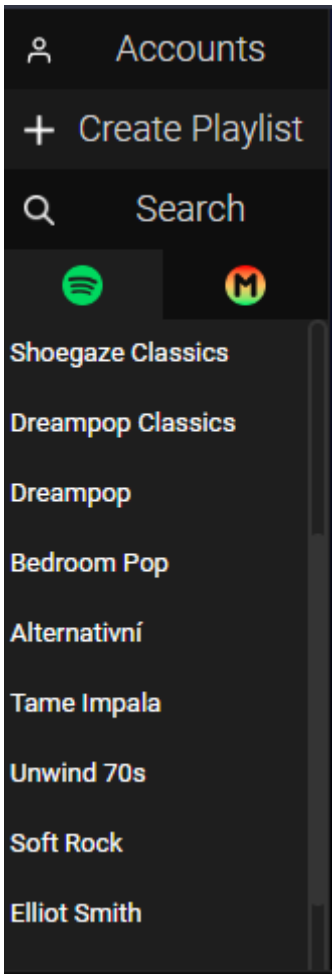
Obrázek 18 - implementace kódu v Player.tsx. Zdroj autor

- InfoPlayerContainer – Chová se jako informační panel.
- PlayerButton – Reprezentuje tlačítko v sekci ovládacích tlačítek. Každé tlačítko má v sobě atributy, na které lze aplikovat funkcionalitu na základě předané funkce. Lze také přidat obrázek a boolean pro zapnutí nebo vypnutí tlačítka.
- ProgressBar – Komponent reprezentující ukazatele průběhu.

- VolumeSlider – Komponent sloužící k ovládání hlasitosti přehrávače.

8.1.3.2 Boční panel

Boční panel slouží pro uživatele jako navigační panel. Panel je implementován jako komponent typu Sidebar a skládá se ze dvou částí.



Obrázek 19 - Boční panel. Zdroj autor

- Kontejner pro tlačítka – Zde může uživatel nalézt tlačítka, kterými lze navigovat skrze aplikaci.

- Seznamy skladeb (Playlists) – Tento komponent v sobě zahrnuje seznam seznamů skladeb (playlistů), které uživatel buďto vlastní nebo sleduje. Dále v sobě obsahuje karty, které umožňují přepínání mezi seznamy skladeb Spotify a této aplikace.

Vykreslovací část komponentu SideBar je složená z následujících komponentů:

- SidebarButton – Reprezentuje jedno z tlačítek v horní části panelu.

- Playlists – Komponent, který zobrazuje uživateli seznam skladeb.

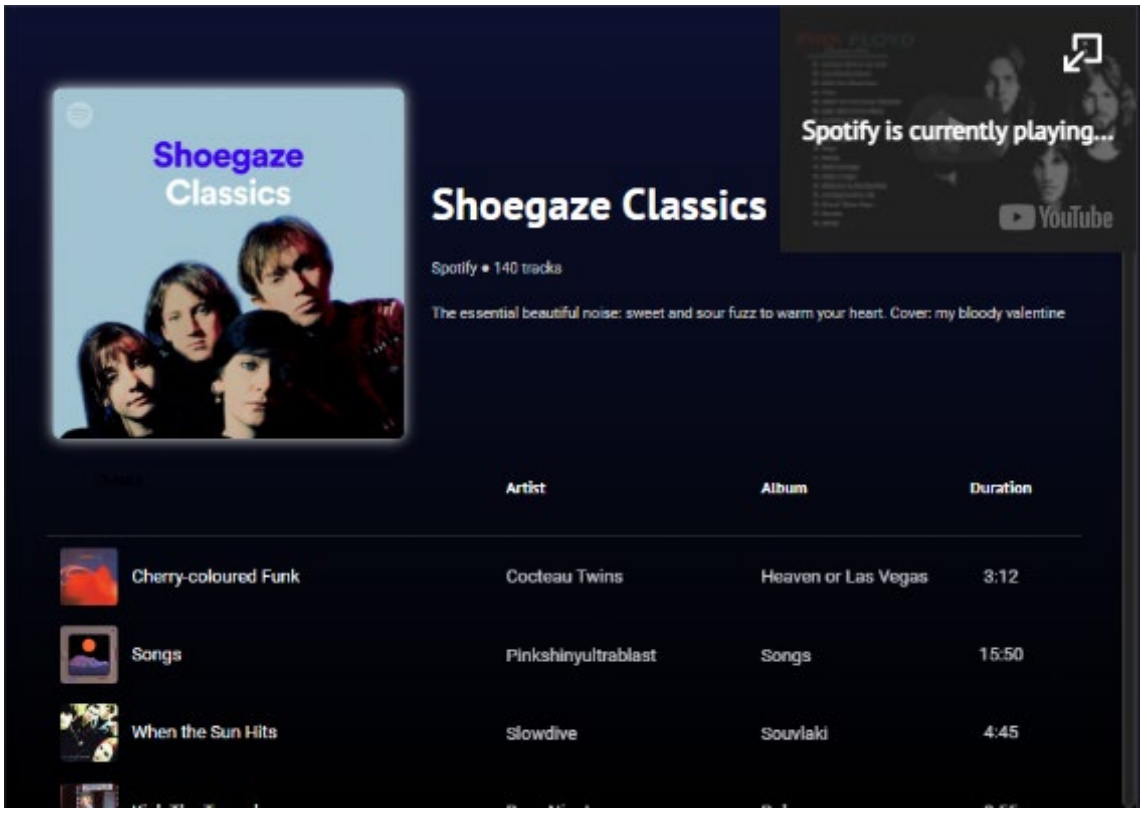
```
return (
    <div id="side-bar">
        <div id="buttons">
            {(spotifyUserInfo || mergerUserInfo) ?
                <>
                    <SidebarButton
                        img="/images/defaultUser.svg"
                        link="/account" text="Accounts" />
                    <SidebarButton
                        img="/images/addimg.png"
                        link="/createPlaylist"
                        text="Create Playlist" />
                </>
                : <SidebarButton
                    img="/images/defaultUser.svg"
                    link="/login"
                    text="Login" />}
            {spotifyUserInfo &&
                <SidebarButton
                    text="Search"
                    img="/images/search.png"
                    link="/spotify/search" />}
            {mergerUserInfo &&
                <SidebarButton
                    text="Liked songs"
                    img="/images/heart.png"
                    link="/likedSongs" />}
        </div>
        <Playlists />
    </div>
)
```

Obrázek 20 - Implementace Sidebar.tsx. Zdroj autor

Dle určitého stavu aplikace může komponent vykreslovat vyžadovaná tlačítka. Např. pokud uživatel není přihlášený, není mu zobrazeno tlačítko pro navigaci k vyhledávání a výpisu oblíbených skladeb. Je zobrazeno pouze tlačítko „Login“, kterým se uživatel musí přihlásit.

8.1.3.3 Hlavní okno

Tento komponent slouží k navigaci a zobrazení veškerých stránek, které uživatel navštěvuje. Obsahuje v sobě dílčí komponent pro zobrazení přehrávače YouTube.



Obrázek 21 - Hlavní okno. Zdroj autor

Pokud uživatel chce roztáhnout nebo zmenšit YouTube přehrávač, lze kliknout na tlačítko umístěné v pravém horním rohu. Komponent je reprezentován v Reactu jako MainWindow.tsx.

```
return (
    <div id="main-window">
        <ShrinkButton />
        <div id="youtube-container">
            <div id="blocking-overlay" style={window.youtubePlayer &&
currentPlayer === Merger.PlayerType.Spotify ? {display:"flex"} :
{display:'none'}}>
                <h3>
                    Spotify is currently playing...
                </h3>
            </div>
            <div id="youtube-player-window">
            </div>
        </div>
        {props.children}
    </div>
)
```

Obrázek 22 - Implementace MainWindow.tsx. Zdroj autor

Komponent obsahuje v sobě kontejner pro YouTube přehrávač a přijímá jako atribut potomky. Tito potomci jsou získávány odlišným způsobem, než je známo v Reactu. komponent MainWindow je získává pomocí knihovny React Router DOM, který zajišťuje navigaci stránek v aplikaci.

```
export const Routes: React.FC = () => (
    <BrowserRouter>
        <div id="main-container">
            <div id="horizontal-container">
                <SideBar>
                </SideBar>
                <MainWindow>
                    <Switch>
                        <Route exact path="/createPlaylist"
component={CreatePlaylistPage}/>
                        <Route exact path="/register" component={RegisterPage}/>
                        <Route exact path="/login" component={LoginPage}/>
                        <Route exact path="/" component={Home}/>
                        <Route exact path="/likedSongs"
component={LikedSongsPage}/>
                        <Route exact path="/account" component={AccountPage}/>
                        <Route exact path="/youtube/playlist/:id">
                            <YoutubePlaylistPage/>
                        </Route>
                        <Route exact path="/merger/mergeWithSpotify/:playlistId">
                            <MergeWithSpotify/>
                        </Route>
```

Obrázek 23 - Úryvek kódu komponentu Routes.tsx. Zdroj autor

React Router DOM poskytuje komponent BrowserRouter, jehož funkcí je zavedení přepínání mezi jednotlivými cestami aplikace. MainWindow se chová jako potomek BrowserRouteru a zakomponuje v sobě Switch komponent, který se stará o přepínání mezi požadovanými stránkami. Každá stránka je reprezentovaná komponentem Route. Aby

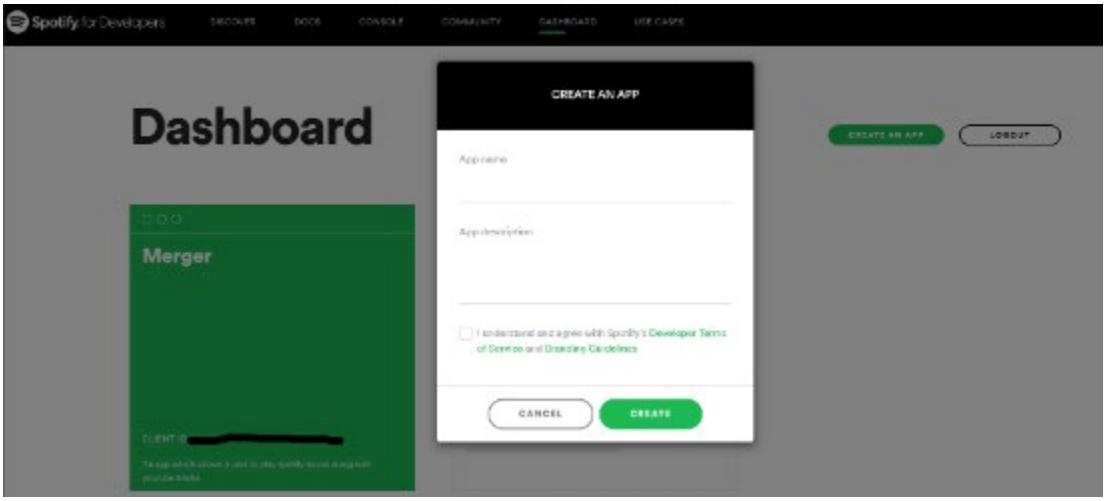
Route správně zobrazil danou stránku, je potřeba do jeho atribut předat cestu a typ komponentu, který bude reprezentovat stránku.

Pokud vývojář chce použít Spotify API, musí provést předem několik kroků před jeho zprovozněním. V tomto popisu je předpokládáno, že vývojář má založený účet Spotify. Bez účtu není možné Spotify API používat.

8.1.4 Příprava před implementací přehrávačů

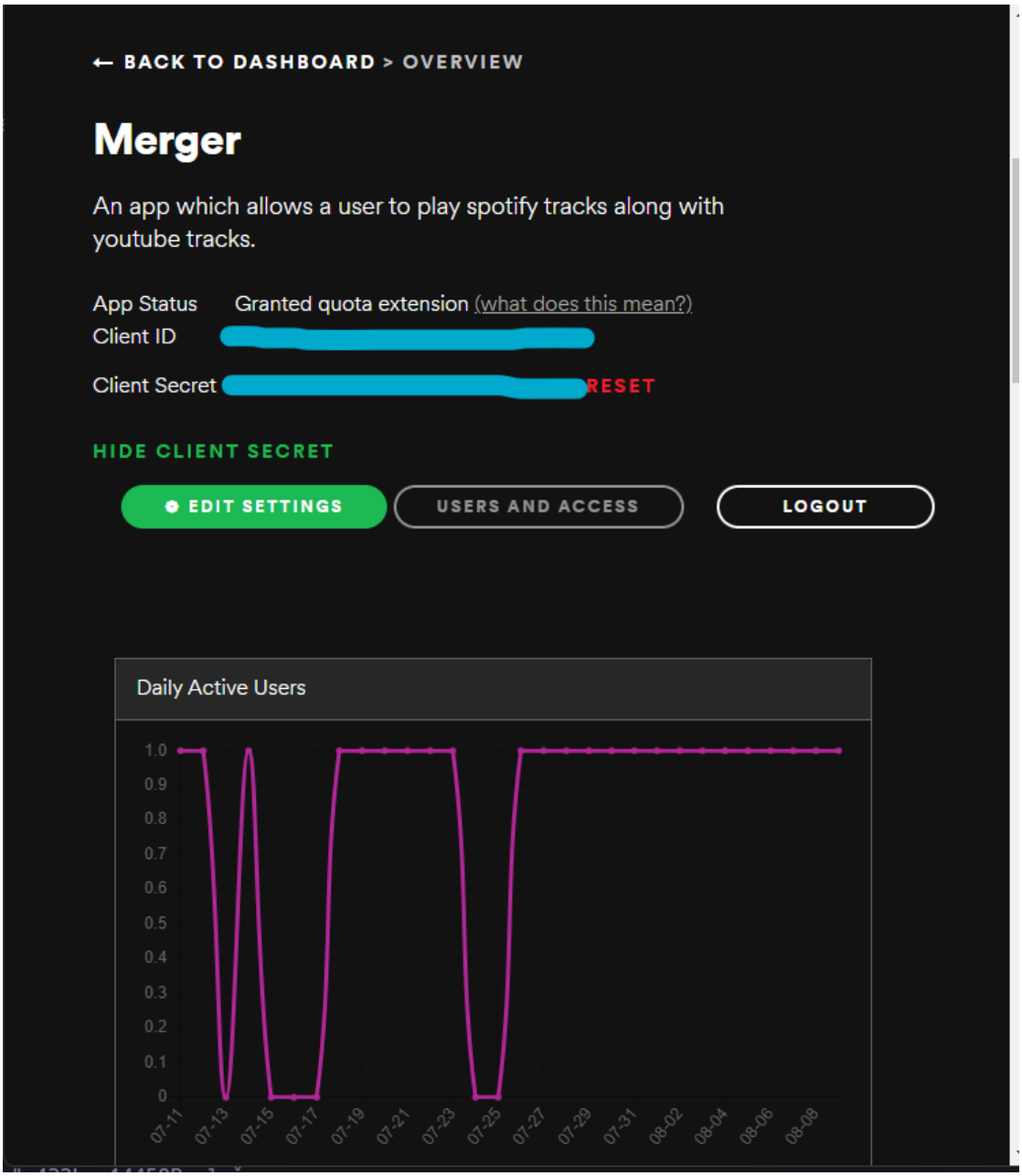
8.1.4.1 Spotify API

Před používáním Spotify API musí vývojář na svém dashboardu zaregistrovat svoji aplikaci, kde zadá název a popis své aplikace. [10]



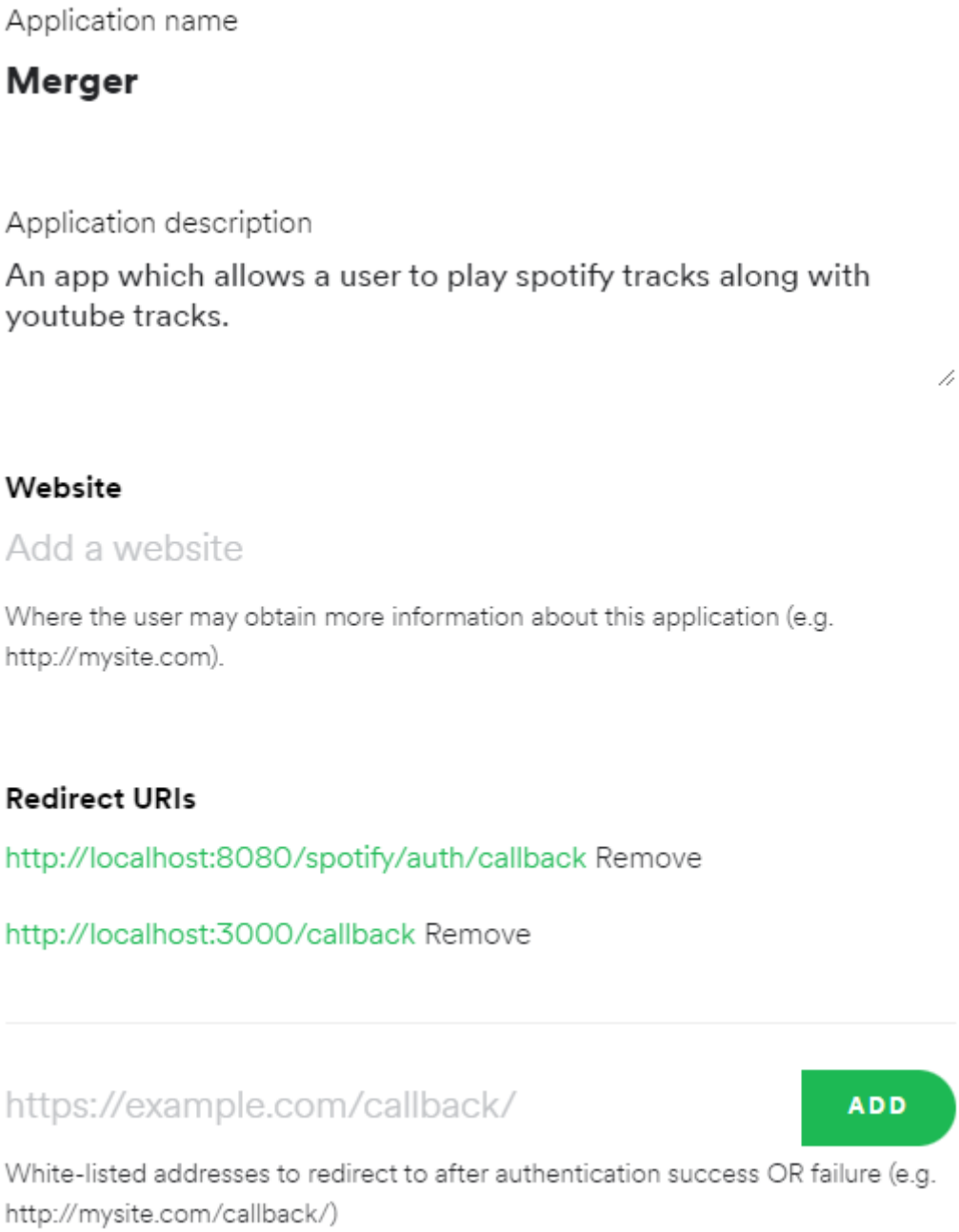
Obrázek 24 - Dashboard uživatele Spotify. Zdroj [11]

Po úspěšné registraci aplikace jsou v jejím přehledu vygenerovány potřebné údaje ke zprovoznění Spotify API nebo Web Playback SDK. Důležitými údaji jsou Client ID a Client Secret, které slouží pro autorizaci použití nástrojů. Vývojáři jsou také zobrazené statistické údaje pro monitorování aktivity aplikace.



Obrázek 25 - Přehled zaregistrované aplikace ve Spotify. Zdroj autor

V přehledu aplikace je dále potřeba nastavit alespoň jednu adresu přesměrování (Redirect URI). Spotify po úspěšné nebo selhané autorizaci přesměruje uživatele na tyto specifikované adresy, kde je případně potřeba provést další operace. Pro tuto aplikaci je zahrnuta adresa přesměrování pro klientskou aplikaci a budoucí server.

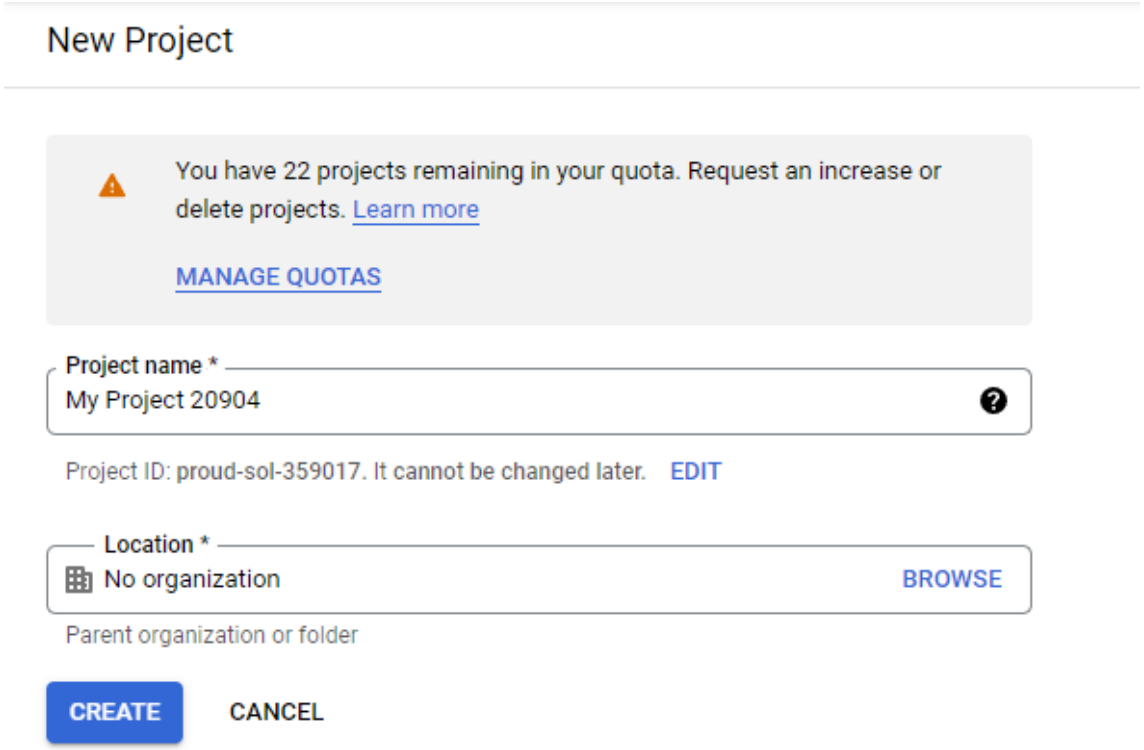


Obrázek 26 - Nastavení adres přesměrování. Zdroj autor

Údaje získané při registraci aplikace budou dočasně uloženy v klientské aplikaci. Později budou přesunuty na server.

8.1.4.2 YouTube API

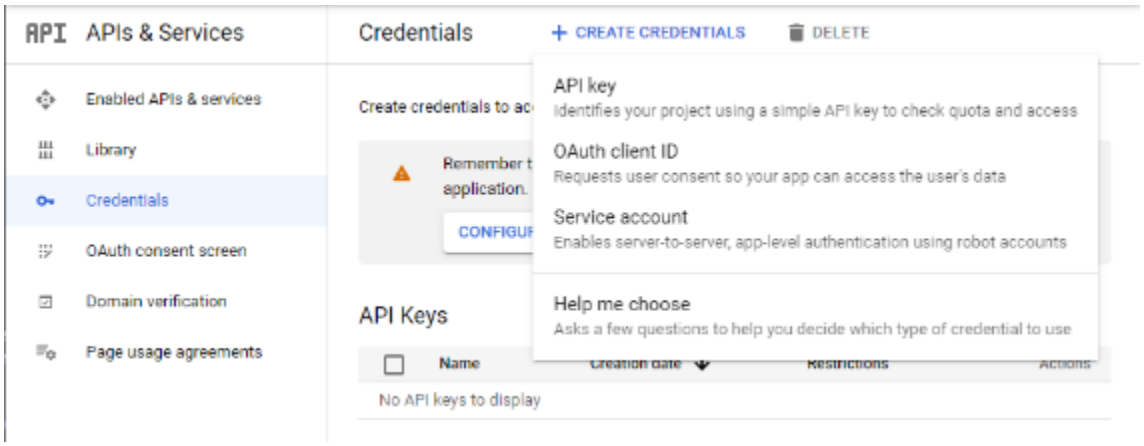
Příprava YouTube Data API probíhá podobným způsobem jako u Spotify. Předtím, než je možné použít YouTube Data API, je potřeba založit na Google Cloudu nový projekt, kde uživatel zadá název projektu a jeho lokaci.



Obrázek 27 - Formulář pro vytvoření nového projektu. Zdroj autor

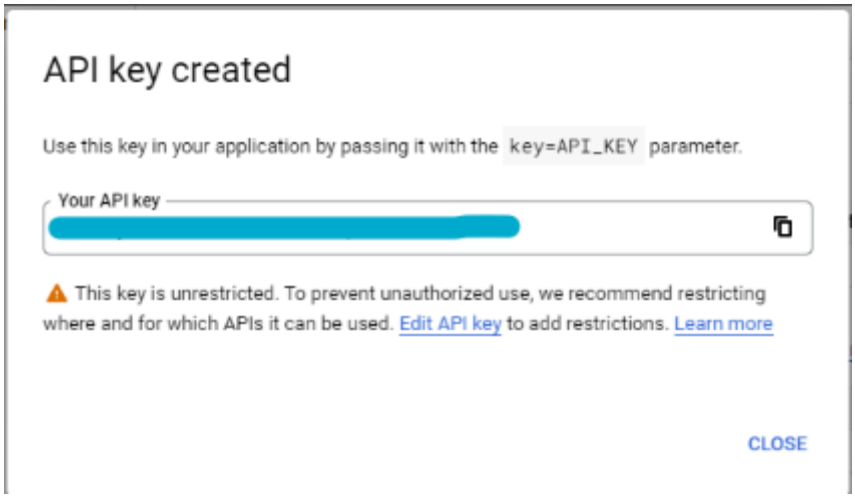
Poté, co vývojář úspěšně vytvořil nový projekt, se musí navigovat do sekce „APIs & Services“ a karty „Credentials“, kde je specifikováno, jakým způsobem bude použití YouTube Data API autorizováno. Existují tři způsoby, jakými lze získat přístup:

1. Pomocí API klíče – Vývojář nemá přístup k osobním údajům nebo soukromým datům uživatele. Tento způsob bude použit, jelikož v této aplikaci nebudou vyžadována osobní data.
2. Pomocí autorizace uživatele (OAuth 2.0)
3. Vytvoření servisního účtu



Obrázek 28 - Vygenerování API klíče. Zdroj autor

Po vygenerování API klíče může vývojář při odesílání požadavku zahrnout do parametrů tento klíč a tímto získá požadovaná data. Tento API klíč bude během implementace klientské aplikace vždy vložený do požadavku, dokud není vytvořený server.



Obrázek 29 - Okénko s vygenerovaným klíčem. Zdroj autor

8.1.5 implementace jednotlivých přehrávačů

Než se mohou tyto přehrávače zprovoznit, je potřeba provést pár kroků, které nám zajistí chod přehrávačů.

8.1.5.1 Instalace Přehrávačů

Instalace u Web Playback SDK je veden přímočarým postupem. Pro jeho provoz je potřeba vložit do aplikace určitý JavaScript kód, který nám zprovozní SDK. [11] V aplikaci je soubor vestavěný při inicializaci SDK nebo může být zahrnut přímo v hlavičce souboru index.html.

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <!-- icon: <link rel="icon" href="%PUBLIC_URL%favicon.ico" /> -->
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <link href="./css/index.css" rel="stylesheet"/>
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%manifest.json" />
    <title>Merger</title>
    <script src="https://sdk.scdn.co/spotify-player.js" ></script>
  </head>
```

Obrázek 30 - Instalace Web Playback SDK. Zdroj autor

Postup instalace YouTube přehrávače je velmi identická předchozímu postupu instalace Web Playback SDK. Pro jeho zprovoznění je potřeba zahrnout určitý kód do index.html. Buďto je přidaný v souboru na pevno, nebo je přidán automaticky poté, co uživatel zapne přehrávání videa z YouTube. [3]

```html
<!DOCTYPE html>
<html>
  <body>
    <!-- 1. The <iframe> (and video player) will replace this <div> tag. -->
    <div id="player"></div>

    <script>
      // 2. This code loads the IFrame Player API code asynchronously.
      var tag = document.createElement('script');

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

Obrázek 31 - Instalace YouTube IFrame API. Zdroj [3]

8.1.5.2 Inicializace Přehrávačů

Jelikož Web Playback SDK a Spotify API při inicializaci vyžadují přístupový token, je potřeba vytvořit funkci, která přihlásí uživatele a poté získá vygenerovaný přístupový token. Jelikož zatím neexistuje server, který by nám zajistil plnou autorizaci, postačí tuto funkci vytvořit na straně klienta pomocí komponentu Login. Tento komponent zahrnuje tlačítko, které po stisknutí přesměruje uživatele na přihlašovací stránku Spotify. Pokud se uživatel přihlásí úspěšně je vrácen přístupový token v URL parametrech v prohlížeči. Jakým způsobem autorizace funguje je popsána pozdější kapitole.

```
export const Login: React.FC = () => {

    const generateRandomString = (length: number) => {
        var text = '';
        var possible =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

        for (var i = 0; i < length; i++) {
            text += possible.charAt(Math.floor(Math.random() *
possible.length));
        }
        return text;
    };

    const handleLogin = () => {

        let state = generateRandomString(16);

        let clientId = '*****************************';
        let scope = 'user-read-private user-read-email';
        let redirectUri = 'http://localhost:3000/callback'

        let authUrl: string | Location =
'https://accounts.spotify.com/authorize';
        authUrl += '?response_type=token';
        authUrl += '&client_id=' + encodeURIComponent(clientId);
        authUrl += '&scope=' + encodeURIComponent(scope);
        authUrl += '&redirect_uri=' + encodeURIComponent(redirectUri);
        authUrl += '&state=' + encodeURIComponent(state);

        window.location.replace(authUrl);
    }

    return (
        <a id="login-button" onClick={handleLogin}>
            Login
        </a>
    )
}
```

Obrázek 32 - Implementace komponentu pro přihlášení uživatele. Zdroj autor

Získaný token je poté uložen do místního uložiště prohlížeče. Je potřeba brát v potaz, že toto řešení je provizorní a tímto způsobem by se z bezpečnostních důvodů nesměl ukládat přístupový token.

```
export const Callback: React.FC = () => {

    const history = useHistory();

    const getHashParams = () => {
        let hashParams: any = {};
        let e, r = /([^&;=]+)=?([^&;]*)/g,
            q = window.location.hash.substring(1);
        while ( e = r.exec(q)) {
            hashParams[e[1]] = decodeURIComponent(e[2]);
        }
        return hashParams;
    }

    useEffect( () => {
        let params = getHashParams();

        localStorage.setItem("access_token", params.access_token);
        history.push("/");
    },[])

    return (<></>)
}
```

Obrázek 33 - komponent pro získání přístupového tokenu. Zdroj autor

Po implementaci funkce lze přejit na inicializaci Web Playback SDK. Inicializace se vykonává v komponentu Player. Pro snadnější implementaci je do aplikace nainstalována knihovna spotify-web-playback-sdk, který aplikaci rozšiřuje o potřebné typy a funkce.

```
const initPlayer = async () => {
   console.log("Initializing player");

   const spotifyScript = document.createElement("script");
   spotifyScript.src = "https://sdk.scdn.co/spotify-player.js";
   spotifyScript.async = true;
   document.body.appendChild(spotifyScript);

   await waitForSpotifyWebPlaybackSDKToLoad();
   let spotifyPlayer = new Spotify.Player({
      name: "Merger",
      getOAuthToken: (cb: (token: string) => void) => {
         getSpotifyAccessToken().then((token: string) => {
            cb(token);
         }).catch((err) => {
            console.error("failed to obtain the access token! SDK can not
be initialized!", err);
         })
      },
      volume: 0.5,
   });
```

Obrázek 34 - Inicializace Web Playback SDK část 1. Zdroj autor

Pro zprovoznění Web Playback SDK je potřeba zkonstruovat objekt Spotify přehrávače. Konstruktor přijímá objekt typu PlayerInit, který v sobě zahrnuje následující položky:

- Name – Proměnná sloužící k pojmenování momentálně běžícího Spotify přehrávače. Aplikace se objeví pod tímto názvem u jiných zařízení, které právě používají Spotify.
- getOAuthToken – Položka přijímající funkci s callback funkci. Tato funkce musí instanci přehrávače zajistit validní přístupový token k úspěšné inicializaci přehrávače. Pokud není přehrávači poskytnut přístupový token, uživatel nemůže Web Playback SDK používat.
- Volume – Nastaví přehrávači výchozí hlasitost, ve které začne přehrávat hudbu.

Pokud vývojář potřebuje, dle libosti může k instanci přehrávače přidat posluchače, kteří sledují přicházející události. Pokud na jeden z posluchačů je získán určitý typ události, je vykonána vývojářem implementovaná funkce.

```
spotifyPlayer.addListener("initialization_error", ({ message }: { message:
string }) => {
    console.log(message);
});

spotifyPlayer.addListener("authentication_error", ({ message }: { message:
string }) => {
    console.log(message);
});

spotifyPlayer.addListener("account_error", ({ message }: { message: string
}) => {
    console.log(message);
});

spotifyPlayer.addListener("playback_error", ({ message }: { message: string
}) => {
    console.log(message);
});

spotifyPlayer.addListener("account_error", ({ message }: { message: string
}) => {
    console.error("Failed to validate Spotify account", message);
});

spotifyPlayer.addListener('player_state_changed', (state) => {
    spotifyUpdateState(state);
    console.log(state);
});

spotifyPlayer.addListener("ready", ({ device_id }) => {
    dispatcher({ type: ActionTypeDeviceID.SET_DEVICE_ID, payload: device_id
});
    console.log("Ready with Device ID", device_id);
});
```

Obrázek 35 - Inicializace Web Playback SDK část 2. Zdroj autor

Tímto způsobem lze sledovat stav přehrávače a průběh jeho inicializace. Pokud je přehrávač připraven k provozu, vygeneruje se k němu identifikační kód zařízení. Tento identifikační kód je užitečný pro pozdější vykonávání požadavků, pokud je potřeba řídit Spotify přehrávač skrze Spotify API.

Posledním krokem pro zapnutí Spotify přehrávače je funkce connect. Tato funkce propojí instanci přehrávače s Web Playback SDK a zajistí dostupnost uživateli. Pro pozdější manipulaci s přehrávačem si instanci uložíme do proměnné, ke které bude komponent Player a jeho potomci mít přístup.

```
spotifyPlayer.connect().then(
    (res) => res ? console.log("Spotify Connected.") :
console.error("Spotify couldn't connect!")
);

setSDK(spotifyPlayer);
```

Obrázek 36 - Připojení instance přehrávače. Zdroj autor

Postup inicializace YouTube IFrame API je skoro identická k Web Playback SDK. Jedinou odlišností od Web Playback SDK je jeho způsob nasazení do aplikace.

Jelikož je YouTube přehrávač vestavěný jako samostatné okno, je zapotřebí pro něj určit HTML element, který obsadí. V komponentu MainWindow je implementován HTML element typu div s id „youtube-player-window“.

```
let youtubePlayer = YouTubePlayer('youtube-player-window', {
...YoutubeOptions, videoId: videoId });
```

Obrázek 37 - Konstrukce YouTube přehrávače. Zdroj autor

Po vytvoření určeného elementu lze vytvořit novou instanci YouTube Player a inicializovat ho pomocí uvedeného konstruktoru. Konstruktor přijímá jako vstup určité parametry.

- id HTML elementu – Tento element obsadí YouTube přehrávač, který se transformuje do IFrame elementu.
- Objekt možností pro inicializaci – V tomto parametru lze nastavit chování IFrame elementu. Mezi nejčastěji používané patří, výška a šířka IFrame elementu, id přehrávaného videa, autoplay (automatické přehrávání), a mnoho dalších.

```
export const YoutubeOptions: Options = {
    height: '480',
    width: '640',
    playerVars: {
        autoplay: 1,
        enablejsapi: 1,
        controls: 0
    }
};
```

Obrázek 38 - parametry při inicializaci YouTube přehrávače. Zdroj autor

Pokud vývojář potřebuje, lze nad instancí přehrávače zavést posluchače, který pozoruje její změny stavu.

```
youtubePlayer.on("stateChange", async (event) => {
    let duration: number | undefined;
    let progress: number = await window.youtubePlayer.getCurrentTime() *
1000;
    console.log(event.data);

    if (video !== undefined) {
        duration = await
moment.duration(video.contentDetails?.duration).asMilliseconds();
    } else {
        duration = store.getState().state.duration;
    }

    switch (event.data) {
        case -1: {
            store.dispatch({
                type: ActionTypeState.STATE_CHANGE,
                payload: {
                    currentPlayer: Merger.PlayerType.Youtube,
                    ...store.getState().state,
                    ytState: -1
                }
            }
            );
            break;
        }
        case 0:
            mergerNextSong();
            break;
        case 2:
            store.dispatch({
                type: ActionTypeState.STATE_CHANGE, payload: {
                    currentPlayer: Merger.PlayerType.Youtube,
                    ...store.getState().state,
                    paused: true,
                    resuming: false,
                    progressMs: progress,
                    ytState: 2
                }
            });
            break;
        case 1: {
            store.dispatch({
                type: ActionTypeState.STATE_CHANGE, payload: {
                    currentPlayer: Merger.PlayerType.Youtube,
                    ...store.getState().state,
                    paused: false,
                    resuming: true,
                    progressMs: progress,
                    ytState: 1
                }
            });
            break;
        }

        case 3: {
            store.dispatch({
                type: ActionTypeState.STATE_CHANGE, payload: {
                    ...store.getState().state,
                    currentPlayer: Merger.PlayerType.Youtube,
```

Obrázek 39 - zavedení posluchače nad instancí YouTube přehrávače. Zdroj autor

Po inicializaci přehrávače může vývojář uložit instanci YouTube přehrávače na své určené místo. V této aplikaci se instance ukládá do objektu window. Protože je instance umístěna v tomto objektu, vývojář má přístup k instanci v rámci celé aplikace a může nad ní provádět požadované funkce.

```
window.youtubePlayer = youtubePlayer;
    let element = document.getElementById("youtube-player-window");
    if (element !== null) element.style.visibility = "visible";
}

window.youtubePlayer.playVideo();
```

Obrázek 40 - Připojení instance YouTube přehrávače k window objektu. Zdroj autor

8.1.5.3 Manipulace s přehrávači

Instance SDK se nachází ve stavové proměnné v komponentu typu Player. Nad touto proměnnou lze volat určité funkce pro manipulaci s playbackem.

```
const merger State = useAppSelector(root State); // this is used as a
reference!
const dispatcher = useAppDispatch();
const [SDK, setSDK] = useState<Spotify.Player | undefined>();
```

Obrázek 41 - Stavová proměnná SDK. Zdroj autor

V kapitole Ovládací panel 8.1.3.1 byla zmínka o struktuře komponentu Player a jeho potomcích, které manipuluji s přehrávačem Spotify. Mezi tyto potomky patří VolumeSlider, PlayerButton a ProgressBar. Každý z těchto potomku přijímá jako vlastnost funkci.

Pokud uživatel klikne na jeden z těchto komponentů (např. PlayerButton) nebo vykoná určitou akci, akce vyvolá událost, která se propaguje nahoru skrze určené vlastnosti.

```
interface Props {
  text?: string,
  execFunc: Function;
  src?: string,
  width?: number,
  height?: number;
  id: string;
  disabled: boolean;
}

export const PlayerButton: React.FC<Props> = (props: Props) => {

  const handleClick = () => {
    props.execFunc();
  }

  return <button disabled={props.disabled} className="player-button"
id={props.id} onClick={handleClick}>
          <img src={props.src} alt="X"></img>
        </button>;
}
```

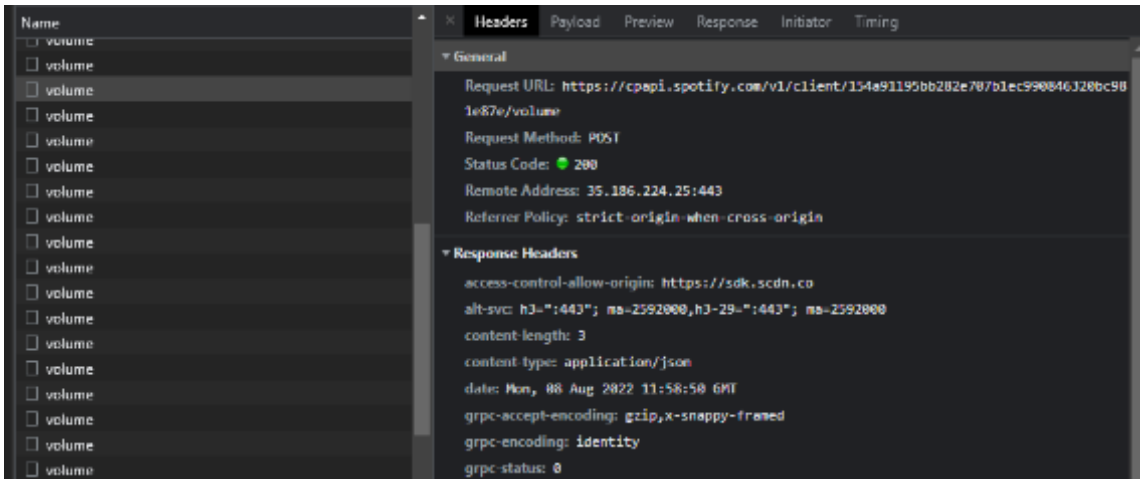
Obrázek 42 – Kód PlayerButton.tsx. Zdroj autor

```
<PlayerButton disabled={!merger State.state.previousSong} id="prev-button"
              src="/images/PrevButton.png" execFunc={merger PrevSong}/>
<PlayerButton disabled={false} src={merger State.state.paused ?
state elmg.play : state elmg.pause}
              text="Toggle Play"
              id="play-button" execFunc={merger TogglePlayBack}/>
<PlayerButton disabled={!merger State.state.nextSong} id="next-button"
              src="/images/NextButton.png" execFunc={merger NextSong}/>
```

Obrázek 43 - Vložené atributy do komponentů typu PlayerButton. Zdroj autor

To znamená, že PlayerButton uvnitř funkce handleClick zavolá funkci v atributu execFunc a tímto způsobem může být vykonána definovaná funkce v komponentu Player. V tomto případě jsou ukázaný funkce pro pozastavení nebo přehrávání playbacku a přechod na další píseň nebo předchozí píseň. Existuje další způsob, jakým můžeme ovlivňovat přehrávání hudby, a to pomocí Spotify API.

Web Playback SDK samostatně neumí zajistit manipulaci přehrávání hudby. Převážně je to sada nástrojů, která umožňuje samotné streamování hudby skrze aplikaci. Pokud se podíváme do DevTools prohlížeče Chrome, můžeme sledovat požadavky, které SDK odesílá (např. při změně hlasitosti).



Obrázek 44 - Reprezentace ovládání SDK v Chrome DevTools. Zdroj autor

Z tohoto vyplývá, že je možné ovládat playback skrz HTTP požadavky, které jsou odesílané Spotify API. Pro vývojáře to přináší jednu velkou výhodu. Vývojář má přístup k manipulaci s playbackem v rámci celé aplikace. Pokud se vývojář vydá tímto přístupem, potřebuje brát v potaz pár omezení:

- Pro úspěšně odeslání požadavku, je potřeba odeslat identifikační kód zařízení, který je přidružený k SDK. Vývojář ve výsledku musí uložit identifikační kód na vhodné místo, ke které bude mít vždy přístup.

- Existuje maximální limit, který omezuje počet odesílaných požadavků za sekundu. Tento limit je velmi často překračován. Pokud uživatel manipuluje s hlasitostí nebo s ukazatelem průběhu, v rámci sekundy je možné odeslat desítky požadavků. U ostatních akcí, je tento limit zanedbatelný.

```
store.dispatch({type: ActionTypeState.STATE_CHANGE, payload: newState});
return
axios.put(`${process.env.REACT_APP_API_LINK}/spotify/player/play?device_id=
${store.getState().deviceId}`, spotify_uris);
```

Obrázek 45 - Odeslání požadavku pro zapnutí přehrávání. Zdroj autor

Pro manipulaci s SDK existují všechny funkce, který vývojář pro ovládání potřebuje. Jako např. pozastavení, pokračování, získání momentálního stavu a další.

```
class Player {
    readonly _options: PlayerInit & {id: string};
    constructor(options: PlayerInit);

    connect(): Promise<boolean>;
    disconnect(): void;
    getCurrentState(): Promise<PlaybackState | null>;
    getVolume(): Promise<number>;
    nextTrack(): Promise<void>;

    addListener: AddListenerFn;
    on: AddListenerFn;

    removeListener(
        event: 'ready' | 'not_ready' | 'player_state_changed' | ErrorTypes,
        cb?: ErrorListener | PlaybackInstanceListener |
PlaybackStateListener,
    ): void;

    pause(): Promise<void>;
    previousTrack(): Promise<void>;
    resume(): Promise<void>;
    seek(pos_ms: number): Promise<void>;
    setName(name: string): Promise<void>;
    setVolume(volume: number): Promise<void>;
    togglePlay(): Promise<void>;
}
```

Obrázek 46 - Reprezentace funkci, které SDK nabízí. Zdroj autor

YouTube IFrame API nabízí mnohem víc funkcí než Web Playback SDK, avšak hlavním rozdílem je absence manipulace s přehráváním pomocí HTTP požadavků. Vývojář musí ovládat přehrávání jenom skrz jeho instanci.

8.1.5.4 Reprezentace stavu přehrávačů

Oba přehrávače, které jsou v aplikaci integrované, v sobě udržují určitý stav. Toto je velmi zásadní pro zajištění správného chodu přehrávačů. Existují dva způsoby, jak získat stav daného přehrávače:

- Pomocí posluchače, který přijímá události týkající se změny stavu.
- Pomocí funkcí nad instancemi, jež vrací jejich momentální stav.

Jakým způsobem je u obou přehrávačů reprezentován se velice liší.

Spotify přehrávač reprezentuje svůj stav mnoha vlastnostmi jako například:

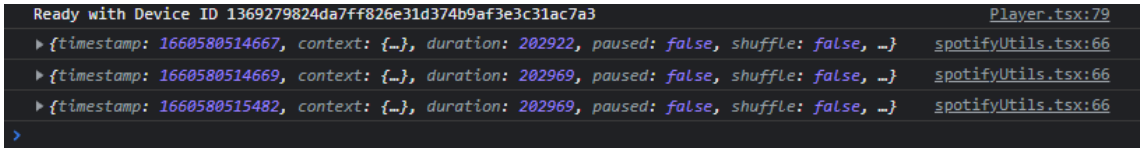
- PlaybackContext – Tato vlastnost v sobě obsahuje určitá metadata o SDK.
- Paused – Boolean proměnná reprezentující pozastavený nebo přehrávající stav.
- Track_window – Obsahuje data o momentálně přehrávané písničce.
- Position – Zobrazuje časovou pozici písničky, která je přehrávána.
- A mnoho dalších…

```
interface PlaybackState {
    context: PlaybackContext;
    disallows: PlaybackDisallows;
    duration: number;
    paused: boolean;
    position: number;
    loading: boolean;
    /**
     * 0: NO_REPEAT
     * 1: ONCE_REPEAT
     * 2: FULL_REPEAT
     */
    repeat_mode: 0 | 1 | 2;
    shuffle: boolean;
    restrictions: PlaybackRestrictions;
    track_window: PlaybackTrackWindow;
}
```

Obrázek 47 - reprezentace objektu PlayerState. Zdroj autor

Web Playback SDK také automaticky v určitých intervalech notifikuje posluchače o svém stavu. Touto funkcí lze zajistit, že průběh přehrávání je synchronizovaný s naší aplikací.

Na druhou stranu tu vyvstává problém. Pokud Spotify aktualizuje svůj stav, může přijít nadbytečné množství události o změně. Toto může v aplikaci vyvolávat nepředvídatelné a nechtěné chování, pokud se nimi aplikace řídí jako např. u automatického nasazení dalšího nebo předchozího písně.



Obrázek 48 - Počet změn stavu při zapnutí přehrávání. Zdroj autor

YouTube IFrame API a jeho posluchač na změnu stavu získává pouze enumerátor, který reprezentuje určitý stav. Enumerátor se skládá ze sedmi hodnot:

- BUFFERING – Tento stav informuje vývojáře, že přehrávané video se načítá.
- ENDED – Přehrávání videa bylo ukončeno.
- PAUSED – Video bylo pozastaveno.
- PLAYING – Momentálně probíhá přehrávání videa.
- UNSTARTED – YouTube přehrávač se načítá.
- VIDEO_CUED – Video bylo nahráno do přehrávače.

```
declare enum PlayerStates {
    BUFFERING = 3,
    ENDED = 0,
    PAUSED = 2,
    PLAYING = 1,
    UNSTARTED = -1,
    VIDEO_CUED = 5,
}
```

Obrázek 49 - Reprezentace stavu YouTube přehrávače. Zdroj autor

Pokud vývojář chce získat jiné informace, jako např. délku videa, id momentálně přehrávaného videa a další, musí použít příslušné funkce instance YouTube IFrame API.

8.1.6 Management vlastního stavu

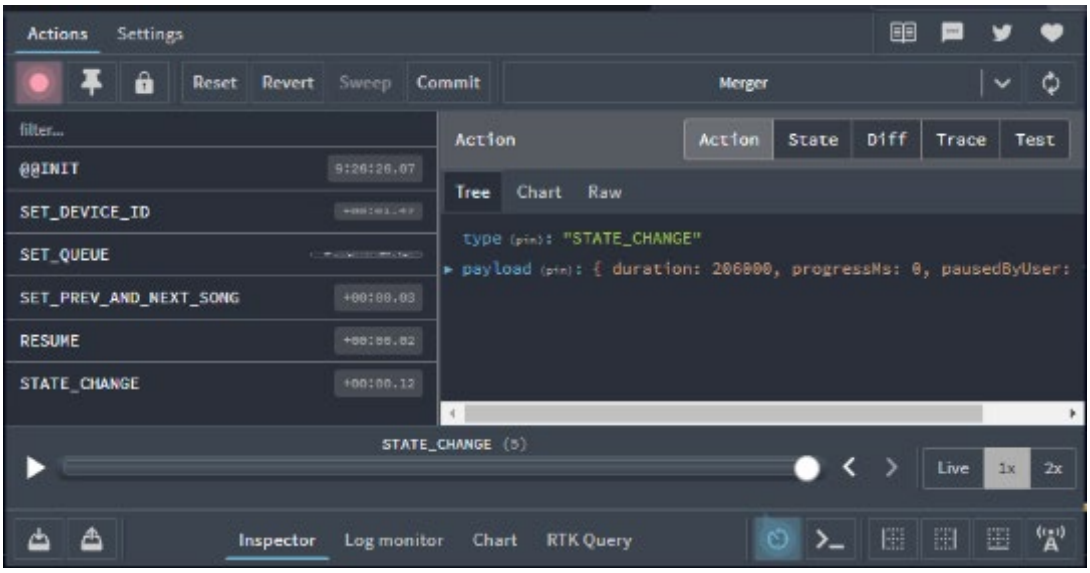
Pro účely bakalářské práce je potřeba mezi těmito přehrávači udržovat svůj vlastní stav. Pokud by nebyl implementován, nedocházelo by k automatizaci přepínání mezi přehrávači. Dalším požadavkem je zajistit přístup ke stavu aplikace z jakéhokoliv místa, což nám zajistí globální stav. Pro implementaci globálního stavu byla vybrána knihovna React Redux.

8.1.6.1 React redux

React Redux je obalem existující knihovny Redux, která slouží pro management globálního stavu JavaScriptové aplikace. Tím, že je React Redux implementovaný jako obal, lze ho jednoduše integrovat a spojit s jmenovanou knihovnou React. Pro jednodušší manipulaci obsahuje svoje API vystavěný Reactu. React oficiálně také nabízí funkce neboli hooky pro implementaci globálního stavu, které nazývá React Context. Oproti jejich React Contextu je React Redux přívětivější z následujících důvodů:

- Předvídatelnost – Pokud sledujeme chování React Contextu, může se stát, že jeho chování je nekonzistentní. Zápisy se provádějí asynchronně, a to pro účely této aplikace není vhodné. React Redux právě tento problém eliminuje tím, že svůj stav aktualizuje synchronně.[12]

- Možnost ladění – React Context neobsahuje nástroje, které umožní vývojáři jednoduché ladění globálního stavu. React Redux tuto funkcionalitu doplňuje pomocí Redux DevTools. Tento nástroj, který se instaluje do prohlížeče, umožňuje vývojáři zaznamenávat změny stavu a jednotlivé akce, které způsobili určité změny.



Obrázek 50 - Ukázka Redux Toolkit. Zdroj autor

8.1.6.1.1 Základy Reduxu

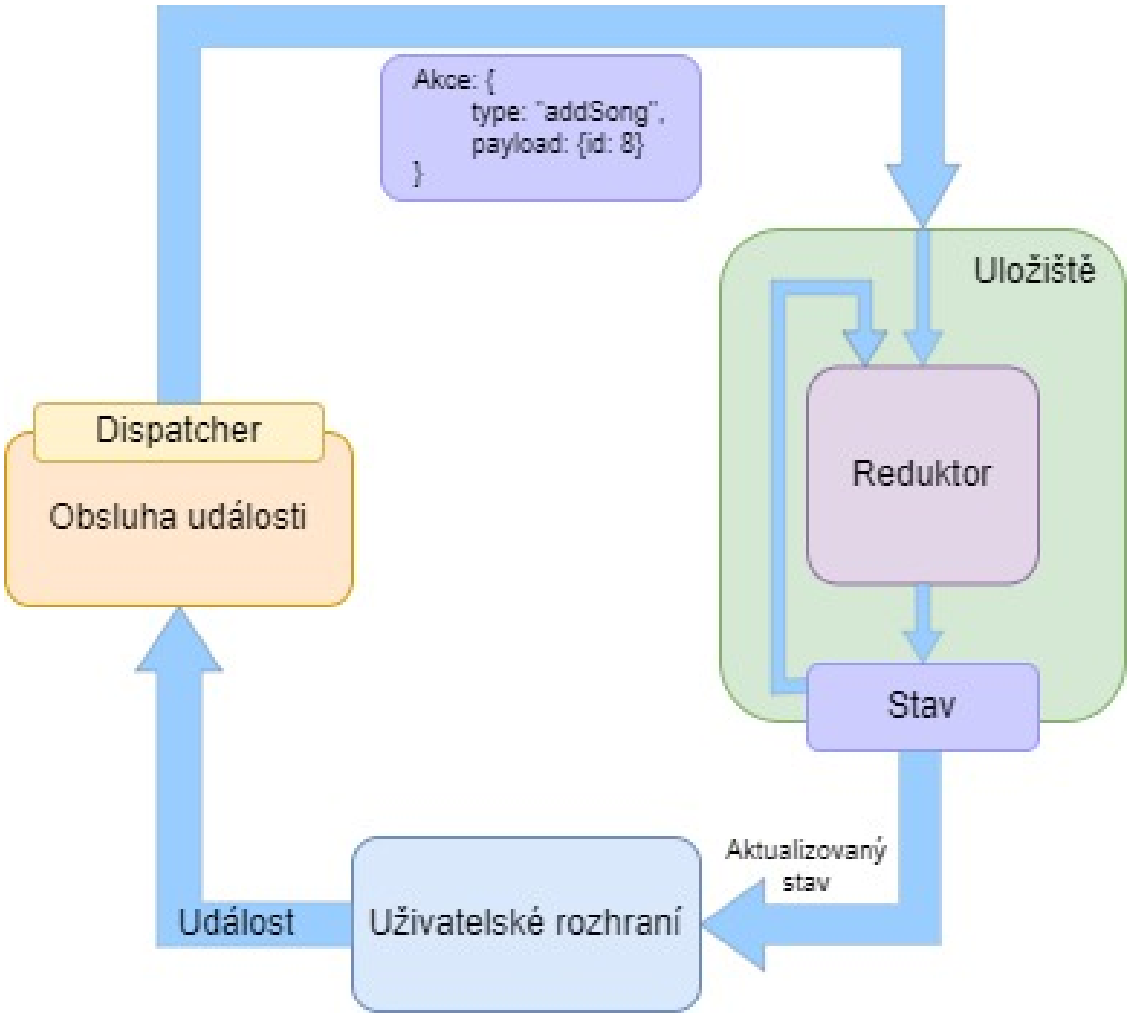
Jádro React Reduxu pro management stavu aplikace vytváří jeho uložiště. Uložiště slouží jako kontejner, který v sobě udržuje globální stav aplikace. Chová se jako JavaScript objekt se speciálními funkcemi a schopnostmi. Pro správně fungování, vývojář musí dodržet určitá pravidla:

- Vývojář nikdy nesmí modifikovat nebo měnit globální stav Reduxu v úložišti napřímo.
- V Reduxu se nesmí vykonávat žádná asynchronní logika (K tomu slouží tzv. Middlewary).

Pokud vývojář chce změnit globální stav aplikace, musí být určen jiný způsob provedení. Pro aktualizaci stavu je potřeba si ujasnit datový tok React Reduxu.

8.1.6.1.1.1 Datový Tok Reduxu

Redux oproti React Contextu zaujímá velmi odlišný přístup. Datový tok Reduxu se může shrnout pomocí následujícího diagramu.



Obrázek 51 - Reprezentace datového toku v Reduxu. Zdroj autor

Průběh změny stavu začíná v uživatelském rozhrání. Uživatel v rozhraní provede určitou akci, která vyvolá událost. Tato událost je obsloužena odesílatelem (dispatcherem), který se postará o vytvoření nového objektu akce. Akce slouží jako JSON objekt, jehož vlastnostmi popisuje, o jaký typ akce se jedná, případně dodává užitečná data potřeba k aktualizaci stavu. Akce následně putuje k úložišti, které danou akci přijme a zpracuje. Část úložiště starající se o změnu stavu se nazývá „reduktor“.

Reduktor na svůj vstup získává příslušnou akci a předchozí stav objektu. Reduktor musí dodržovat určitá pravidla při aktualizaci globálního stavu:

- Nový stav musí být vypočítaný jen dle přicházející akce a předešlého stavu.
- Reduktor nesmí stav modifikovat napřímo. Pokud reduktor chce modifikovat stav, lze ho měnit jenom na kopii předešlého stavu.

Poté co reduktor aktualizuje stav uložiště, nový stav je promítnut zpátky do uživatelské rozhraní. Objekty, které lze umístit do uložiště, musí být serializovatelné (přeložitelné do takového formátu, který je vhodný pro uložení do paměti). To znamená, že ve stavu mohou být uložený jenom JSON objekty.

Existují situace, kdy vývojář chce vykonat určitou funkci při změně stavu v úložišti. Redux nabízí nad uložištěm funkci subscribe, jejíž parametrem je libovolná funkce.

```
const updateType = () => {
    setCurrentPlayer(store.getState().state.currentPlayer)
}

store.subscribe(updateType)
```

Obrázek 52 - Aktualizace typu momentálně hrajícího přehrávače pomocí odběru. Zdroj autor

8.1.6.2 Implementace React Reduxu v aplikaci

Před implementací globálního stavu do aplikace je zapotřebí definovat z jakých vlastností se stav bude skládat. Převážně se struktura stavu bude odvíjet od přehrávačů. Navrch k tomuto stavu je potřeba přidat další vlastnosti, které jsou důležité pro správný chod aplikace jako např.:

- CurrentPlayer – Tato proměnná uchovává v sobě hodnotu enumerátoru, který určuje momentálně hrající přehrávač.
- YTState – Tato proměnná si ukládá stav přehrávače YouTube IFrame API. Momentálně je implementovaná pro účely ladění aplikace.
- PausedByUser – Tato boolean proměnná označuje, zda přehrávač byl pozastavený uživatelem. Slouží jako pomocná proměnná při přepínání mezi jinými přehrávači.

Po implementaci definice je nyní možné se zaměřit na implementaci React Redux.

```
export interface PlayerState {
    currentPlayer?: PlayerType.Spotify | PlayerType.Youtube,
    paused: boolean,
    previousSong?: gapi.client.youtube.Video | SpotifyApi.TrackObjectFull,
    currentSong?: gapi.client.youtube.Video | SpotifyApi.TrackObjectFull,
    nextSong?: gapi.client.youtube.Video | SpotifyApi.TrackObjectFull,
    progressMs?: number,
    duration?: number,
    pausedByUser?: boolean,
    ytState?: number
}
```

Obrázek 53 - Definice PlayerState. Zdroj autor

Objekt potřebný pro údržbu globálního stavu aplikace je implementován v souboru combineReducers.tsx následovně:

- Device id – Tato položka udržuje hodnotu id zařízení, které je získáno od Web Playback SDK.
- State – Vlastnost typu PlayerState, dle které se řídí celá aplikace.
- Queue – Objekt obsahující pole písniček, které jsou nasazeny k přehrávání, a počítadlo ukazující na pozici v poli.

```
export const rootReducer = combineReducers({
    deviceId: deviceIDReducer,
    state: stateReducer,
    queue: queueReducer,
})
```

Obrázek 54 - Objekt reprezentující stav aplikace. Zdroj autor

V obrázku lze pozorovat jakým způsobem je údržba stavu implementována. Na vrchu Reduxu je implementován tzv. kořenový reduktor, který se stará o dílčí reduktory. Tedy každá proměnná v objektu má deklarovaný svůj reduktor starající se o určitou proměnnou. Pokud chceme zkombinovat dílčí reduktory do jednoho, Redux nabízí funkci combineReducers. Touto funkcí se zkonstruuje nový kořenový reduktor.

```
export enum ActionTypeDeviceID {
    SET_DEVICE_ID = "SET_DEVICE_ID"
}

interface ActionSetDeviceId {
    type: ActionTypeDeviceID.SET_DEVICE_ID,
    payload: string
}

const initState: string = "";

export const deviceIDReducer = (state: string = initState, action:
ActionSetDeviceId): string => {
    switch (action.type) {
        case ActionTypeDeviceID.SET_DEVICE_ID: return action.payload;
        default: return state;
    }
}
```

Obrázek 55 - Implementace reduktoru pro id zařízení v deviceIdSlice.tsx. Zdroj autor

Příkladem dílčího reduktoru je např. deviceIdReducer. Dle předchozí teorie lze v kódu pozorovat jeho parametry, které přijímá. Jsou jimi akce typu ActionSetDeviceId a předešlý stav typu textového řetězce. Po příjmu akce a předešlého stavu se zavolá tento reduktor a testuje se, o jaký typ akce se jedná. Dle typu vykoná implementovaný kód. Po vypočítání nového stavu reduktor musí vrátit zpátky aktualizovaný stav dle jeho návratového typu. Tímto způsobem se hodnota zapíše a propaguje nahoru. Ve stejném souboru lze vyjádřit typy jednotlivých akcí a jejich parametry. Ačkoliv je toto místo pro vyjádření typu příznivější a přehlednější, vývojáři nebrání si je implementovat v jiném souboru.

8.1.6.3 Fungování React Reduxu v praxi

Jakým způsobem React Redux funguje v aplikaci lze jednoduše demonstrovat při spuštění písničky.

```
export const spotifyPlay = async (spotify_uris?: string[]): Promise<void>
=> {
    if (window.Spotify === undefined) throw new
Error(spotifyIsUndefinedError);

    if (spotify_uris !== undefined) {
        let track: SpotifyApi.TrackObjectFull = await
getSpotifyTrack(getTrackUniqueId(spotify_uris[0]));

        let currentState: Merger.PlayerState = store.getState().state;

        let newState: Merger.PlayerState = {
            duration: track.duration_ms,
            progressMs: 0,
            pausedByUser: false,
            paused: false,
            currentSong: track,
            nextSong: currentState.nextSong,
            previousSong: currentState.previousSong,
            currentPlayer: Merger.PlayerType.Spotify
        };

        store.dispatch({type: ActionTypeState.STATE_CHANGE, payload:
newState});
        return
axios.put(`${process.env.REACT_APP_API_LINK}/spotify/player/play?device_id=
${store.getState().deviceId}`, spotify_uris);
    }
    store.dispatch({type: ActionTypeState.PLAY_BY_USER});
    return
axios.put(`${process.env.REACT_APP_API_LINK}/spotify/player/play?device_id=
${store.getState().deviceId}`);
};
```

Obrázek 56 -Implementace změny stavu při zapnutí přehrávání Spotify. Zdroj autor

Pokud uživatel vyvolá v uživatelském rozhrání událost, která zavolá funkci spotifyPlay, získá se pomocí URI píseň a posléze jsou její data předána do nového objektu stavu. V předposledním řádku je tento nový stav přiřazen objektu akce a uložiště poté odešle akci typu STATE_CHANGE.

```
export const stateReducer = (state: Merger.PlayerState = initState, action:
Action): Merger.PlayerState => {
    switch (action.type) {
        case ActionTypeState.STATE_CHANGE: {
            return {
                ...state,
                currentPlayer: action.payload.currentPlayer,
                paused: action.payload.paused,
                previousSong: action.payload.previousSong,
                currentSong: action.payload.currentSong,
                nextSong: action.payload.nextSong,
                progressMs: action.payload.progressMs,
                duration: action.payload.duration,
                pausedByUser: action.payload.pausedByUser,
                ytState: action.payload.ytState
            }
        }
```

Obrázek 57 - Reduktor pro změnu stavu typu PlayerState. Zdroj autor

Po odeslání akce uložiště zavolá reduktor stateReducer, který se postará o aktualizaci stavu typu PlayerState. Po provedení změn jsou odběratelé uložiště upozorněni a tím vykonají své funkce.

Vývojáři je také umožněno použití tzv. „selektorů“. Tyto selektory slouží pro výběr určitých položek z globálního stavu. Při použití selektoru se využívá React hook useAppSelector. Tento hook přijímá jako parametr funkci, jehož úkolem je získat celý globální stav z uložiště a vybrat z něho takové položky, které jsou potřeba. Tuto funkci si musí vývojář implementovat sám.

```
export const rootState = (state: RootState) => state;

export type RootState = ReturnType<typeof rootReducer>;
```

Obrázek 58 - Implementace funkce pro získání stavu pro useAppSelector. Zdroj autor

```
export const QueuePage: React.FC = () => {

    const mergerQueue: Merger.Queue = useAppSelector(rootState).queue;
```

Obrázek 59 - Příklad použití React hooku useAppSelector. Zdroj autor

8.1.7 Získávání a zobrazení obsahu Spotify a YouTube

8.1.7.1 Vyhledávání

K tomu, aby uživatel mohl spustit jakoukoliv píseň z databáze Spotify nebo YouTube, je potřeba vytvořit novou stránku pro vyhledávání. Pro vykonání této funkcionality budou sloužit nové komponenty typu SpotifySearchWindow a YoutubeSearchWindow. Základní stavební prvek, které tyto dva komponenty sdílejí je jejich vyhledávácí pole.

Obrázek 60 - Vyhledávácí pole pro Spotify. Zdroj autor

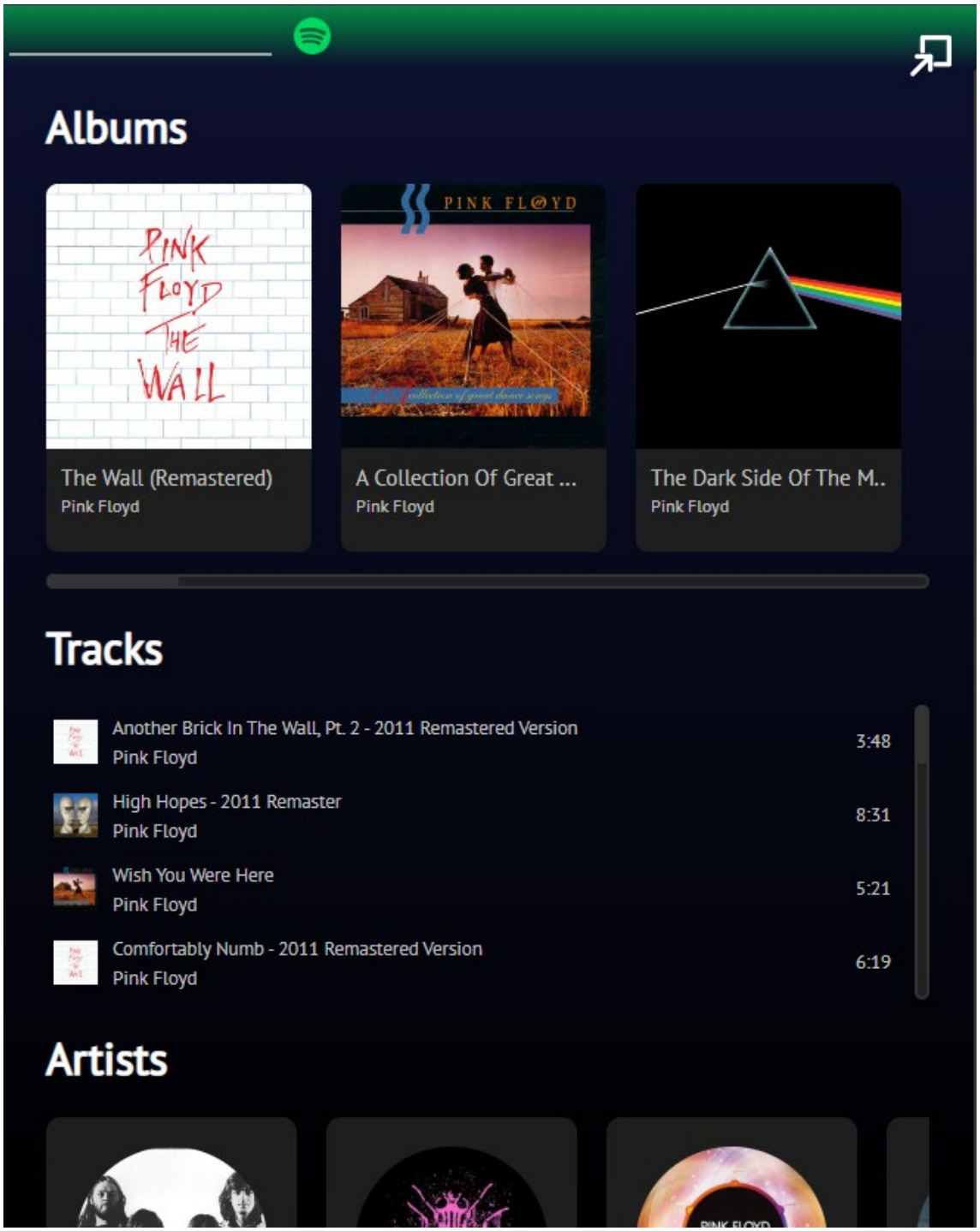
Vyhledávací pole neboli komponent SearchBar se skládá ze samotného textového pole a ikony, která funguje jako navigační ikona pro přepínání mezi jednotlivými typy vyhledávání. V obrázku lze vidět v pozadí zelenou barvu, která naznačuje uživateli, že vyhledává u Spotify. Pokud uživatel klikne na ikonu Spotify, přemění se ikona na logo Youtube a pozadí změní barvu na červenou.

Obrázek 61 - Vyhledávácí pole pro YouTube. Zdroj autor

Pokud uživatel zadá do textového pole textový řetězec, po určitém zpoždění je odeslán požadavek o vyhledávání a uživateli jsou vráceny výsledky. Zpoždění je implementováno z důvodu ušetření zdrojů. Pokud by zpoždění neexistovalo, pří každém zadání nového písmene by se odesílal nový požadavek.

Jak bylo zmíněno při návrhu, u Spotify jsou výsledky rozdělený do skupin dle jejich typu (umělci, skladby, seznam skladeb a alba) a u YouTube je každý výsledek reprezentován řádkem. U Spotify každý výsledek alba, seznam skladby nebo umělce naviguje na další stránku, která zahrnuje specifičtější obsah dle výběru. Pokud uživatel klikne na jednu z deseti stop v kolekci skladeb, zahájí se spuštění přehrávače.

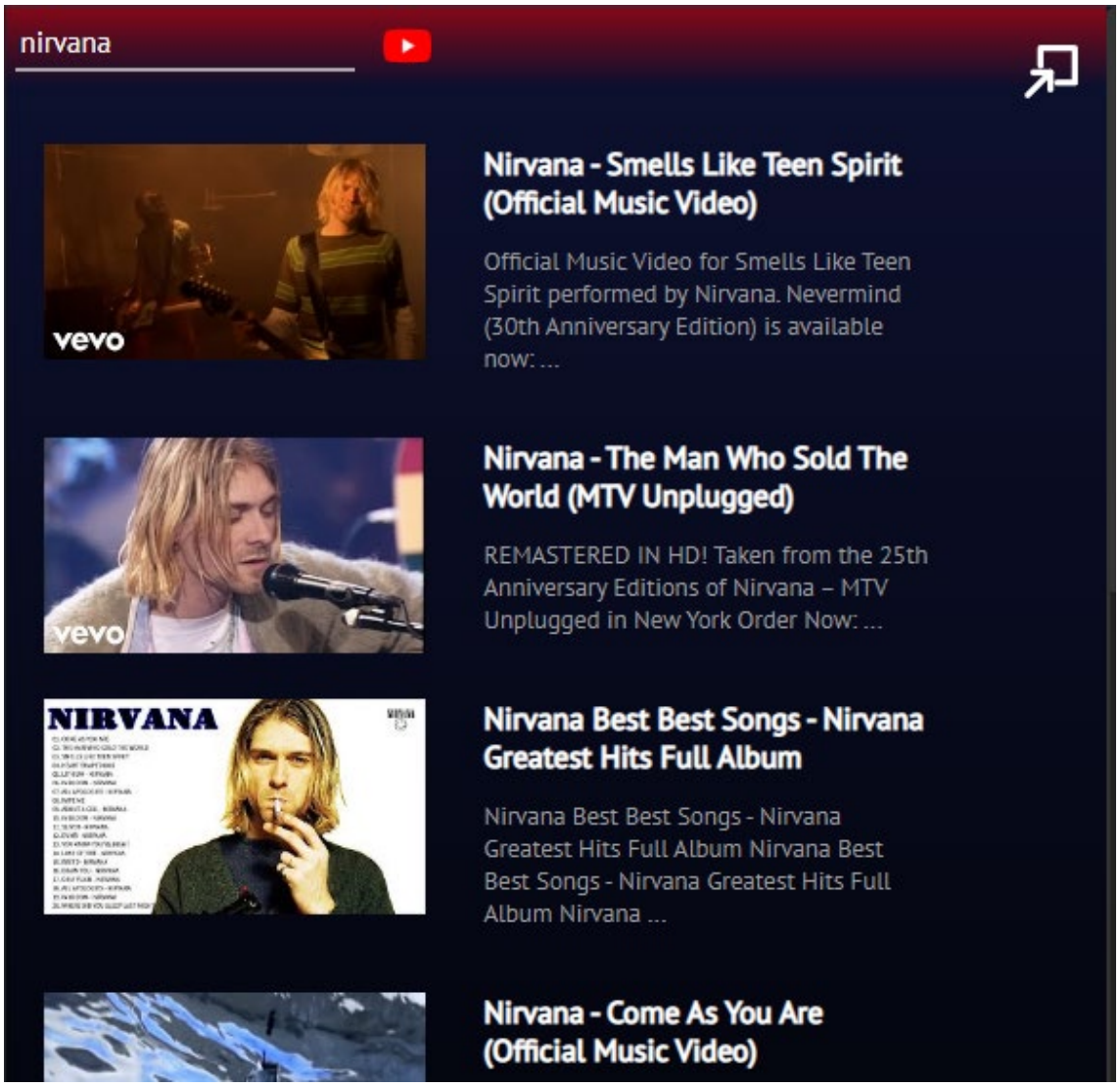
Na okně YouTube vyhledávání lze kliknout na jakoukoliv z výsledku a video je automaticky spuštěno.



Obrázek 62 - Reprezentace výsledku při vyhledávání u Spotify. Zdroj autor

```
return (
    <div id="spotify-search-window">
        <SearchBar type={Merger.PlayerType.Spotify}
func={handleSearch}></SearchBar>
        {

            results !== null &&
            <div id="search-result-container">
                <h1>Albums</h1>
                {(results.albums && results.albums?.items.length > 0) &&
                    <AlbumCollection albums={results.albums.items}/>}

                <h1>Tracks</h1>
                {(results.tracks && results.tracks?.items.length > 0) &&
                    <TrackCollection tracks={results.tracks.items}/>}

                <h1>Artists</h1>
                {(results.artists && results.artists.items.length > 0) &&
                    <ArtistCollection artists={results.artists?.items}/>}

                <h1>Playlists</h1>
                {(results.playlists && results.playlists?.items.length > 0) &&
                    <PlaylistCollection playlists={results.playlists.items}/>}
            </div>
        }
    </div>
)
```

Obrázek 63 - Kód pro zobrazení kolekcí výsledků u Spotify. Zdroj autor



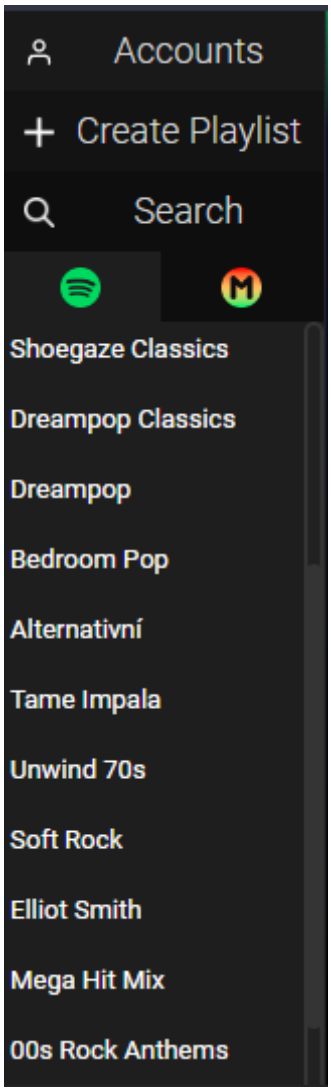
Obrázek 64 - Reprezentace výsledků vyhledávání YouTube. Zdroj autor

```
const generateResults = (result: gapi.client.youtube.SearchResult):
JSX.Element | null => {
    if (result.id?.videoId) return <YoutubeVideoSearchResult
            playVideo={playVideo} key={result.id?.videoId}
            item={result}/>
    if (result.id?.playlistId) return (<YoutubePlaylistSearchResult
playlistId={result.id.playlistId}
key={result.id?.playlistId}
img={result.snippet?.thumbnails?.default?.url}
title={result.snippet?.title}/>)

    return null;
 }

 return (
    <div id="youtube-search-window">
        <SearchBar type={Merger.PlayerType.Youtube} func={(value: string)
=> handleSearch(value)}></SearchBar>
        {results !== null &&
            <div id="search-result-container">
                {results.items?.map(generateResults)}
            </div>
        }
    </div>
 )
```

Obrázek 65 - Implementace zobrazení výsledků u YouTube. Zdroj autor

8.1.7.2 Výpis seznamů skladeb Uživatele

Seznamy skladeb jsou zobrazeny v levém bočním panelu. Tento komponent v horní části zahrnuje karty s ikonami pro přepínání mezi seznamy Spotify a této aplikace. Seznamy skladeb jsou vypsány jen v momentě, kdy je uživatel přihlášen.



Obrázek 66 - Výpis seznamů skladeb u Spotify. Zdroj autor

Tento boční panel obsahuje odkazy, které navigují uživatele na samostatné stránky těchto seznamů skladeb.

```
export const Playlists: React.FC = () => {

    const [index, setIndex] = useState<number>(0);

    const backgroundColor: CSSProperties = {backgroundColor: '#1e1e1e'};

    return (
   <div id="playlists">
            <div id="tabs">
                <div style={index === 0 ? backgroundColor : {}} onClick={()
=> setIndex(0)}>
                    <img src="/images/spotify.png" alt="couldn't load"/>
                </div>
                <div style={index === 1 ? backgroundColor : {}} onClick={()
=> setIndex(1)}>
                    <img  src="/images/mergericon.png" alt="couldn't
load"/>
                </div>
            </div>
            <SpotifyPlaylists display={index === 0 ? 'flex' : 'none'}/>
            <MergerPlaylist display={index === 1 ? 'flex' : 'none'} />

        </div>
    )
}
```

Obrázek 67 - Implementace komponentu Playlists. Zdroj autor

V kódu lze pozorovat dva komponenty typu SpotifyPlaylists a MergerPlaylists. Mezi těmito komponenty se přepíná pomocí karet a každý komponent má v sobě uložené seznamy skladeb získané ze serverů.

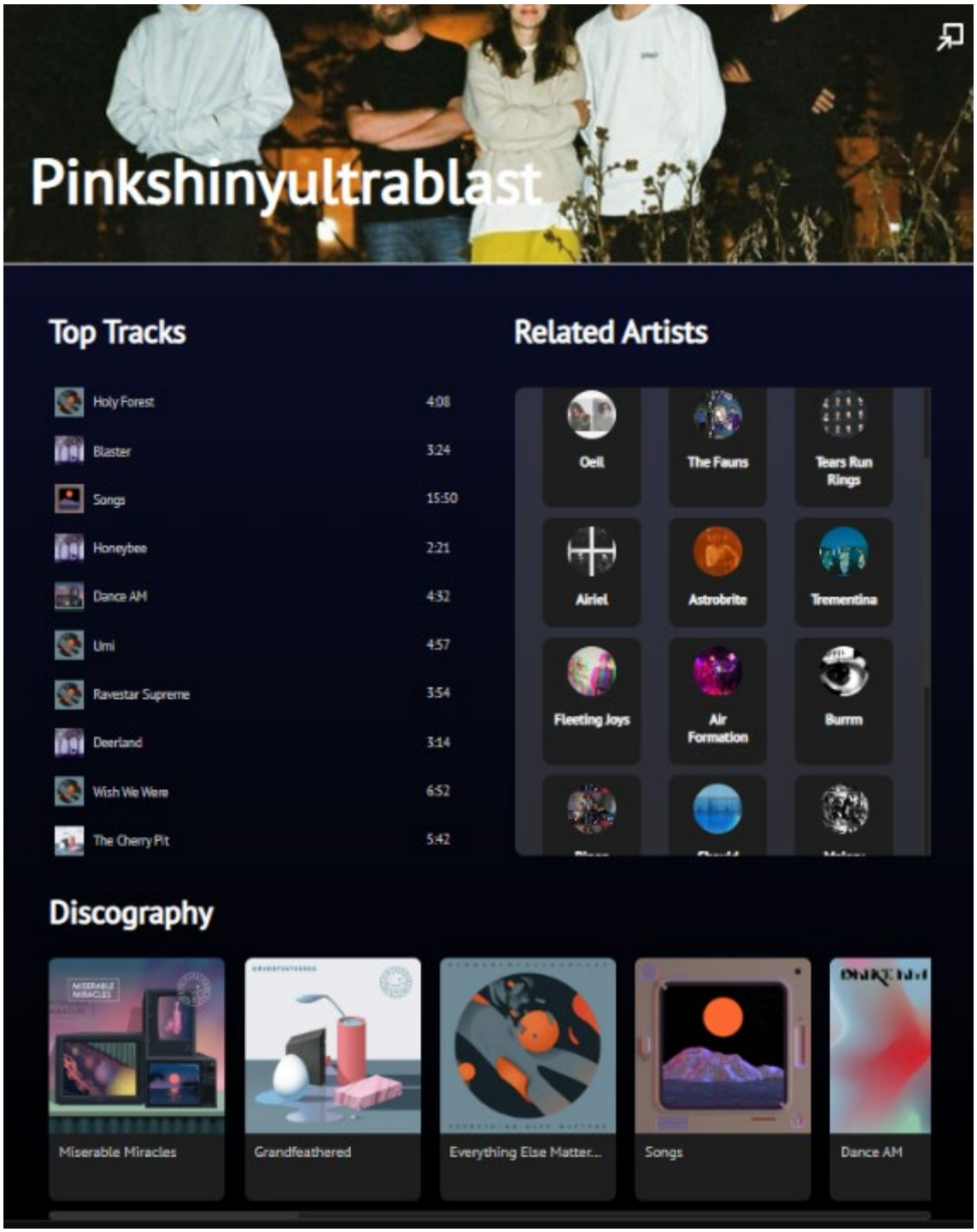
8.1.7.3 Stránky pro album, seznam skladeb a umělců

Uživatel muže navigovat k těmto stránkám neboli komponentům skrze předešlé bvyhledávání u Spotify. Jejich logika pro stažení dat jsou principiálně identické. Poprvé co se komponent načte, pomocí React hooku useParams se získá z URL parametr id a poté lze stáhnout potřebná data k zobrazení stránky.

Obrázek 68 - Zobrazení Spotify alba. Zdroj autor

```
export const SpotifyAlbumPage: React.FC = () => {

   const { id } = useParams<{ id: string | undefined }>();
   const [album, setAlbum] = useState<SpotifyApi.AlbumObjectFull>();
   const [tracks, setTracks] =
useState<Array<SpotifyApi.TrackObjectFull>>();

   const loadAlbum = async () => {
      if (!id) return console.error("Can't load, id is undefined!");

      try {
         let album = await axios.get<SpotifyApi.AlbumObjectFull>(
            `${process.env.REACT_APP_API_LINK}/spotify/albums/${id}`
         );

         setAlbum(album.data);

         let tracks = await axios.get<Array<SpotifyApi.TrackObjectFull>>(
            `${process.env.REACT_APP_API_LINK}/spotify/albums/${id}/tracks`
         );

         setTracks(tracks.data);

      } catch (e: unknown) {
         console.error(e);
      }

   }
   const handlePlay = (track: SpotifyApi.TrackObjectSimplified) => {
      if (tracks) addOtherSongsToQueuePlaylist(track.uri, tracks);
      mergerLoadAndPlay(track);
   }

   useEffect(() => {
      loadAlbum();
   }, [id])
```

Obrázek 69 - Logika pro získávání alba u komponentu SpotifyAlbumPage. Zdroj autor



Obrázek 70 - Rozložení stránky pro Spotify umělce

```
export const SpotifyArtistPage: React.FC = () => {

   const { id } = useParams<{ id: string | undefined }>();
   const [artist, setArtist] = useState<SpotifyApi.ArtistObjectFull>();
   const [topTracks, setTopTracks] =
useState<SpotifyApi.ArtistsTopTracksResponse>();
   const [albums, setAlbums] =
useState<SpotifyApi.ArtistsAlbumsResponse>();
   const [relatedArtists, setRelatedArtists] =
useState<SpotifyApi.ArtistsRelatedArtistsResponse>();

   const loadArtist = async () => {

      if (!id) return console.error("Can't load an artist! ID is
undefined!");

      try {
         let artist = axios.get<SpotifyApi.ArtistObjectFull>(
            `${process.env.REACT_APP_API_LINK}/spotify/artist/${id}`
         );

         setArtist((await artist).data);

         let topTracks = axios.get<SpotifyApi.ArtistsTopTracksResponse>(

`${process.env.REACT_APP_API_LINK}/spotify/artist/${id}/topTracks`
         );

         setTopTracks((await topTracks).data);

         let albums = axios.get<SpotifyApi.ArtistsAlbumsResponse>(
            `${process.env.REACT_APP_API_LINK}/spotify/artist/${id}/albums`
         );

         setAlbums((await albums).data);

         let relatedArtists =
axios.get<SpotifyApi.ArtistsRelatedArtistsResponse>(

`${process.env.REACT_APP_API_LINK}/spotify/artist/${id}/relatedArtists`
         );

         setRelatedArtists((await relatedArtists).data);
      } catch (e: unknown) {
         console.error(e);
      }
   }

   useEffect(() => {
      loadArtist()
   },[])
```

Obrázek 71 - Kód pro načítání umělce v souboru SpotifyArtistPage.tsx

8.1.8 Ovládání implementovaného přehrávače

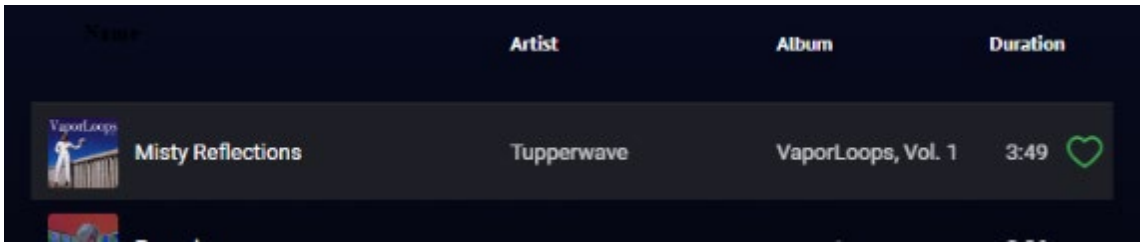
8.1.8.1 Spouštění nové písně

Ke spuštění určené skladby, která může být ze Spotify nebo YouTube, je potřeba implementovat vlastní funkci, která se stará o rozlišení skladby a jeho původního typu přehrávače. K tomuto slouží funkce mergerLoadAndPlay.

```
export const mergerLoadAndPlay = async (track: SpotifyApi.TrackObjectFull |
gapi.client.youtube.Video | gapi.client.youtube.ResourceId | string) => {

    let state: Merger.PlayerState = store.getState().state;

    store.dispatch({ type: ActionTypeState.RESUME })

    if (isSpotifyTrackObject(track)) {

        if (state.currentPlayer === Merger.PlayerType.Youtube)
            window.youtubePlayer.stopVideo();

        if (!window.Spotify) throw new Error(initializationError);

        spotifyPlay([track.uri]);

        return;
    }
    if (state.currentPlayer === Merger.PlayerType.Spotify)

axios.put(`${process.env.REACT_APP_API_LINK}/spotify/player/pause?device_id
=${store.getState().deviceId}`);

    if (isYoutubeVideo(track))
        return youtubePlay(track.id);

    return youtubePlay(track);
}
```

Obrázek 72 - Implementace funkce pro spouštění skladby. Zdroj autor

Tato funkce v prvním kroku získá momentální stav aplikace. Poté se nad uložištěm Reduxu odešle nová akce typu RESUME, která se stará o změnu jedné položky v globálním stavu. Tato položka informuje aplikaci, zda je přehrávač pozastaven nebo ne. V tomto případě je nastaven na hodnotu true. Dalším krokem je rozlišení TypeScriptového typu získané skladby. Skladba může být reprezentována čtyřmi typy a to videem, skladbou od Spotify, ResourceId (tento objekt může být získány z výsledku vyhledávání), anebo textovým řetězcem. Po rozlišení skladby je před spuštěním ověřeno, zda momentálně nehraje jiný přehrávač. Toto je ověřené dle položky currentPlayer získané z globálního stavu. Pokud ano, momentálně hrající přehrávač je pozastaven a následně může být nová písnička nasazena. Písničku lze spustit ze seznamu skladeb, z alba, vyhledávání nebo top skladeb umělce.



Obrázek 73 - Zobrazení Spotify skladby v seznamu skladeb. Zdroj autor

8.1.8.2 Přeskok na další nebo předchozí píseň

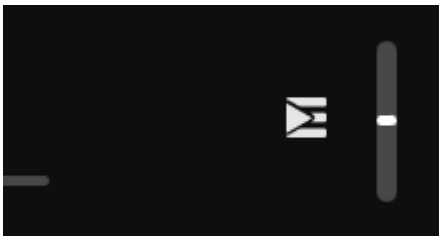
Tyto funkce jsou implementovány podobným způsobem jako mergerLoadAndPlay. Po nasazení nové skladby je dále potřeba aktualizovat v globálním stavu počítadlo fronty a definovat novou předchozí a následující skladbu.

```
export const mergerPrevSong = async () => {
    let state: Merger.PlayerState = store.getState().state;
    if (state.previousSong !== undefined) {

        if (isSpotifyTrackObjectFull(state.previousSong)) {
            if (!state.paused && state.currentPlayer ===
Merger.PlayerType.Youtube)
                youtubePause();
            spotifyPlay([state.previousSong.uri]);
        } else {
            if (!state.paused && state.currentPlayer ===
Merger.PlayerType.Spotify)
                spotifyPause();
            youtubePlay(state.previousSong.id);
        }

        /*This needs to be done in order to prevent unexpected behaviour, if
I really depend on getting the value straight
        from the state, the value can't be guaranteed to be updated right
away*/
        let oldIndex: number = store.getState().queue.counter - 1;
        store.dispatch({ type: ActionTypeQueue.DEC_QUEUE_COUNTER });

        let queue: Array<SpotifyApi.TrackObjectFull |
gapi.client.youtube.Video> = store.getState().queue.queue;

        store.dispatch({
            type: ActionTypeState.SET_PREV_AND_NEXT_SONG, payload: {
                previous: queue[oldIndex - 1],
                next: queue[1 + oldIndex]
            }
        })
    }
}
```

Obrázek 74 - Implementace funkce pro nasazení předchozí skladby. Zdroj autor

8.1.8.3 Ovládání hlasitosti

Posuvník hlasitosti se nachází na pravé straně ovládacího panelu. Jak již bylo zmíněno, tento komponent VolumeSlider je potomkem komponentu typu Player. Pokud uživatel posouvá s hlasitostí, komponent VolumeSlider zaznamená tuto událost a zavolá funkci, která je definovaná skrze jeho atribut func. Tudíž v komponentu Player je poté zavolána v atributu func funkce setVolume, který získá novou hodnotu a tou nastaví hlasitost momentálně hrajícího přehrávače.



Obrázek 75 - Zobrazení posuvníku hlasitosti. Zdroj autor

```
const VolumeSlider: React.FC<Props> = ({func, isDisabled} : Props) => {
    let [value, setValue] = useState<number>(50);

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        setValue(e.target.valueAsNumber);
        if (func) {
            if (value > 100) return func(100);
            if (value < 0) return func(0);

            return func(value);
        }
    }

    return (
        <div id="volume-container">
            <input id="volume-slider"
                   disabled={isDisabled}
                   type="range"
                   min="-2"
                   max="102"
                   onChange={(e) => handleChange(e)}
                   value={value}/>
        </div>
    )
}
```

Obrázek 76 - Implementace komponentu VolumeSlider (zdroj. autor)

```
const setVolume = (value: number) => {
    if (!store.getState().state.currentPlayer) return
console.error(initializationError)

    if (store.getState().state.currentPlayer === Merger.PlayerType.Spotify)
{
        if (!SDK) throw new Error(initializationError)

        SDK.setVolume(value / 100);
        return;
    }

    if (window.youtubePlayer !== undefined) {
        window.youtubePlayer.setVolume(value);
        return;
    }

    throw new Error(initializationError)
}

return (
    <div id="player">
        <InfoPlayerContainer />
        <div>
            <div id="player-buttons-container">
                <PlayerButton disabled={!merger.State.state.previousSong}
id="prev-button"
                    src="/images/PrevButton.png" execFunc={merger.PrevSong} />
                <PlayerButton disabled={false} src={merger.State.state.paused ?
stateImg.play : stateImg.pause}
                    text="Toggle Play"
                    id="play-button" execFunc={merger.TogglePlayBack} />
                <PlayerButton disabled={!merger.State.state.nextSong} id="next-
button"
                    src="/images/NextButton.png" execFunc={merger.NextSong} />
            </div>
            <ProgressBar func={setProgress} />
        </div>
        <div>
            <Link to={"/queue"}>
                <img src="/images/queueIcon.png" alt="error" />
            </Link>
        </div>
        <VolumeSlider isDisabled={!store.getState().state.currentPlayer}
func={setVolume} />
    </div>
);
```

Obrázek 77 - Kód pro zavolání SetVolume v komponentu Player. Zdroj autor

8.1.8.4 Ovládání playbacku

Proces vykonání funkce pro ovládání playbacku probíhá podobným způsobem jako u nastavení hlasitosti. Pro řízení playbacku jsou implementovány komponenty PlayerButton a ProgressBar. Pomocí tlačítek lze přejít na další nebo předchozí skladbu a pozastavovat nebo přehrávat hudbu. Ukazatel průběhu se stará o přetáčení playbacku.

```
const setProgress = async (value: number) => {
    if (mergerState.state.currentPlayer !== undefined) {
        if (mergerState.state.currentPlayer === Merger.PlayerType.Spotify) {
            if (SDK) {
                SDK.seek(value);
                updatePlaybackState();
                return;
            }
            throw new Error(initializationError)
        }

        if (window.youtubePlayer) {
            window.youtubePlayer.seekTo(Math.round(value / 1000), true);
            return;
        }
    }
    throw new Error(initializationError);
}

return (
    <div id="player">
        <InfoPlayerContainer />
        <div>
            <div id="player-buttons-container">
                <PlayerButton disabled={!mergerState.state.previousSong}
id="prev-button"
                    src="/images/PrevButton.png" execFunc={mergerPrevSong} />
                <PlayerButton disabled={false} src={mergerState.state.paused ?
statelmg.play : statelmg.pause}
                    text="Toggle Play"
                    id="play-button" execFunc={mergerTogglePlayBack} />
                <PlayerButton disabled={!mergerState.state.nextSong} id="next-
button"
                    src="/images/NextButton.png" execFunc={mergerNextSong} />
            </div>
            <ProgressBar func={setProgress} />
        </div>
        <div>
            <Link to={"/queue"}>
                <img src="/images/queueIcon.png" alt="error" />
            </Link>
        </div>
        <VolumeSlider isDisabled={!store.getState().state.currentPlayer}
func={setVolume} />
    </div>
);
```

Obrázek 78 - Implementace ovládání playbacku. Zdroj autor

```
export const mergerTogglePlayBack = () => {
    let state: Merger.PlayerState = store.getState().state;
    if (state.currentPlayer !== undefined) {
        if (state.currentPlayer === Merger.PlayerType.Spotify) {
            if (!state.paused) return spotifyPause();
            return spotifyPlay();
        }
        if (!state.paused) return window.youtubePlayer.pauseVideo();
        return window.youtubePlayer.playVideo();
    }
    throw new Error(initializationError);
}
```

Obrázek 79 - Funkce pro přepínání playbacku v mergerUtils.tsx. Zdroj autor

8.1.9 Implementace stránky pro sjednocení dvou seznamů skladeb

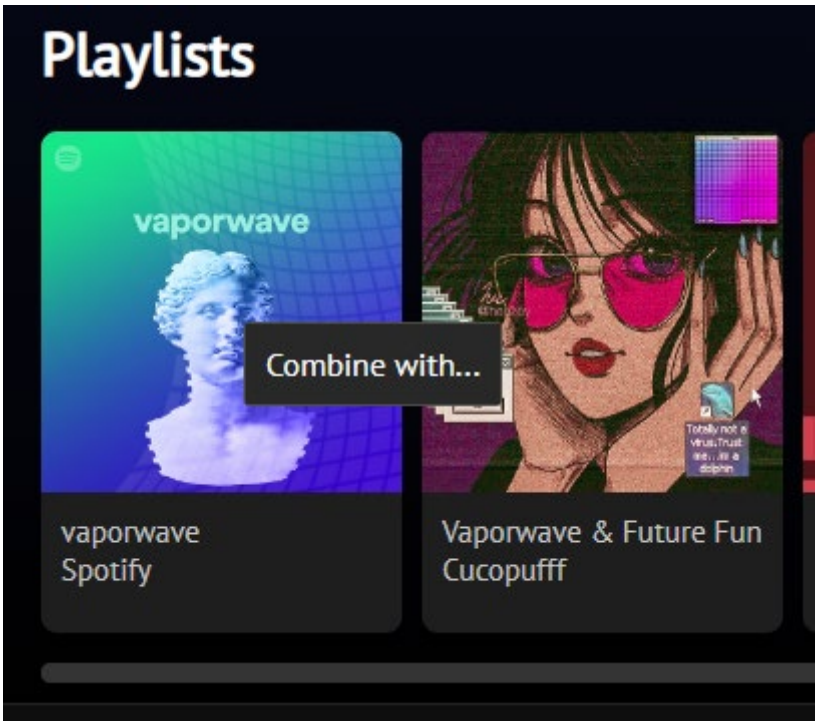
Dalším zaměřením této bakalářské práce zahrnuje implementaci mechanismu pro spojení dvou seznamů skladeb ze Spotify a YouTube. Pokud uživatel chce spojit vybraný seznam skladeb s druhým, musí být vhodným způsobem implementována navigace na stránku spojení dvou seznamů skladeb. Jako nejvhodnější způsob je zavedení kontextového menu. Kontextové menu je seznam tlačítek pro zobrazení různých akcí, které jsou možné vykonat nad objektem kliknutý pravým tlačítkem myši. Toto menu lze implementovat pomocí knihovny React Context Menu.

```
export const PlaylistBlock: React.FC<Props> = ({ playlist }: Props) => {

    const history = useHistory();

    const mergeWithPlaylist = () =>
history.push(`/merger/mergeWithYoutube/${playlist.id}`)

    return (
        <>
            <ContextMenuTrigger id={`playlist-${playlist.id}`}>
                <div className="playlist-block">
                    <img src={playlist.images[0] ? playlist.images[0].url :
"/images/noteimg.png"} alt="Error loading!"></img>
                    <div>
                        <Link className='playlist-title'
to={`/spotify/playlist/${playlist.id}`}>{trimString(playlist.name,
22)}</Link>
                        <div>
                            {playlist.owner.display_name}
                        </div>
                    </div>
                </div>
            </ContextMenuTrigger >
            <ContextMenu id={`playlist-${playlist.id}`}>
                <MenuItem onClick={mergeWithPlaylist}>
                    Combine with...
                </MenuItem>
            </ContextMenu>
        </>
    )
}
```

Obrázek 80 - Implementace React Context Menu v PlaylistBlock.tsx. Zdroj autor

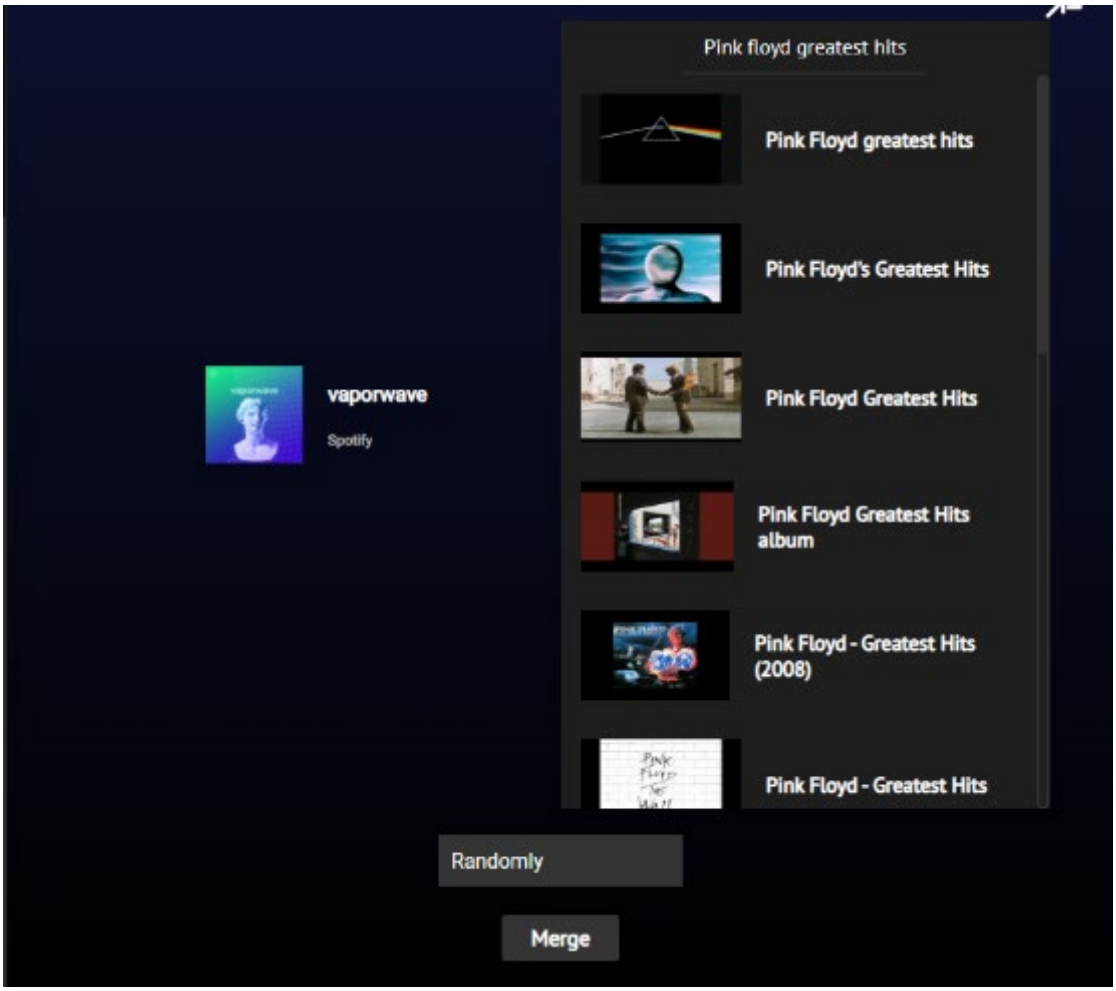
Při implementování React Context Menu je nutné implementovat dva komponenty typu ContextMenuTrigger a ContextMenu. ContextMenuTrigger slouží jako spouštěč pro zobrazení kontextového menu neboli komponentu typu ContextMenu. Protože se komponenty do sebe nevnořují, musí mít komponenty ContextMenu a ContextMenuTrigger zavedený stejné id. Tim je zajištěna jejich propojenost a následné zobrazení kontextové menu. Pro zobrazení jednotlivých tlačítek menu slouží komponent typu MenuItem, který se vkládá jako potomek do ContextMenu.

Pro navigaci na stránku spojení dvou seznamů skladeb je implementováno tlačítko, které po kliknutí zavolá funkci mergeWithPlaylist a uživatel je přesměrován na požadovanou stránku. Přesměrování na určitou stránku je dosaženo pomocí React hooku useHistory.



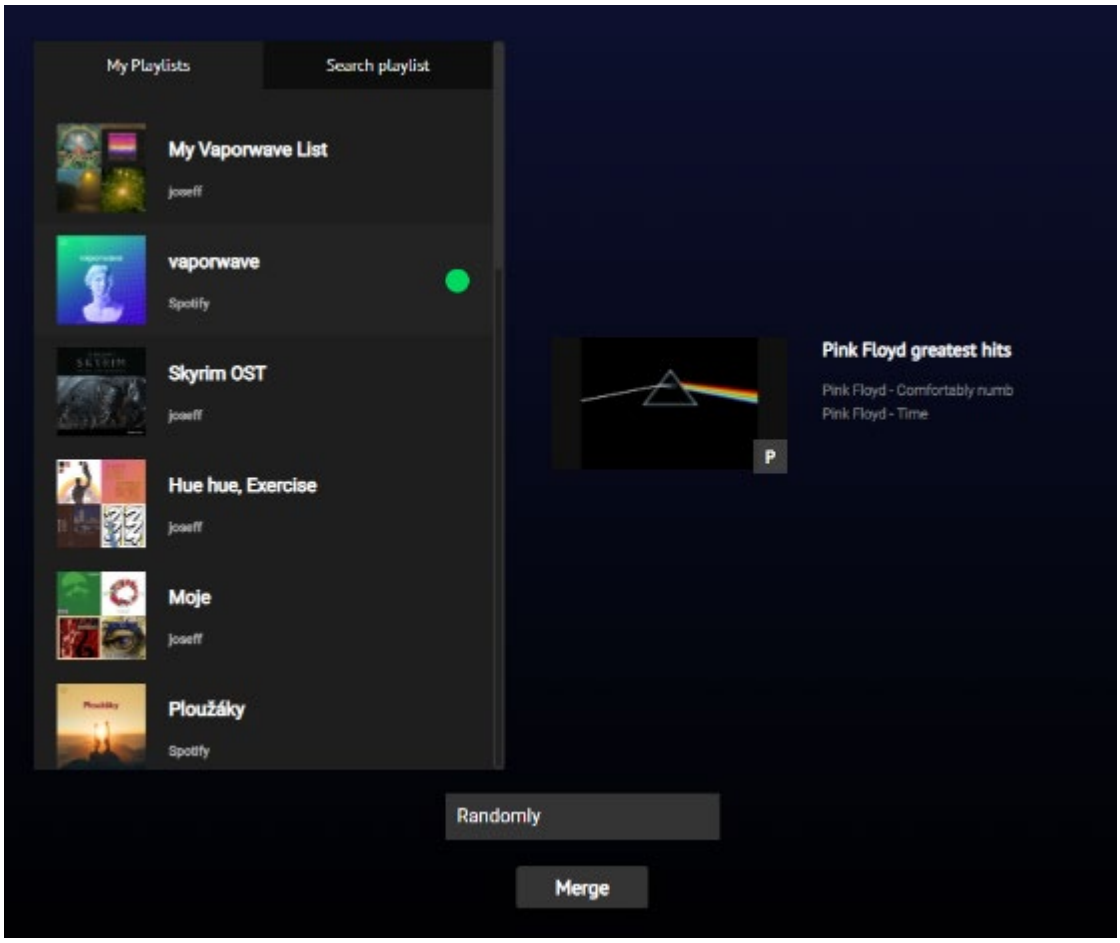
Obrázek 81 - Zobrazení kontextového menu. Zdroj autor

Po navigaci je uživatel prezentován stránkou, která zobrazuje jeho vybraný seznam skladeb a oknem pro vyhledávání seznamů skladeb. U Spotify je přídavkem v tomto okně zobrazení sledovaných nebo uložených seznamů skladeb přihlášeného uživatele.



Obrázek 82 - Zobrazení stránky pro spojení seznamů skladeb YouTube a Spotify. Zdroj autor

Pokud uživatel vybere jednu položku ze seznamu, označí se uživatelův výběr jako vybraná položka pro spojení. Položky v seznamu fungují jako přepínače.

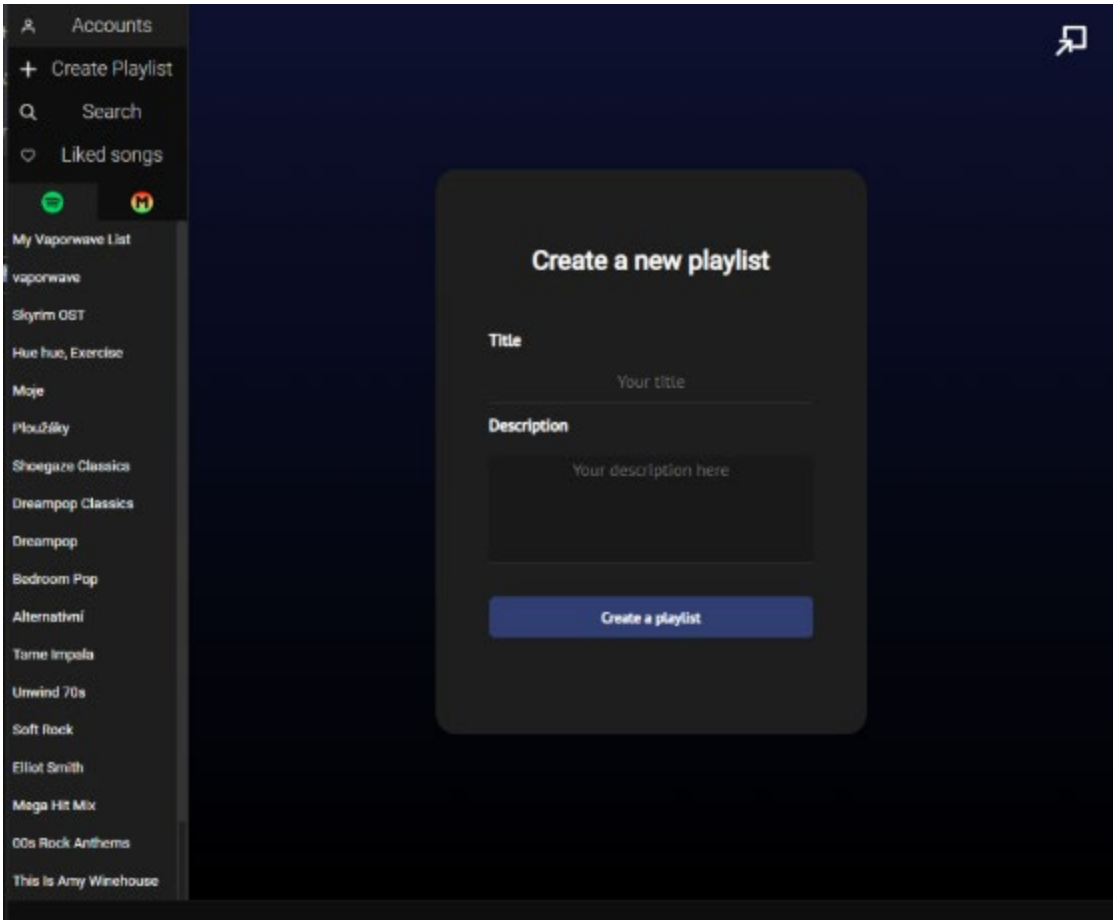


Obrázek 83 - Zobrazení stránky pro spojení seznamů skladeb Spotify s YouTube. Zdroj autor

Nad tlačítkem „Merge“ je uživateli nabízeno jakým způsobem lze tyto dva seznamy skladeb spojit. Náhodné vkládání skladeb do seznamu je výchozí volbou. Pokud uživatel chce, lze zařadit Spotify skladby jako první a YouTube skladby jako poslední nebo naopak. Po stisknutí tlačítka „Merge“ je vytvořen nový seznam skladeb a uživatel je přesměrován automaticky na stránku nově vytvořeného seznamu skladeb.

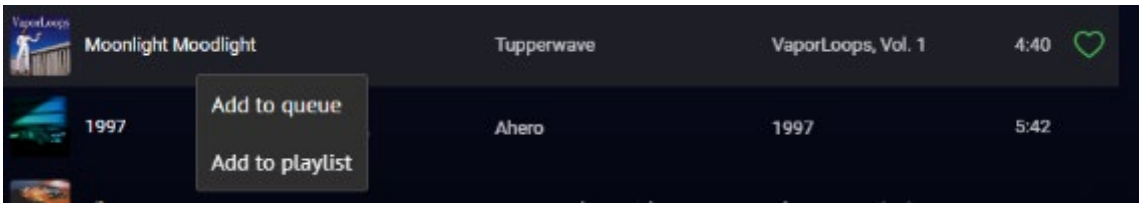
8.1.10 Implementace stránky pro vytvoření seznamu skladeb

Pokud je uživatel přihlášen, lze vytvořit nový seznam skladeb do kterého může vkládat písničky ze Spotify nebo YouTube.

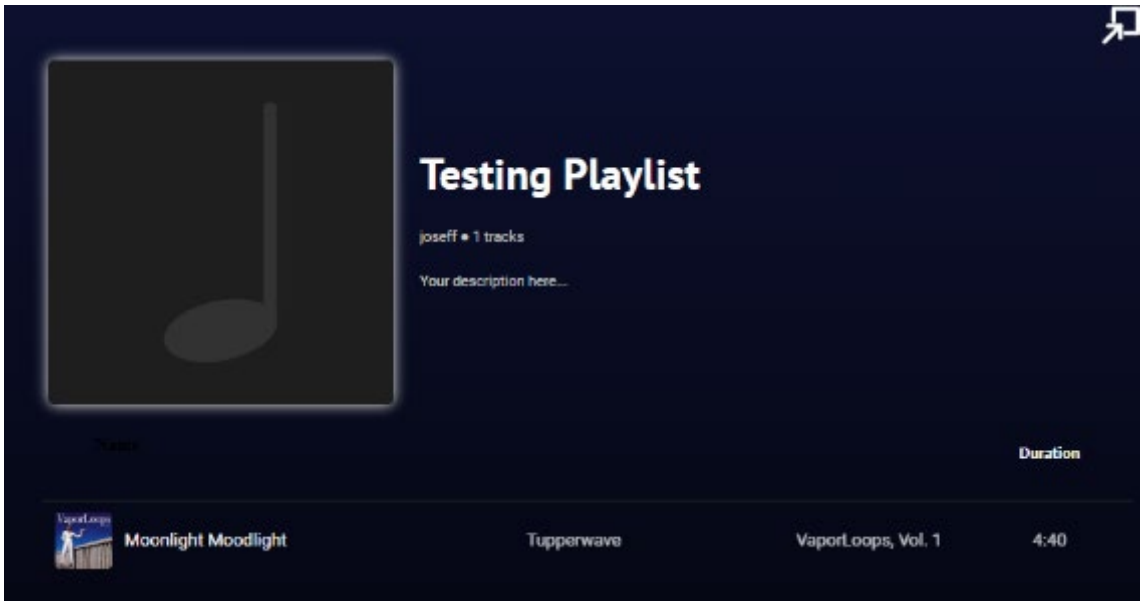


Obrázek 84 - Stránka pro vytvoření nové seznamu skladeb. Zdroj autor

Uživatel může navigovat na tuto stránku pomocí tlačítka „Create Playlist“. Na této stránce je zobrazen formulář pro vyplnění názvu seznamu skladby a jeho popisu. Po jeho vytvoření lze vybrat jednu skladbu od Spotify nebo YouTube a pomocí kontextového menu přidat do vytvořeného seznamu.



Obrázek 85 - Zobrazení kontextové menu nad skladbou. Zdroj autor



Obrázek 86 - Vložená skladba v seznamu. Zdroj autor

8.1.11 Implementace stránky oblíbených skladeb a fronty

Tyto stránky se vzhledově velmi podobají. Na druhou stranu je jejich funkcionalita odlišná. Oblíbené skladby uživatele jsou uloženy permanentně v databázi, zatím co u fronty je zobrazeno uživateli momentálně nasazena fronta skladeb. Tato fronta se může dynamicky měnit dle přehrávaného obsahu.

Obrázek 87 - Zobrazení stránky pro frontu skladeb. Zdroj autor

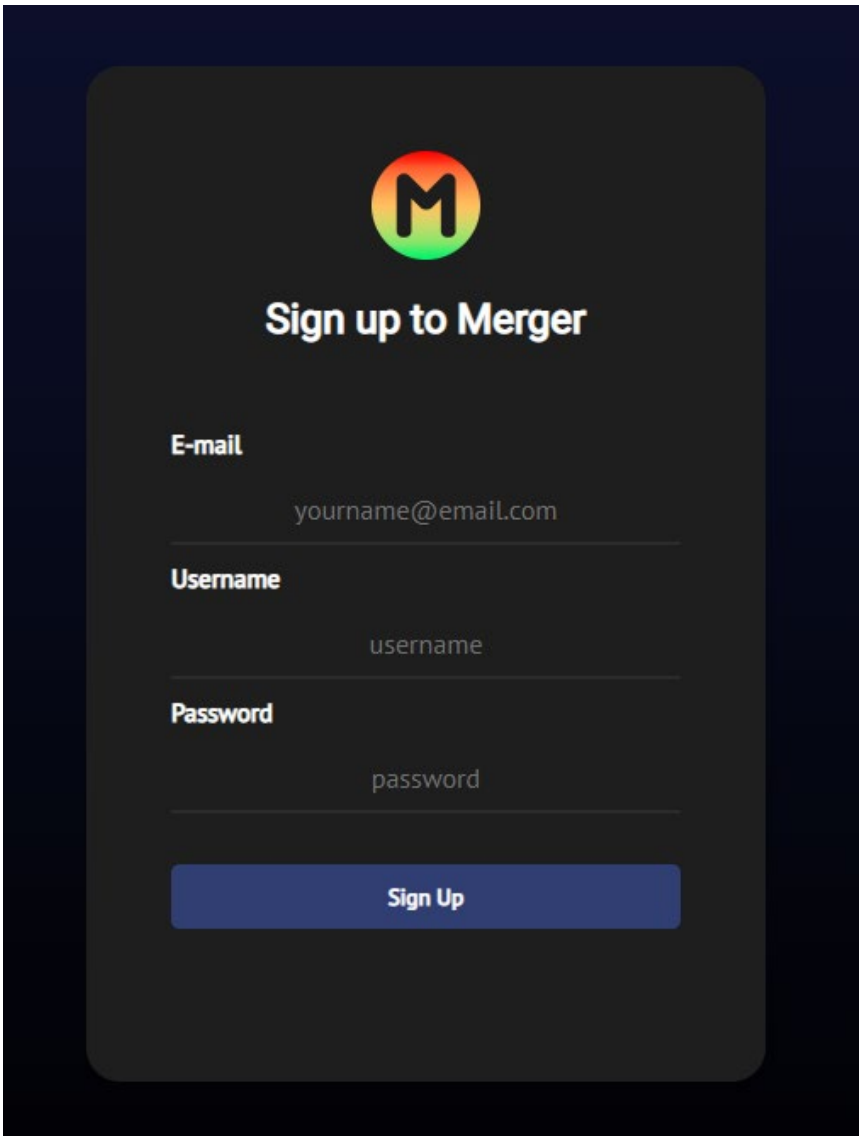
```
export const QueuePage: React.FC = () => {

    const mergerQueue: Merger.Queue = useAppSelector(rootState).queue;

    const handlePlay = (track: SpotifyApi.TrackObjectSimplified |
gapi.client.youtube.Video) => {
        if (isSpotifyTrackObject(track)) {
            addOtherSongsToQueuePlaylist(track.uri, mergerQueue.queue);
        } else if (track.id) {
            addOtherSongsToQueuePlaylist(track.id, mergerQueue.queue);
        } else {
            return console.error("Can not play! Id is undefined!");
        }
        return mergerLoadAndPlay(track);
    }

    const generateRows = (value: SpotifyApi.TrackObjectFull |
gapi.client.youtube.Video,
                          index: number): JSX.Element | null => {
        if (index >= mergerQueue.counter) {
            if (isSpotifyTrackObjectFull(value))
                return <SpotifyTrackRow showLike={true} album={value.album}
img={value.album.images[2].url}
                                        showArtist={true}
onClick={handlePlay} track={value}
                                        key={generateRandomString(14)}/>

            return <YoutubeTrackRow showLike={true}
key={generateRandomString(14)} video={value}
                                    handleOnClick={handlePlay}/>
        }
        return null;
    }

    useEffect(() => {

    }, [mergerQueue.queue, mergerQueue.counter])

    return (
        <div id="queue-page">
            <h1>Queue</h1>
            <div id="playlist-table">
                <TrackListHeader/>
                {
                    mergerQueue.queue &&
mergerQueue.queue.map(generateRows)
                }
            </div>
        </div>
    )
}
}
```

Obrázek 88 - Implementace komponentu typu QueuePage. Zdroj autor

Tento komponent sleduje obsah fronty, který je uložen v globálním stavu. Pokud zaznamená změnu ve frontě, skladby zobrazené na stránce se aktualizují a vykreslí.

8.1.12 Implementace stránky pro registraci a přihlášení

Jak již bylo zmíněno při navrhování těchto stránek, tento komponent obsahuje formulář, do které uživatel zadá potřebné údaje. Formulář se skládá z tlačítka a z textových polí, které jsou reprezentovány komponentem TextField. Po stisknutí tlačítka uživatelem je poté zavolána funkce pro registraci nebo přihlášení uživatele.



Obrázek 89 - Zobrazení stránky s formulářem pro registraci. Zdroj autor

```
export const RegisterPage: React.FC = () => {
    const [email, setEmail] = useState<string>("");
    const [password, setPassword] = useState<string>("");
    const [username, setUsername] = useState<string>("");
    const [errorMsg, setErrorMsg] = useState<string>();

    const tryRegistering = (event: FormEvent<HTMLFormElement>) => {
        event.preventDefault();
        axios.put(`${process.env.REACT_APP_API_LINK}/merger/register`, {
            username,
            email,
            password
        }).catch((err) => {
            if (err.response.status as number === 400) return
setErrorMsg(err.response.data.message);
            return console.error(err);
        })
    }

    return (
        <div id="form-window">
            <img src="/images/mergericon.png" alt="Merger icon"/>
            <h2>Sign up to Merger</h2>
            <form onSubmit={tryRegistering}>
                <h6>E-mail</h6>
                <TextField placeholder={"yourname@email.com"} type={"text"}
value={email} onChange={setEmail}/>
                <h6>Username</h6>
                <TextField placeholder={"username"} type={"text"}
value={username} onChange={setUsername}/>
                <h6>Password</h6>
                <TextField placeholder={"password"} type={"password"}
value={password} onChange={setPassword}/>
                <input id="submit-btn" type="submit" value="Sign Up"/>
            </form>
            <div style={errorMsg ? {display: "block"} : {display: "none"}}
id="response-box">
                <h4>{errorMsg}</h4>
            </div>
        </div>
    )
}
```

Obrázek 90 - Implementace komponentu pro registraci RegisterPage.tsx. Zdroj autor

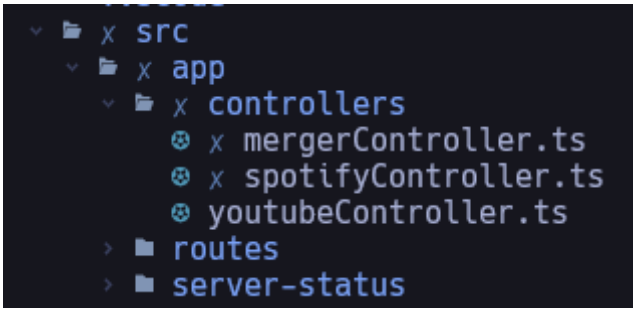
8.2 Implementace Serveru

Do této doby bylo možné získávat data pouze ze strany klienta. V tento moment je vhodné vytvořit nový server, který nám bude získávat data z YouTube a Spotify skrze automatizovanou autorizaci pomocí OAuth 2.0. Do tohoto serveru se implementují určité knihovny, které zajistí komunikaci serveru s těmito API. Jelikož server bude zprovozněn v prostředí Node.js, lze využít mnoho frameworků, které usnadní a urychlí jeho implementaci. Mezi vhodné a nejpopulárnější frameworky, ve kterém se server bude vyvíjet, patří Express.js. Celý server je nahraný na GitHub. [13]

8.2.1 Vytváření řadičů

Jelikož je potřeba implementovat vlastní API a získávat data ze dvou různých platforem, server je rozdělen na tři řadiče:

- Spotify řadič – Tento řadič se stará o získávání dat ze Spotify API.
- YouTube řadič – Úkolem tohoto řadiče je získávat data z YouTube Data API.
- Merger řadič – Slouží jako API specificky pro tuto aplikaci.



Obrázek 91 - Soubory pro řadiče serveru. Zdroj autor

8.2.2 Zprovoznění YouTube Data API a Spotify API

8.2.2.1 Spotify API

Před přípravou implementace přehrávačů již bylo zmíněno, že při registraci aplikace je vývojáři poskytnuto Client ID a Client Secret. Jelikož tyto údaje byli uložené v kódu klientské aplikace, nyní je lze přesunout na server a uložit je jako proměnné prostředí. Tyto hodnoty jsou uloženy v souboru process.env. K těmto údajům je přidána adresa přesměrování, která byla definována při registraci a nastavení aplikace.

Pro implementaci Spotify API se použije knihovna spotify-web-api-node, která slouží jako jeho obal (wrapper). Pro inicializaci Spotify API stačí vytvořit novou instanci SpotifyWebApi, do které je povinné předat objekt typu Credentials. Tento objekt musí obsahovat Client ID, Client Secret a adresu přesměrování. Tyto údaje jsou později použity pro autorizaci.

```
const spotifyApi: SpotifyWebApi = new SpotifyWebApi({
    clientId: process.env.SPOTIFY_CLIENT_ID,
    clientSecret: process.env.SPOTIFY_CLIENT_SECRET,
    redirectUri:
`http://localhost:${process.env.PORT}/spotify/auth/callback`
})
```

Obrázek 92 - Inicializace Spotify API. Zdroj autor

Spotify API zavádí tři různé způsoby, jakými lze udělit přístupová oprávnění aplikaci nebo uživateli pro komunikaci se Spotify API:

1. Autorizace na straně klienta – Autorizace probíhá na straně klienta a nezahrnuje sdílení tajných klíčů. Tudíž není potřeba vytvářet na straně serveru kód pro autorizaci. Nevýhodou této autorizace je generace přístupových tokenu, které mají krátkodobou platnost. [14]

2. Autentizace mezi servery – Tento způsob nezahrnuje autorizaci, tudíž server nemůže přistupovat k údajům uživatelů. [15]

3. Dlouhodobá autorizace – Způsob autorizace určený pro dlouhotrvající aplikace, kde uživatel uděluje aplikaci oprávnění jenom jednou. Tento způsob autorizace bude implementovaný ve vyvíjeném serveru. [16]

Prvním krokem pro implementaci autorizace je definice rozsahů oprávnění. Tyto rozsahy oprávnění informují uživatele při přihlášení, k jakým druhům dat daná aplikace požaduje přístup jako např. přístup k osobním údajům, ke sledovaným nebo vytvořeným seznamům skladeb uživatelem a další.

Po určení rozsahů oprávnění lze přesměrovat uživatele na přihlašovací okno Spotify, do kterého zadá potřebné údaje.

```
export const login = (req: express.Request, res: express.Response) => {
    if (spotifyApi.getCredentials().clientId != undefined) {
        const scope = "streaming \
                       user-read-email \
                       user-read-private \
                       user-read-playback-state \
         playlist-read-private \
         playlist-read-collaborative"

        const state = generateRandomString(32);

        const auth_query_parameters = new URLSearchParams({
            response_type: "code",
            scope: scope,
            redirect_uri: spotifyApi.getCredentials().redirectUri as
string,
            state: state,
            client_id: spotifyApi.getCredentials().clientId as string,
        })

        res.redirect('https://accounts.spotify.com/authorize/?' +
auth_query_parameters.toString());
        res.end();
        return;
    }
    res.send("Invalid Spotify Client ID!");
    res.end();
}
```

Obrázek 93 - Implementace funkce pro přihlašování ve Spotify řadiči. Zdroj autor

Dle kódu lze pozorovat v proměnné scope definované rozsahy oprávnění. Před přesměrováním uživatele jsou definovaný URL parametry potřebné pro správnou autorizaci:

- Client ID
- Adresu pro přesměrování
- Typ autorizace
- Rozsahy oprávnění
- State – Tento parametr není povinný, ale velice doporučený. Poskytuje protekci proti útokům typu jako padělaní požadavků napříč stránkami.

Upřesněné parametry jsou poté vložený do URL adresy pro autorizaci uživatele u Spotify.

Pokud se uživatel úspěšně přihlásí, je přesměrován na adresu, která byla dříve definována v nastavení aplikace nebo je uložená v souboru process.env.

```
export const authCallback = (req: express.Request, res: express.Response,
err: express.Errback) => {

    if (req.query.error === undefined) {
        const code = req.query.code;

        const authOptions = {
            url: 'https://accounts.spotify.com/api/token',
            form: {
                code: code,
                redirect_uri: spotifyApi.getCredentials().redirectUri,
                grant_type: 'authorization_code',
            },
            headers: {
                'Authorization': 'Basic ' + (Buffer.from(
                    spotifyApi.getCredentials().clientId + ':' + spoti-
fyApi.getCredentials().clientSecret)
                                                    .toString('base64')),
                'Content-Type': 'application/x-www-form-urlencoded'
            },
            json: true
        };

        request.post(authOptions, (error: Error, response: express.Re-
sponse, body: any) => {
            if (!error && response.statusCode === 200) {
                spotifyApi.setCredentials({...spotifyApi.getCredentials(),
accessToken: body.access_token});
                spotifyApi.setRefreshToken(body.refresh_token);
                refreshInterval = setInterval(refreshSpotifyToken,
(body.expires_in-100) * 1000);
                res.redirect('http://localhost:3000/');
                res.end();
            } else {
                res.status(500);

                res.send({
                            message: "Trying to obtain the access token
failed!",
                            stacktrace: err.prototype.stacktrace
                        } as merger.Error)

                res.redirect('http://localhost:3000/');
                res.end();
            }
        });
    }
```

Obrázek 94 - Získání přístupového a obnovujícího tokenu od Spotify. Zdroj autor

V tento moment může server požádat Spotify API o přístupový a obnovující token, který server poté uloží na vhodné místo. Protože byl získán přístupový token, serveru je povoleno získávat data od Spotify. Přístupový token platí po dobu jedné hodiny. Pokud platnost přístupového tokenu vyprší, lze ho znovu vygenerovat pomocí obnovujícího tokenu. Pro automatizaci obnovení lze implementovat JavaScriptový časovač, který každou hodinu obnovuje a získává přístupový token.

```
export const refreshToken = () => {
    clearInterval(refreshInterval);
    spotifyApi.refreshAccessToken().then((spRes) => {
        if (spRes.body.refresh_token)
            spotifyApi.setRefreshToken(spRes.body.refresh_token);

        spotifyApi.setAccessToken(spRes.body.access_token);

        refreshInterval = setInterval(refreshSpotifyToken,
(spRes.body.expires_in-1000) * 1000);
    }).catch((err) => {
        console.error("failed to refresh the token!", err);
    })
}
```

Obrázek 95 - Kód pro automatizaci obnovení přístupového tokenu. Zdroj autor

8.2.3 Použití Spotify A YouTube API

Jakým způsobem se získávají data od těchto API se implementací velmi podobají. Instance API nabízí funkce pro čtení dat, který vývojář potřebuje např. vyhledávání, čtení skladeb nebo videí, hledání umělců nebo kanálu atd. Jelikož se v tomto případě YouTube API obsluhuje pomocí API klíče, je povinností ho u každého požadavku předávat do parametrů. Vývojář se u Spotify API o toto nemusí starat. Přístupový token je obsažen v instanci Spotify API. Tudíž každý odeslaný požadavek v sobě automaticky zahrnuje přístupový token.

```
export const search = (req: express.Request, res: express.Response) => {
    if (req.query.query !== undefined) {

        let types: string[] = ["playlist", "video"];

        if (req.body.types) types = req.body.types;

        const searchQuery: string = req.query.query as string;
        youTube.search.list({
            auth: api,
            part: ['snippet'],
            q: searchQuery,
            maxResults: 15,
            type: types
        }, (ytErr, ytRes) => {
            if (!ytErr) {
                res.send(ytRes?.data).end();
                return;
            }
            res.status(500).send(ytErr).end();
        })
    }
}
```

Obrázek 96 - Příklad použití YouTube Data API na serveru. Zdroj autor

```
export const search = async (req: express.Request, res: express.Response)
=> {

    if (req.query.q !== undefined) {

        try {
            const searchRes = await spotifyApi.search(req.query.q as
string, req.body.types);

            return res.send(searchRes.body);

        } catch (e: unknown) {
            console.error(e)
            return res.status(500).send(createMergerError("Spotify search
has failed!", 500))
        }
    }

    return res.status(204).send(createMergerError("Query is undefined!",
204))
}
```

Obrázek 97 - Příklad použití Spotify API na serveru. Zdroj autor

8.2.4 Implementace vlastního řadiče

Pro účely bakalářské práce je potřeba implementovat následující funkcionality:

- Spojování dvou seznamů
- Vytvoření seznamů
- Přihlašování a registraci
- Přidávání skladby do seznamu skladeb
- Přidávání skladby do oblíbených skladeb

K tomu je potřeba zprovoznit novou databázi pro ukládání těchto dat. Vybraný systém, který se bude používat pro vkládání dat, je MySQL. Jelikož není cílem této práce popsat jakým způsobem se tato databáze zprovozňuje, je předpokládáno že je předem vytvořená a připojená k serveru.

8.2.4.1 Registrace

Na straně klienta při registraci uživatele je vyžadována přezdívka, e-mail a heslo. Po odeslání požadavku pro registraci jsou tyto údaje otestovány pomocí regulárních výrazu. Pokud jsou údaje definované a test je úspěšný, lze přejít na vložení nového uživatele do databáze. Předtím než je uživatel uložen, je z bezpečnostních důvodu důležité zašifrovat heslo. Pro jeho zašifrování je použita knihovna bcrypt. Výhodou bcryptu a jeho šifrování (hašování) je použití tzv. „soli“, která reprezentuje řetězec přidaný k zašifrovanému textovému řetězci pro zvýšení bezpečnosti. Řetězec lze vygenerovat náhodně nebo definovat. Po zašifrování hesla je poté nový uživatel vložen do databáze a následně je mu umožněno se přihlásit.

```
const user: merger.User = req.body as merger.User;

if (!user.password || !testPassword(user.password))
    return res.status(400).send(
        createMergerError("Password wasn't provided or illegal chars have
been used!", 400)
    );

if (!testEmail(user.email) || !testUsername(user.username))
    return res.status(400).send(
        createMergerError("Email or username has not been provided or
prohibited chars have been used!", 400)
    );

const hashedPassword: string = await bcrypt.hash(user.password, 10);

try {
    if (!user.email) throw new Error("Email is undefined!");

    await db.promise().query(queries.insertUser(user.username, user.email,
hashedPassword));

} catch (e: unknown) {
    console.error(e)
    return res.status(500).send(
        createMergerError("Failed to register a user!", 500)
    );
}
```

Obrázek 98 - Kód pro registraci uživatele. Zdroj autor

8.2.4.2 Přihlášování

Při přihlašování se jako u registrace ověří, zda získané údaje jsou validní. Pokud ano, lze pomocí e-mailu získat příslušného uživatele z databáze. Pomocí bcryptu a jeho funkce compare je ověřeno, zda odeslané heslo uživatelem koresponduje se zašifrovaným heslem z databáze. Pokud ano, uživatel je přihlášen.

Jako přídavek je na serveru nainstalovaná knihovna express-sessions. Cílem této knihovny je automatizovat operace s probíhajícími sessiony uživatelů a ukládat je. Tímto lze ověřit, zda uživatel je oprávněn používat určité operace API.

```
if (req.session.authenticated) return res.status(200);

const {email, password} = req.body;

if (!email || !password) return res.status(403).send(createMergerError("Bad
Credentials", 403));

db.promise().query(queries.selectUserByEmail(email)).then(async (sqlRes) =>
{

    const user = (sqlRes[0] as RowDataPacket[])[0] as merger.User;

    if (!user) {
        return res.status(403).send(
            createMergerError("Given email wasn't found! If you don't have
an account please sign up.", 403));
    }

    if (!user.password) {
        return res.status(500).send(createMergerError("Password is
undefined!", 500));
    }

    bcrypt.compare(password, user.password).then(() => {
        req.session.username = user.username;
        req.session.email = user.email;
        req.session.userId = user.id;
        req.session.img = user.img;
        req.session.authenticated = true;

        return res.json(req.session);
    }).catch(() => {
        return res.status(403).send(
            createMergerError("Bad Credentials", 403)
        );
    });

    return;
}).catch((sqlErr) => {
    return res.send(sqlErr);
})
```

Obrázek 99 - Kód pro přihlášení uživatele. Zdroj autor

8.2.4.3 Vytvoření nového seznamu

Před vytvořením nového seznamu skladeb je ověřeno, zda je uživatel přihlášen. Pokud ano, server odešle databázi příkaz, aby vložila nový záznam s předanými údaji na své uložiště. Po úspěšném vykonání tohoto příkazu je zpátky klientské aplikaci odesláno id, které lze využít pro přesměrování uživatele na nově vytvořený seznam.

```
if (!isUserAuthenticated(req.session))
    return res.status(403).send(createMergerError("User is not
authenticated or user id is invalid!", 403));

try {
    const {title, desc} = req.body!;

    const sqlRes: OkPacket = (await db.promise().query(
        queries.insertPlaylist(title, req.session.userId, desc)))[0] as
OkPacket;

    return res.json(sqlRes.insertId);
} catch (e: unknown) {

    console.error(e);
    return res.status(500).send(createMergerError("Failed to create a new
playlist!"));
}
```

Obrázek 100 - Kód pro vytvoření nového seznamu. Zdroj autor

8.2.4.4 Přidávání skladby do seznamu skladeb

Před vykonáním operace je jako vždy ověřeno, zda je uživatel přihlášen. Pokud ano, do databáze se vloží identifikační kód skladby s typem přehrávače a následně je vytvořen další záznam, který vytváří relaci mezi seznamem skladeb a přidanou skladbou.

```
if (!isUserAuthenticated(req.session))
    return res.status(403).send(createMergerError("User is not
authenticated or user id is invalid!", 403));

if (!req.body.playlistId || !req.body.trackId)
    return res.status(400).send(createMergerError("Playlist Id or track
object is undefined!", 400));

try {
    await db.promise()
            .query(queries.insertTrack(req.body.trackId));
    await db.promise()
            .query(queries.insertTrackToPlaylist(req.body.trackId,
req.body.playlistId))

    return res.status(200);

} catch (e: unknown) {
    console.error(e)
    return res.status(500).send(createMergerError("Execution of the query
failed!", 500));
}
```

Obrázek 101 - Kód pro přidání skladby do seznamu skladeb. Zdroj autor

8.2.4.5 Přidávání skladby do oblíbených skladeb

Přidávání skladby do oblíbených je implementováno podobným způsobem jako při přidávání do seznamu skladeb. Po úspěšném ověření uživatele je identifikační kód skladby s jejím typem přehrávače vložen do databáze a následně je vytvořen záznam tvořící relaci mezi skladbou a uživatelem.

```
if (!isUserAuthenticated(req.session))
    return res.status(403).send(createMergerError("User is not authenticated or
user id is invalid!", 403));

try {
    if (!req.body.uri) res.status(401).send(createMergerError("URI is not valid!"))

    await db.promise().query(queries.insertTrack(req.body.uri as string));

    db.promise().query(queries.insertSongToUserData(req.session.userId,
req.body)).then(() => {
        return res.status(200);
    }).catch((err) => {
        console.error("failed to like a track!", err)
        res.status(500).send(createMergerError("failed to like a track!", 500))
    })

} catch (e: unknown) {
    console.error(e);
    return res.status(500).send(createMergerError("Failed to execute a query!",
500))
}
```

Obrázek 102 - Kód pro přidání skladby do oblíbených. Zdroj autor

8.2.4.6 Spojování dvou seznamů skladeb

Spojení dvou seznamů je složené z několika operací. Prvním úkolem je získat všechny skladby ze seznamů skladeb, které byli odeslané klientem. Skladby jsou potřeba získat jak ze Spotify, tak z YouTube. Po získání daných skladeb je vytvořen nový seznam skladeb s názvem Merged Playlist a vložen do databáze. Následně jsou vytvořeny a vykonány příkazy pro uložení skladeb do databáze. Další krok zahrnuje vytvoření relací mezi skladbami a seznamy skladeb. V klientské aplikaci bylo již zmíněno, že si uživatel může vybrat v jakém pořadí tyto písně seřadit. O tento výběr se stará switch, který vykoná příslušnou funkci pro zařazení skladeb v daném pořadí. Po splnění tohoto kroku lze nyní odeslat zpátky klientské aplikaci identifikační číslo nově vytvořeného seznamu skladeb, které použije k přesměrování uživatele.

```
export const merge = async (req: express.Request, res: express.Response) =>
{
    if (!isUserAuthenticated(req.session))
        return res.status(403).send(createMergerError("User is not
authenticated or user id is invalid!", 403));

    if (!req.body.spotifyId && !req.body.youtubeId)
        return res.status(401).send(createMergerError("Youtube or Spotify
playlist id is not defined!"));


    try {
        let order: merger.Order = merger.Order.Random;

        if (req.body.order)
            order = req.body.order as merger.Order;


        const spotifyTracks: Array<SpotifyApi.TrackObjectFull> = (await
spotifyController.getPlaylistById(
            req.body.spotifyId)).tracks.items.map((val) => {
            return val.track
        });

        const youtubeTracks: Array<youtube_v3.Schema$Video> = await
youtubeController.getAllVideosFromPlaylistById(
            req.body.youtubeId);

        const sqlRes = (await
db.promise().query(queries.insertPlaylist("Merged playlist",
req.session.userId)))[0];
        const insertId: number = (sqlRes as OkPacket).insertId;

        let insertQuery = "";

        for (const track of spotifyTracks) {
            insertQuery =
insertQuery.concat(queries.insertTrack(track.uri), "\n");
        }

        for (const video of youtubeTracks) {
            if (!video.id) throw new Error("Video id is not defined!");

            insertQuery = insertQuery.concat(queries.insertTrack(video.id),
"\n");
        }

        await db.promise().query(insertQuery);

        let tracksQuery: string;

        switch (order) {
            case merger.Order.Random:
                tracksQuery = orderRandomly(spotifyTracks, youtubeTracks,
insertId);
                break;
            case merger.Order.SpotifyFirst:
                tracksQuery = orderSpotifyFirst(spotifyTracks,
youtubeTracks, insertId);
                break;
```

Obrázek 103 - Operace pro spojení dvou seznamů. Zdroj autor

```
const orderRandomly = (spTracks: Array<SpotifyApi.TrackObjectFull>,
ytVideos: Array<youtube_v3.Schema$Video>,
                       playlistId: number): string => {

    const totalLength: number = spTracks.length + ytVideos.length;

    let insertIntoPlaylistQuery = "";

    for (let i = 0; i < totalLength; i++) {
        const rnd: number = Math.random();

        if (rnd < 0.5 && spTracks.length > 0) {
            const track = spTracks.pop();
            if (track) {
                insertIntoPlaylistQuery = insertIntoPlaylistQuery.concat(
                    queries.insertTrackToPlaylist(track.uri, playlistId));
                continue;
            }
        }

        if (ytVideos.length > 0) {
            const video = ytVideos.pop();

            if (video?.id)
                insertIntoPlaylistQuery = insertIntoPlaylistQuery.concat(
                    queries.insertTrackToPlaylist(video.id, playlistId));
        }

    }

    return insertIntoPlaylistQuery;
}
```

Obrázek 104 - Algoritmus pro náhodné seřazení skladeb v seznamu. Zdroj autor

8.2.4.7 Získávání dat pro zobrazení seznamu skladeb

Aby byl uživateli zobrazen vytvořený seznam skladeb, je zapotřebí získat tyto data určitým způsobem. Nejdříve se z databáze načte požadovaný seznam a jeho informace. Po získání seznamu skladby je jeho identifikační kód použit pro načtení všech jeho skladeb. Pole identifikačních kódů skladeb se následně předá funkci requestTracks, kde je implementována logika pro získání všech videí z YouTube a skladeb ze Spotify. Posledním krokem zbývá vytvoření nového JSON objektu typu PlaylistFull, který je poté odeslán klientské aplikaci.

```
export const getPlaylist = async (req: express.Request, res:
express.Response) => {

    if (!isUserAuthenticated(req.session)) {
        return res.status(403)
                  .send(createMergerError("User is not authenticated or
user id is invalid!", 403));
    }

    if (!req.params.id) {
        return res.status(500)
                  .send(createMergerError("playlistId wasn't provided!",
403));
    }

    try {
        const playlist = (
            (await db.promise()

.query(queries.selectPlaylistById(parseInt(req.params.id))))[0] as
RowDataPacket[]
        )[0];

        if (!playlist?.id) throw new Error("Playlist id is undefined!");

        const trackUris: Array<merger.Song> = (await db.promise().query(
            queries.selectTracksByPlaylistId(playlist.id)))[0] as
Array<merger.Song>;

        const tracks = await requestTracks(trackUris);

        const playlistObj: merger.PlaylistFull = {
            title: playlist.title,
            creator: {
                id: playlist.creator,
                username: playlist.username
            },
            id: playlist.id,
            desc: playlist.desc,
            tracks
        }

        return res.status(200).send(playlistObj);

    } catch (e: unknown) {
        console.error(e);
        res.status(500).send(createMergerError("Execution of the query
failed!", 500))
    }
}
```

Obrázek 105 - Kód pro čtení seznamu skladeb. Zdroj autor

Jelikož z databáze lze číst jenom identifikační kód skladeb a jejich typ přehrávače, je nutné získat pomocí těchto identifikačních kódu data skladeb od Spotify a YouTube. Jak již bylo zmíněno v předchozím kódu, funkce requestTracks nám právě tyto data poskytne. Algoritmus pro získání těchto dat je znázorněn v následujícím kódu.

```
const requestTracks = async (tracks: Array<merger.Song>):
Promise<Array<SpotifyApi.TrackObjectFull | youtube_v3.Schema$Video>> => {

    let results: Array<SpotifyApi.TrackObjectFull |
youtube_v3.Schema$Video> = [];

    let isType: merger.PlayerType = tracks[0].type;
    let startIndex = 0;
    let endIndex = 1;

    for (let i = 1; i < tracks.length; i++) {
        if (isType === tracks[i].type) {
            endIndex++;
            if (endIndex !== tracks.length) continue;
        }

        if (isType === merger.PlayerType.Spotify) {
            const uris: string[] = tracks.slice(startIndex,
endIndex).map(track => track.uri.split(":")[2]);
            const requestedTracks: Array<SpotifyApi.TrackObjectFull> =
await spotifyController.getTracksByUris(uris);

            results = results.concat(requestedTracks);
            startIndex = i;
            endIndex = i + 1;
            isType = merger.PlayerType.Youtube;
            continue;
        }

        const requestedTracks: Array<youtube_v3.Schema$Video> = (await
youtubeController
            .getYoutubeVideoList(tracks.slice(startIndex, endIndex)
                                       .map(val => val.uri)));

        results = results.concat(requestedTracks);
        startIndex = i;
        endIndex = i + 1;
        isType = merger.PlayerType.Spotify;
    }

    if (results.length === tracks.length) return results;

    if (isType === merger.PlayerType.Spotify) {
        const uris: string[] = tracks.slice(startIndex, endIndex).map(track
=> track.uri.split(":")[2]);

        const requestedTracks: Array<SpotifyApi.TrackObjectFull> = await
spotifyController.getTracksByUris(uris);

        results = results.concat(requestedTracks);
    }

    const requestedTracks: Array<youtube_v3.Schema$Video> = (await
youtubeController
        .getYoutubeVideoList(tracks.slice(startIndex, endIndex)
                                   .map(val => val.uri)));

    results = results.concat(requestedTracks);
```

Obrázek 106 - Algoritmus pro získání dat skladeb. Zdroj autor

Hlavní podmínkou tohoto algoritmu je zachování pořadí skladeb v seznamu. Pokud by toto nebylo dodrženo, docházelo by k jejich náhodnému seřazení. Tudíž tento kód rozděluje pole skladeb, které byli získány ze vstupních parametrů, do souvislých bloku podle přehrávače. Tyto bloky jsou poté použity pro získání dat z API přehrávačů. Pří získávání dat skladeb od Spotify API je ještě potřeba vzít v úvahu jakým způsobem se odešle požadavek.

Pokud Spotify API získá požadavek s přečtením více než padesáti skladeb, tento požadavek odmítne. [17] Je potřeba tento jeden požadavek rozdělit na více požadavků, které nepřesáhnou více než padesát identifikačních kódu skladeb. Tento problém rovnou řeší funkce getTracksByUris.

```
export const getTracksByUris = async (uris: Array<string>):
Promise<Array<SpotifyApi.TrackObjectFull>> => {

    /*
       if the array is longer than 50 uris, it has to be
       cut to several counts of requests.
    */
    const countOfRequests: number = Math.ceil(uris.length / 50);

    let results: Array<SpotifyApi.TrackObjectFull> = [];

    for (let i = 0; i < countOfRequests; i++) {
        results = results
                    .concat((await spotifyApi.getTracks(uris.slice(i * 50,
(i + 1) * 50), {})).body.tracks);
    }

    return results;
}
```

Obrázek 107 - Kód pro získání několik skladeb od Spotify. Zdroj autor

9 Závěr

Aplikace, jejíž úkolem bylo integrovat dva přehrávače v jednom, se podařilo vyvinout úspěšně. Při vývoji byli použity určité nástroje k vytvoření aplikace a byly popsány a demonstrovány při implementaci, což bylo hlavním cílem této práce.

Na začátku bylo vysvětleno, jakým způsobem postupovat při vytváření webové aplikace. V první části bylo zkoumáno, jakým způsobem komerční přehrávače fungují a jaké funkcionality nabízí. Na základě těchto funkcionalit byly stanoveny požadavky a funkcionality, které aplikace měla zahrnout. Před implementací aplikace se realizoval návrh celé webové aplikace. Cílem bylo navrhnout uživatelské rozhraní, architekturu aplikace, jak bude fungovat server a co bude databáze aplikace obsahovat.

Po navržení aplikace následovala implementace klientské aplikace. Hlavním zaměřením při vývoji webové aplikace byl popis přípravy a použití příslušných knihoven jako YouTube IFrame API a Web Playback SDK pro vytvoření přehrávače. Následně byly tyto přehrávače integrovány do aplikace a využity v praxi.

Další částí této práce zahrnovala implementaci serveru běžící v prostředí Node.js, který se poté připojil k databázi. Nejdříve se popsalo, jakým způsobem zprovoznit YouTube API a Spotify API na serveru. Následujícím krokem bylo popsání implementace těchto API a ukázka jejich použití v praxi.

Ke konci byla na serveru implementována vlastní API. Byly také popsány jednotlivé funkce, které aplikace využívá jako např. registrace, přihlašování, přidávání skladeb do seznamu skladeb a další.

Nástroje, které byly použity při vytvoření aplikace, jsou vskutku užitečné. Jsou veřejnosti dostupné a kdokoliv chce, může si vyzkoušet implementovat vlastní klon přehrávače. Ačkoliv existuje pár případu, které mohou vývojáře zarazit, nástroje jsou robustní a plní dobře svoji funkci.

10 Seznam použité literatury

[1] *Application Program Interface from FOLDOC* [online]. [vid. 2022-01-31]. Dostupné z: http://foldoc.org/Application+Program+Interface

[2] *What is an Application Programming Interface (API)* [online]. 15. říjen 2021 [vid. 2022-01-31]. Dostupné z: https://www.ibm.com/cloud/learn/api

[3] *YouTube Player API Reference for iframe Embeds | Rozhraní API pro iframe | Google Developers* [online]. [vid. 2022-01-31]. Dostupné z: https://developers.google.com/youtube/iframe_api_reference?hl=cs

[4] *YouTube Data API Overview | Google Developers* [online]. [vid. 2022-01-31]. Dostupné z: https://developers.google.com/youtube/v3/getting-started?hl=cs

[5] Definition of SDK (Software Development Kit) - Gartner Information Technology Glossary. *Gartner* [online]. [vid. 2022-01-31]. Dostupné z: https://www.gartner.com/en/information-technology/glossary/sdk-software-development-kit

[6] Co je jednostránková aplikace (SPA) a kdy ji využít? *Rascasone.com* [online]. [vid. 2021-07-18]. Dostupné z: https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa

[7] *Casgen/merger-client* [online]. [vid. 2022-08-15]. Dostupné z: https://github.com/Casgen/merger-client

[8] KRILL, Paul. React: Making faster, smoother UIs for data-driven Web apps. *InfoWorld* [online]. 15. květen 2014 [vid. 2022-01-31]. Dostupné z: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html

[9] MÁCA, Jindřich. *Lekce 1 - Úvod do React* [online]. [vid. 2021-07-18]. Dostupné z: https://www.itnetwork.cz/uvod-do-react

[10] *My Dashboard | Spotify for Developers* [online]. [vid. 2022-08-10]. Dostupné z: https://developer.spotify.com/dashboard/

[11] *Quick Start | Spotify for Developers* [online]. [vid. 2022-08-07]. Dostupné z: https://developer.spotify.com/documentation/web-playback-sdk/quick-start/

[12] *Redux Essentials, Part 1: Redux Overview and Concepts | Redux* [online]. [vid. 2022-08-08]. Dostupné z: https://redux.js.org/tutorials/essentials/part-1-overview-concepts

[13] *Casgen/merger-api* [online]. [vid. 2022-08-15]. Dostupné z: https://github.com/Casgen/merger-api

[14] *Implicit Grant Flow | Spotify for Developers* [online]. [vid. 2022-08-10]. Dostupné z: https://developer.spotify.com/documentation/general/guides/authorization/implicit-grant/

[15] *Client Credentials Flow | Spotify for Developers* [online]. [vid. 2022-08-10]. Dostupné z: https://developer.spotify.com/documentation/general/guides/authorization/client-credentials/

[16] *Authorization Code Flow | Spotify for Developers* [online]. [vid. 2022-08-10]. Dostupné z: https://developer.spotify.com/documentation/general/guides/authorization/code-flow/

[17] *Web Api Reference | Get Several Tracks* [online]. Dostupné z: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-tracks

11 Seznam obrázku

12 Zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Vítězslav Zotov**

Studium: I1900278

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Integrace streamovacích platforem ve webové aplikaci**

Název bakalářské práce AJ: Integration of streaming platforms in a web application

Cíl, metody, literatura, předpoklady:

Cílem práce je integrovat dvě streamovací platformy do jedné aplikace a vyhodnotit dostupné technologie, které budou využity pro její vývoj. Aplikace umožní uživatelům přehrávat obsah jednotným způsobem nezávisle na platformě.

Následně je účelem práce demonstrovat použití prostředků a nástrojů, které aplikují potřebné funkce k provozu této aplikace a navrhnout, jakým způsobem tyto funkce implementovat (Youtube API, Spotify API a Web Playback SDK)

Osnova:

1. Úvod
2. Cíl práce, metodika
3. Stanovení požadovaných funkcionalit
4. Identifikace možností integrace platforem
5. Vyhodnocení možností
6. Struktura Projektu
7. Uživatelské Rozhraní
8. Front-End
9. Back-end
10. Shrnutí
11. Závěr a doporučení

Co je jednostránková aplikace (SPA) a kdy ji využít? *Rascasone.com* [online]. [vid. 2021-07-18]. Dostupné z: https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa

YouTube Data API Overview | Google Developers [online]. [vid. 2021-07-18]. Dostupné z: https://developers.google.com/youtube/v3/getting-started?hl=cs

YouTube Player API Reference for iframe Embeds. *Google Developers* [online]. [vid. 2021-07-18]. Dostupné z: https://developers.google.com/youtube/iframe_api_reference?hl=cs

Web Playback SDK | Spotify for Developers [online]. [vid. 2021-07-18]. Dostupné z: https://developer.spotify.com/documentation/web-playback-sdk/

Zadávající pracoviště: Katedra informačních technologií, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Oponent: Ing. Milan Kořínek

Datum zadání závěrečné práce: 15.10.2021